

UAKGQuery: Руководство программиста

January 30, 2002

DocumentID: GradSoft-PR-16/06/2000-1.5

Contents

1	Введение	3
1.1	Историческая справка:	3
1.2	Предварительные сведения	3
2	Общее описание механизма работы	3
3	Процесс подключения к БД	4
4	Пример выполнения простых запросов	5
5	Типы данных	7
5.1	FieldValue	7
5.1.1	FieldValueAccess	9
5.2	Record, RecordSeq	10
5.2.1	RecordAccess	10
5.2.2	SRecord	11
5.3	OctSeq, RCHheader	11
5.3.1	RCReader	11
5.3.2	RCWriter	13
5.4	RecordDescription	13
6	Выполнение запросов	14
6.1	Запросы без параметров	14
6.2	Batch выполнение запросов с параметрами	16
7	Интерфейс Query	16
7.1	Prepared Query	21
7.2	Интерактивное извлечение данных	22
7.3	Получение описания	22
8	Большие объекты (Large Objects)	23

9	Интерфейс QueryManager	27
9.1	Последовательности	27
10	Свойства	27
11	Коллекции	27
11.1	Введение	27
11.2	Общее описание механизма работы	28
11.2.1	"Физический смысл" коллекции	28
11.2.2	Порядок работы с коллекцией	28
11.3	Создание Query Collections	29
11.4	Доступ к данным	29
11.4.1	Получение данных при помощи методов объекта Iterator	29
11.4.2	Получение данных при помощи собственных методов коллекции	30
11.4.3	Отбор путем сопоставления с образцом	31
11.4.4	Добавление, изменение и удаление данных	32
11.5	Запросы к коллекции	33
11.5.1	Ограничения	34
11.5.2	Список методов	34
11.6	Подколлекции (subcollections)	35
11.7	UAKGKeyCollection	36
11.7.1	Создание UAKGKeyCollection	36
11.7.2	Работа с UAKGKeyCollection	37
11.8	Использование UAKGCollectionListeners	38
11.9	Ограничения	39
12	Транзакции	39
12.0.1	XA транзакции	39
12.0.2	UAKG OTS транзакции	40
12.1	Типичное использование транзакций	40
13	Приложение 1: IDL спецификации	41
13.1	CosQueryCollection	41
13.2	CosQuery	47
13.3	UAKGQuery	48
13.4	RC.idl	67
14	Приложение 2:	76
15	Приложение 3 Грамматика SQL для коллекций:	78
16	Перечень изменений	85

1 Введение

UAKGQuery представляет собой CORBA [2] сервис для доступа к реляционным базам данных. Он предоставляет программисту CORBA API, позволяющее эффективно интегрировать обращения к базам данных в общую инфраструктуру распределенного CORBA приложения и определяет ряд объектов более высокого уровня (т. н. Collection), который уменьшают необходимость в низкоуровневом программировании SQL запросов.

Используя UAKGQuery можно организовать эффективный и однородный доступ к данным из разных источников, вне зависимости от их расположения и архитектурных ограничений БД.

Этот документ представляет собой более или менее неформальное введение в UAKGQuery. Материал расположен не по степени сложности, а "так, как удобно рассказывать". Строгое и полное описание UAKGQuery находится в Справочнике [1] .

1.1 Историческая справка:

сервис UAKGQuery вырос из реализации CosQuery [3], и начал развиваться с 1998 г. После анализа масштабируемости и производительности структура CosQuery API была полностью пересмотрена. Некоторые из примененных методов построения эффективных распределенных приложений можно найти в [7] [6]

1.2 Предварительные сведения

Предполагается, что читатель знаком с архитектурой CORBA [2] и имеет опыт работы с CORBA приложениями на C++. (для изучения мы рекомендуем [5]). Также предполагается что читатель знаком с основами реляционных баз данных. Последняя глава также требует знакомства с такими понятиями как транзакции, CORBA Transaction Service [4] и XA транзакции [8]. Впрочем, при первом чтении ее можно опустить.

2 Общее описание механизма работы

Процесс работы пользовательского приложения с UAKGQuery выглядит следующим образом:

- Приложение инициализирует ORB, При необходимости, сервис транзакций и UAKGQueryService.
- Получает ссылку на CORBA объект типа DBConnectionManager (менеджер соединений с базами данных), посредством механизма разрешения инициальных объектов в CORBA.

- Создает Объект QueryManager, с помощью вызова DBConnectionManager::createQueryManager , который предоставляет API для создания и выполнения SQL запросов и коллекций.
- Пользуясь QueryManager можно либо
 - сразу получить результат SQL запроса,
 - создать запрос для интерактивного чтения данных,
 - создать коллекцию и обращаться к данным используя API коллекции.
- В конце работы необходимо уничтожить объект QueryManager, вызвав метод destroy.

3 Процесс подключения к БД

Для того, чтобы установить соединение с базой данных необходимо:

1. Инициализировать CORBA окружение, пользуясь функцией CORBA::ORB_init
2. Инициализировать сервис, пользуясь функцией initUAKGQueryService. Эта функция определена в заголовочном файле UAKGQueryInit.h.
3. Получить инициальный объект по имени "UAKGQueryService", пользуясь стандартным механизмом инициальных ссылок в CORBA.
4. Преобразовать его к типу DBConnectionManager.
5. Получить QueryManager.

Это можно проиллюстрировать следующим фрагментом кода на C++ :

```
#include <iostream>
#include CORBA_H // для CORBA окружения
#include <UAKGQuery.h> // для UAKGQuery
#include <UAKGQueryInit.h> // для метода инициализации.

.....

// инициализировать ORB
orb = CORBA::ORB_init(argc,argv);

// инициализировать UAKGQueryService
initUAKGQueryService(orb.in());

// получить инициальный объект
CORBA::Object_var obj;
try {
    obj = orb->resolve_initial_references("UAKGQueryService");
```

```

}catch(const CORBA::ORB::InvalidName&){
    std::cerr << argv[0] << ": can't resolve UAKGQueryService" << std::endl;
    return 1;
}

if(CORBA::is_nil(obj)) {
    std::cerr << argv[0] << ": UAKGQueryService is a nil object reference"
    std::cerr << std::endl;
    return 1;
}

// Преобразовать его тип в DBConnectionManager.
UAKGQuery::DBConnectionManager_var dbManager =
    UAKGQuery::DBConnectionManager::_narrow(obj);
if (CORBA::is_nil(dbManager)) {
    std::cerr << argv[0] << ": can't narrow dbManager to correct type" << std::endl;
    return 1;
}

// Получить QueryManager, с помощью вызова DBConnectionManager
UAKGQuery::QueryManager_var queryManager;
try {
    queryManager =
        dbManager->createQueryManager("skott","tiger","Oracle","Oracle8","");
}catch(UAKGQuery::QueryManagerNotFound){
    std::cerr << argv[0] <<": can't find query manager." << std::endl;
    return 1;
}

// now you can do something with QueryManager
.....
.....

//time to disconnect.
queryManager->destroy();

```

4 Пример выполнения простых запросов

Выполнение простых запросов можно произвести, пользуясь семейством методов QueryEvaluator::evaluate:

```

    evaluate_rc, evaluate_records, evaluate_record, evaluate_records_inout,
    evaluate_rc_inout, evaluate_records_e, evaluate_rc_e,

```

эти методы отличаются только типом входных и выходных параметров. Мы сейчас приведем простой пример обращения к базе данных, а потом приступим к описанию параметров метода и типов данных:

```
try {
    UAKGQuery::OctSeq_var octSeq =
        queryManager->evaluate_rc("select * from tab",
            "SQL92",
            RecordDescriptionAccess::empty(),
            RecordAccess::emptyOctSeq()
        );
} catch(const CosQuery::QueryInvalid& ex){
    cerr << "QueryTypeInvalid:" << ex.why << endl;
} catch(const CosQuery::QueryTypeInvalid& ex){
    cerr << "QueryTypeInvalid" << endl;
} catch(const CosQuery::QueryProcessingError& ex){
    cerr << "QueryProcessingError:" << ex.why << endl;
}
```

Что это означает:

- OctSeq – это тип результата, обозначающий пакованную последовательность байт. Доступ к ней осуществляется с помощью классов RCReader и RCWriter, о которых мы расскажем после.
- "select * from tab" – собственно текст SQL запроса.
- "SQL92" - флаг запроса, где можно указывать язык запроса и другие параметры, такие как "количество считываемых за один раз записей".
- RecordDescriptionAccess - константа типа RecordDescription: в ней содержится описание входных параметров запроса. Так как в нашем запросе мы не передаем никаких параметров, мы передаем туда константу, содержащую пустое значение. Она определена в классе RecordDescriptionAccess с тем, чтобы не определять:

```
UAKGQuery::RecordDescription emptyDescription;
emptyDescription.length(0);
```

перед каждым вызовом запроса без параметра.

- RecordAccess::emptyOctSeq - собственно параметры запроса. Они у нас должны быть в той-же форме, что и возвращаемое значение. Как вы наверное уже догадались, RecordAccess::emptyOctSeq() возвращает пустую константу.

Теперь для полноты картины приведем функцию печати полученной информации, а потом перейдем к рассказу о типах данных.

```
void printRC(std::ostream& out, const UAKGQuery::OctSeq& octSeq)
{
    RCHeader header;
    CORBA::ULong pos=0;
    RCReader::readHeader(header,pos,octSeq);
    out << "Header: nRecords=" << header.nRecords << endl;
    out << "Header: nFields=" << header.nFields << endl;
    for(CORBA::Long nRecord=0; nRecord<header.nRecords; ++nRecord){
        for(CORBA::ULong nField=0; nField<header.nFields; ++nField){
            FieldValue_var fv = RCReader::readField(pos,octSeq);
            CORBA::String_var str = FieldValueAccess::getAsString(fv);
            out << str ;
            out << "|";
        }
        out << '\n';
    }
}
```

5 Типы данных

5.1 FieldValue

Этот тип данных представляет значение поля в базе данных. Он определен в модуле CosQueryCollection как:

```
///  
typedef boolean Null;  
  
/**  
 * this union represent one field in DB  
 **/  
union FieldValue switch(Null){  
    case FALSE : Value v;  
};
```

Value может иметь значение одного из следующих типов:

```
/**  
 * what can be not null value in DB:  
 **/  
union Value switch(FieldType) {  
    ///  
    case TypeBoolean: boolean b;
```

```
///
case TypeChar: char c;
///
case TypeOctet: octet o;
///
case TypeShort : short s;
///
case TypeUShort : unsigned short us;
///
case TypeLong : long l;
///
case TypeULong : unsigned long ul;
///
case TypeFloat : float f;
///
case TypeDouble : double d;
///
case TypeString : string str;
///
case TypeObject : Object obj;
///
case TypeSmallInt : short si;
///
case TypeInteger : long i;
///
case TypeReal : float r;
///
case TypeDoublePrecision : double dp;
///
case TypeCharacter : string ch;
///
case TypeDecimal : Decimal dec;
///
case TypeNumeric : Decimal n;
///
case TypeDateTime : DateTime dt;
///
case TypeRaw : sequence<octet> raw;
///
case TypeWString : wstring wstr;
///
case TypeBlob : Blob bl;
///
case TypeClob : Clob cl;
///
case TypeWclob : Wclob wcl;
```

};

Соответствие SQL типов данных и CORBA типов приведено в следующей таблице:

	<i>CORBA</i>	<i>SQL ORACLE</i>	<i>SQL InterBase</i>
<i>TypeBoolean</i>	<i>boolean</i>	<i>CHAR(1)[in]</i>	<i>CHAR(1)[in]</i>
<i>TypeChar</i>	<i>char</i>	<i>CHAR(1)</i>	<i>CHAR(1)</i>
<i>TypeOctet</i>	<i>octet</i>	<i>CHAR(1)[in]</i>	<i>CHAR(1)[in]</i>
<i>TypeShort</i>	<i>short</i>	<i>NUMBER(5,0)</i>	<i>SMALLINT</i>
<i>TypeUShort</i>	<i>unsignedshort</i>	<i>NUMBER(5,0)</i>	<i>SMALLINT</i>
<i>TypeLong</i>	<i>long</i>	<i>NUMBER(10,0)</i>	<i>INTEGER</i>
<i>TypeULong</i>	<i>unsignedlong</i>	<i>NUMBER(10,0)</i>	<i>INTEGER</i>
<i>TypeFloat</i>	<i>float</i>	<i>NUMBER(x,y) x < 5 ^ y > 0</i>	<i>FLOAT</i>
<i>TypeDouble</i>	<i>double</i>	<i>NUMBER(x,y) x > 5 ^ y > 0</i>	<i>DOUBLEPRECISION</i>
<i>TypeString</i>	<i>string</i>	<i>VARCHAR2(X)</i>	<i>VARCHAR(X)</i>
<i>TypeObject</i>	<i>Object</i>	<i>VARCHAR2(X) ∨ UDF</i>	<i>VARCHAR(X)</i>
<i>TypeSmallInt</i>	<i>short</i>	<i>NUMBER(5,0)</i>	<i>SMALLINT</i>
<i>TypeInteger</i>	<i>long</i>	<i>NUMBER(10,0)</i>	<i>INTEGER</i>
<i>TypeReal,</i> <i>TypeDoublePrecision</i>			<i>FLOAT</i> <i>DOUBLEPRECISION</i>
<i>TypeCharacter</i>			<i>CHAR(1)[in]</i>
<i>TypeDecimal</i>	<i>CosQueryCollection :: Decimal</i>	<i>NUMBER(x,y) ^ x > 10</i>	<i>NUMERIC</i>
<i>TypeNumeric</i>	<i>CosQueryCollection :: Decimal</i>		<i>NUMERIC</i>
<i>TypeDateTime</i>	<i>CosQueryCollection :: DateTime</i>	<i>DATE</i>	<i>DATE</i>
<i>TypeRaw</i>	<i>sequence < octet ></i>	<i>LOB</i>	
<i>TypeLongRaw</i>	<i>sequence < octet ></i>	<i>BLOB</i>	
<i>TypeLongString</i>	<i>sequence < char ></i>	<i>CLOB</i>	
<i>TypeLongRaw</i>	<i>sequence < octet ></i>	<i>BLOB</i>	<i>BLOB</i>
<i>TypeLongString</i>	<i>sequence < octet ></i>		
<i>TypeWString</i>	<i>wstring</i>	<i>NCHAR, NVARCHAR</i>	
<i>TypeLongWString</i>	<i>sequence < octet ></i>		
<i>TypeBlob</i>	<i>CosQueryCollection :: Blob</i>	<i>BLOB</i>	<i>BLOB</i>
<i>TypeClob</i>	<i>CosQueryCollection :: Clob</i>	<i>CLOB</i>	<i>BLOB</i>
<i>TypeWclob</i>	<i>CosQueryCollection :: Wclob</i>	<i>NCLOB</i>	<i>BLOB</i>

5.1.1 FieldValueAccess

Со значениями FieldValue проще всего работать с помощью класса FieldValueAccess, в котором определены методы доступа к значениям в форме, более удобной по сравнению со стандартным отображением IDL в C++,

- `FieldValueAccess::isNil(const FieldValue& fv)` возвращает 1, если fv NIL, 0 в противном случае.

```
if (FieldValueAccess::isNil(myRecord[0])) {
    std::cout << "NIL here" << std::endl;
}
```

- `FieldValueAccess::setNil(FieldValue& fv)` устанавливает fv в NIL.
- Для каждого типа данных XXX есть функции `FieldValueAccess::setXXX` и `FieldValueAccess::getXXX` для установки и чтения FieldValue. так, например:

```
CORBA::Long x = FieldValueAccess::getLong(fv); // прочитать CORBA::Long из fv
FieldValueAccess::setLong(fv, 1); // установить fv в 1.
```

Если при чтении значения, `fv` имеет значение `NIL`, то вызывается прерывание `FieldValueIsNull`. Если `fv` имеет другой тип, то бросается прерывание `InvalidFieldTypeException`

- `FieldValueAccess::setAsString` и `getAsString` позволяет работать с `FieldValue` как со строками.
- Наконец, сам класс `FieldValueAccess` определен в `$(prefix)/include/CosQueryFacade`, поэтому для того, чтобы им пользоваться необходимо включить файл `CosQueryFacade/FieldValueAccess.h`

```
#include <CosQueryFacade/FieldValueAccess.h>
```

5.2 Record, RecordSeq

`Record` соответствует одной записи БД, и представляет собой просто последовательность `FieldValue`. `RecordSeq` - последовательность записей. Т. е. IDL определения следующие:

```
typedef sequence<FieldValue> Record;
typedef sequence<Record> RecordSeq;
```

К примеру, запись из двух полей: целого '2' и строки 'qqq', может быть сформирована следующей последовательностью команд:

```
Record_var record = new Record;
record->length(2);
FieldValueAccess::setShort(record[0], 2);
FieldValueAccess::setString(record[1], "qqq");
```

5.2.1 RecordAccess

Соответствующий Facade класс `RecordAccess` определяет:

- константу - пустую запись `RecordAccess::empty()`
- константу - пустую последовательность записей `RecordAccess::emptySeq()`
- константу - пустую двоичную последовательность `RecordAccess::emptyOctSeq`
- статический метод, преобразующий `Record` в `RecordSeq` с одним элементом: `RecordAccess::createRecordSeq`.
- Семейство статических методов `addXxxField`, добавляющее поле типа `Xxx` в конец записи.

К примеру, такую же последовательность записей, как и в предыдущем примере можно сформировать используя класс `RecordAccess` следующим образом:

```
Record_var record = new Record(RecordAccess::empty());
RecordAccess.addShortField(2);
RecordAccess.addStringField("qqq");
```

5.2.2 SRecord

Это еще один "синтаксический сахар" для представления записей.

```
SRecord sr;
sr._short(2)._string("qqq");
fun(sr.in());
```

5.3 OctSeq, RCHheader

Во всех интерфейсах `UAKGQuery` программисту предоставляется выбор: передавать данные как последовательность записей или как двоичную последовательность в RC-кодировании .

Смысл использования простой двоичной последовательности (`OctSeq`) заключается в том, что `UAKGQuery` определяет свое собственное кодирование последовательностей записи в двоичный поток, более эффективное чем стандартный GIOP. Это позволяет достичь производительности передачи данных, сравнимой с максимально теоретически-возможной.

`OctSeq` определен на уровне IDL просто как поток байт:

```
typedef sequence<octet> OctSeq;
```

Полное BNF определение RC кодирование смотрите в приложении 3 14. Вы в нем нуждаетесь только в том случае, когда вам необходимо читать/писать RC последовательность без помощи `UAKGQueryService`.

Для чтения/записи RC-последовательности `UAKGQuery` представляет классы `RCWriter` и соответственно `RCReader`.

Почему используется `OctSeq` – потому что `UAKGQuery` определяет собственное кодирование записей баз данных в двоичную систему, которая более эффективна чем GIOP. С `UAKGQuery` возможно достичь теоретически максимальной производительности передачи данных.

`OctSeq` определяется в IDL как последовательность байтов:

```
typedef sequence<octet> OctSeq;
```

5.3.1 RCReader

`RCReader` - класс, с набором статических методов для чтения-записи RC-потоков.

Чтобы прочитать поток мы должны сначала прочитать заголовок, потом определить количество записей в потоке и прочитать записи.

Структура заголовка имеет следующий вид:

```
struct RCHeader
{
    octet version;          // версия протокола
    long nRecords;         // количество записей в потоке
    unsigned long nFields; // количество полей в одной записи.
};
```

Вернемся к функции `printRC`, которая читает RC поток и рассмотрим ее подробнее:

```
RCHeader header;
CORBA::ULong pos=0;
RCReader::readHeader(header, pos, octSeq);
out << "Header: nRecords=" << header.nRecords << endl;
out << "Header: nFields=" << header.nFields << endl;
```

Как мы видим, `RCReader::readHeader(header, pos, octSeq)` заполняет заголовок `header` и устанавливает `pos` на индекс первой записи в потоке.

```
for(CORBA::Long nRecord=0; nRecord<header.nRecords; ++nRecord){
    for(CORBA::ULong nField=0; nField<header.nFields; ++nField){
        FieldValue_var fv = RCReader::readField(pos, octSeq);
        printField(out, fv);
        out << "|";
    }
    out << endl;
}
```

`RCReader::readField` читает поле и устанавливает `pos` на индекс следующего поля.

Возможно одним вызовом прочитать целую запись (т. е. предыдущий фрагмент можно переписать следующим образом:

```
for(CORBA::Long nRecord=0; nRecord<header.nRecords; ++nRecord){
    Record_var record = RCReader::readRecord(pos, octSeq);
    printRecord(out, fv);
}
```

Или сразу последовательность записей:

```
RecordSeq_var recordSeq = RCReader::readRecordSeq(pos, octSeq);
```

Подробно `RCReader` интерфейс рассмотрен в справке по API.

5.3.2 RCWriter

RCWriter предоставляет интерфейс для записи потока. Следующий фрагмент создает поток и записывает в него одну запись:

```
UAKGQuery::OctSeq_var octSeq = new OctSeq;
CORBA::ULong pos;
RCWriter::writeHeader(1,record.length(),pos,octSeq);
RCWriter::writeRecord(record,pos,octSeq);
```

Естественно, можно записывать как отдельные поля, так и последовательности записей.

Заметим, что записать количество полей можно и после того, как мы записали данные. Т. е. следующий код:

```
UAKGQuery::OctSeq_var octSeq = new OctSeq;
CORBA::ULong pos;
RCWriter::writeHeader(1,record.length(),pos,octSeq);
RCWriter::writeRecord(record,pos,octSeq);
RCWriter::writeRecordSeq(recordSeq,pos,octSeq);
RCWriter::writeNRecords(recordSeq.length()+1);
```

будет работать.

5.4 RecordDescription

RecordDescription - это описание записи в БД. Определен с помощью следующего IDL интерфейса:

```
/**
 * struct for description of field size.
 * name: name of field in DB.
 * ValueType: field type.
 * size: size of field in bytes. (for strings: include \0, i. e.
 *       for VARCHAR(x) size is x+1
 * precision (have sence only for NUMERIC types) - precision.
 * scale (have sence only for NUMERIC types) - scale, as signed byte.
 */
struct FieldDescription{
    string          name; // Имя поля.
    CosQueryCollection::FieldType    type; // тип поля
    unsigned long  size; // размер поля в байтах
    unsigned short precision; // точность (имеет смысл только для десятичных)
    short          scale; // к-ство знаков после запятой.
};
typedef sequence<FieldDescription> RecordDescription;
```

Т. е. зная `RecordDescription` мы можем определить имена и типы полей в записи.

Wrapper класс, предоставляющий удобный доступ к `RecordDescription` называется `RecordDescriptionAccess`.

6 Выполнение запросов

Итак, как мы уже говорили – выполнение запроса производится с помощью семейства методов `evaluate_xxx` оуъекта `QueryManager`.

6.1 Запросы без параметров

Рассмотрим еще раз пример выполнения простого запроса:

```
try {
    UAKGQuery::OctSeq_var octSeq = queryManager->evaluate_rc("select * from tab","SQL92",
        RecordDescriptionAccess::empty(),
        RecordAccess::emptyOctSeq());
} catch(const CosQuery::QueryInvalid& ex){
    cerr << "CosQuery::QueryTypeInvalid:" << ex.why << endl;
} catch(const CosQuery::QueryTypeInvalid& ex){
    cerr << "QueryTypeInvalid" << endl;
} catch(const CosQuery::QueryProcessingError& ex){
    cerr << "QueryProcessingError:" << ex.why << endl;
}
```

Теперь мы уже знаем, что:

- Возвращаемое значение представляет собой последовательность в RC-количестве.
- Первый параметр - собственно SQL запрос.
- Второй параметр - флаги запроса.
- Третий параметр - тип параметров запроса. В данном случае так как у запроса нет параметров, то мы ему передаем пустое описание записи.
- Четвертый параметр - собственно параметры запроса.

Заметим, что случай выполнения запроса без параметра случается довольно часто, поэтому мы определили для него специальные методы: `evaluate_rc_e` и `evaluate_records_e` в которых нет параметров запроса.

Т. е. этот-же запрос можно записать более компактно:

```

try {

    UAKGQuery::OctSeq_var octSeq = queryManager->evaluate_rc_e("select * from tab","SQL92");

}catch(const CosQuery::QueryInvalid& ex){
    cerr << "QueryTypeInvalid:" << ex.why << endl;
}catch(const CosQuery::QueryTypeInvalid& ex){
    cerr << "QueryTypeInvalid" << endl;
}catch(const CosQuery::QueryProcessingError& ex){
    cerr << "QueryProcessingError:" << ex.why << endl;
}

}

\end{verbatim}

```

Семейство методов `\verb|evaluate_records|` принимает и возвращает параметры как последовательности записей.

```

\begin{verbatim}
RecordSeq_var records;
try {
    records = queryManager->evaluate_records_e("select * from tab","SQL92");
}catch(const CosQuery::QueryInvalid& ex){
    cerr << "QueryTypeInvalid:" << ex.why << endl;
}catch(const CosQuery::QueryTypeInvalid& ex){
    cerr << "QueryTypeInvalid" << endl;
}catch(const CosQuery::QueryProcessingError& ex){
    cerr << "QueryProcessingError:" << ex.why << endl;
}
}

```

\subsection{Выполнение запросов с параметрами }

Смысл появления `RecordDescription` в параметрах `evaluate` -- описание записи, которая подается на вход запроса. Четвертый параметр -- собственно параметры.

Проиллюстрируем это на примере выполнения запроса:

```

\begin{verbatim}
select name from empls where id=:ID

```

Где `:ID` – один из параметров запроса типа `CORBA::Long`.

```

char* getEmployeeNameById(long id)
{
    RecordDescription paramsDescription;
    RecordDescriptionAccess::appendLong(paramsDescription, "ID");
    Record_var params = new Record;
    params->length(1);
}

```

```

FieldValueAccess::setLong(params[0],id);
RecordSeq_var retval = queryManager->evaluate_record(
    "select name from empls where id=:ID","",
    paramsDescription, params);
if (retval->length()==0) {
    throw IncorrectEmployeeId(id);
}
return FieldValueAccess::createString(retval[0][0]);
}

```

Заметим, что `host` переменные могут использоваться как для передачи информации в DB, так и для извлечения, например с помощью SQL конструкции `select into`.

Для этих целей предусмотрено семейство методов `evaluate_*_inout`.

6.2 Batch выполнение запросов с параметрами

В том случае, когда база данных поддерживает т.н. пакетный режим исполнения, вы можете передать как параметр последовательность записей. В этом случае эффект выполнения запроса будет такой же, как и последовательное выполнение нескольких запросов с одной записью, как параметром.

Для Oracle версии 8.0 это могут быть только обращения к PL/SQL процедурам, версии 8.1 и 9 - некоторые виды `insert`, `update` и `delete` запросов.

7 Интерфейс Query

Использование `evaluate` не покрывает всего спектра работы с БД. Основные недостатки:

1. Мы вынуждены получать все данные запроса за 1 раз.
2. Мы должны знать заранее структуру получаемых данных.
3. Мы каждый раз интерпретируем SQL.

Более полное использование возможностей реляционной БД достигается с использованием интерфейса `Query`.

```

interface Query
{
    /**
     *@return owner of query
     */
    readonly attribute QueryManager query_mgr;
}

```

```

/**
 *@return text of query.
 */
readonly attribute string queryText;

/**
 * return status of query: i.e:
 * complete when query is executed, otherwise incomplete
 */
CosQuery::QueryStatus get_status ();

/**
 * prepare query for executing.
 * if query have no parameters, paramsDescription must be empty
 * sequence.
 */
void prepare_query(in RecordDescription paramsDescription)
    raises(CosQuery::QueryProcessingError);

/**
 * synonym for prepare_query
 */
void prepare(in RecordDescription paramsDescription)
    raises(CosQuery::QueryProcessingError);

/**
 * execute query
 *@params octSeq_ records of execute parameters, coded as RCSeq
 * (note, that prepare parameters is record descriptio of execute
 * record).
 */
void execute_rc(in OctSeq octSeq_)
    raises(CosQuery::QueryProcessingError);

/**
 * execute query with inout parameters
 *@params octSeq_ records of execute parameters, coded as RCSeq
 */
void execute_rc_inout(inout OctSeq octSeq_)
    raises(CosQuery::QueryProcessingError);

/**

```

```

* execute query
*@params records -- query host parameters in RecordSeq
* (query will be evaluated records.length() times)
**/
void execute_records(in RC::RecordSeq records)
    raises(CosQuery::QueryProcessingError);

/**
* execute query
*@params record_ -- query host parameters in one record
**/
void execute_record(in CosQueryCollection::Record record_)
    raises(CosQuery::QueryProcessingError);

///
void execute_records_inout(inout RC::RecordSeq recordSeq_)
    raises(CosQuery::QueryProcessingError);

///
RecordDescription get_result_description()
    raises(CosQuery::QueryProcessingError,
           QueryNotPrepared);

/**
* get description of records parameters
*@precondition
* must be called after prepare
**/
RecordDescription get_parameters_description()
    raises(CosQuery::QueryProcessingError);

///
RC::RecordSeq get_all_parameters_records()
    raises(CosQuery::QueryProcessingError);

///
RC::RecordSeq get_parameters_records(in StringSeq neededFields)
    raises(CosQuery::QueryProcessingError,
           InvalidParameterName);

///
OctSeq get_all_parameters_rc()
    raises(CosQuery::QueryProcessingError);

///
OctSeq get_parameters_rc(in StringSeq fieldNames)

```

```

                raises(CosQuery::QueryProcessingError,
                       InvalidParameterName);

/**
 * @returns number of fetched rows.
 */
unsigned long get_row_count()
                raises(CosQuery::QueryProcessingError);

/**
 * fetch query result in records.
 * @param numberOfRecords -- number of records to fetch.
 *      0 means, that we want to fetch all records.
 * @param more -- true, if status is incomplete (i.e. we can query
 * more results), otherwise false.
 * @returns fetched rows packed in RC coding to octet sequence.
 */
OctSeq fetch_rc(in unsigned long numberOfRecords, out boolean more)
                raises(CosQuery::QueryProcessingError);

/**
 * synonym for fetch_rc.
 */
OctSeq get_result_rc(in unsigned long numberOfRecords)
                raises(CosQuery::QueryProcessingError);

/**
 * fetch query result in records.
 * @param numberOfRecords -- number of records to fetch.
 *      0 means, that we want to fetch all records.
 * @param more -- true, if status is incomplete (i.e. we can query
 * more results), otherwise false.
 * @returns fetched records.
 */
RC::RecordSeq fetch_records(in unsigned long numberOfRecords,
                            out boolean more)
                raises(CosQuery::QueryProcessingError);

/**
 * synonym for fetch_records
 */
RC::RecordSeq get_result_records(in unsigned long numberOfRecords)
                raises(CosQuery::QueryProcessingError);

/**

```

```

    * skip N records without retrieving.
    *@returns actual number of skipped records.
    */
unsigned long skip(in unsigned long numberOfRecords,
                  out boolean more)
                  raises(CosQuery::QueryProcessingError);

/**
 * request Blob for filling query parameters
 *@returns empty Blob for writing only.
 */
CosQueryCollection::Blob create_blob()
    raises(CosQuery::QueryProcessingError);

/**
 * request Clob for filling query parameters
 *@returns empty Clob for writing only.
 */
CosQueryCollection::Clob create_clob()
    raises(CosQuery::QueryProcessingError);

/**
 * request Wclob for filling query parameters
 *@returns empty Wclob for writing only.
 */
CosQueryCollection::Wclob create_wclob()
    raises(CosQuery::QueryProcessingError);

/**
 *@return last error.
 * if Query is ok, code in error is 0.
 */
QueryError get_last_error();

/**
 * destroy query, which not longer needed
 **/
void destroy();
};

```

При работе с Query программист должен придерживаться следующей последовательности действий:

1. создать запрос, используя метод `QueryManager::create_query`.
2. подготовить запрос к выполнению, вызвав метод `Query::prepare`. Параметром этого метода является описание записи входных параметров запроса.
3. выполнить запрос при помощи какого-либо из методов семейства `Query::execute_xx`.
4. извлечь данные при помощи методов семейства `Query::fetch_xx`
5. удалить ненужный запрос.

К примеру, типичная задача "получение имени работника по id" будет выглядеть следующим образом:

```
Query_var query = queryManager->create_query(
                                "select name from empls where id=:ID","");
query->prepare(paramsDescription);
query->execute_record(params);
RecordSeq_var retval = query->fetch_rc(0);
query->destroy();
```

7.1 Prepared Query

Запрос можно подготовить, вызвав функцию `prepare` 1 раз, а потом много раз исполнять, меняя только значения параметров. Это более эффективно для часто повторяющихся запросов, так как анализ SQL предложения и передача описаний параметров производится только 1 раз.

Типичное использование этой техники можно проиллюстрировать следующими фрагментами кода:

```
class EmployeeManager
{
    Query id2nameQuery_;

    ....

public:

    char* getNameById();

};

void EmployeeManager::init()
{
    ....
    id2nameQuery_ = qm_->create_query("select name from empls where id=:id","");
    id2nameQuery_->prepare(paramsDescription);
    .....
}
```

```

}

char* EmployeeManager::getNameById(CORBA::Long id)
{
    Record params(1);
    FieldValueAccess::setLong(params[0],id);
    id2nameQuery_ ->execute_record(params);
    .....
}

EmployeeManager::~EmployeeManager()
{
    .....
    id2nameQuery_ ->destroy();
    ....
}

```

Т. е. мы помещаем создание и инициализацию запроса в секцию инициализации класса, удаление запроса – в деструктор, а в функции которая делает реальную работу помещаем собственно это работу.

7.2 Интерактивное извлечение данных

Еще одно полезное свойство интерфейса Query – API для получения данных по частям (в соответствии с шаблоном проектирования Iterator.

Стандартный цикл вывода записей выглядит следующим образом:

```

CORBA::Boolean more=true;
while(more)
{
    UAKGQuery::OctSeq_var rc = query->fetch_rc(chunkSize,more);
    .....
    do something;
}

```

т. е. fetch возвращает chunkSize или меньше записей и устанавливает more в false, если мы получили все записи запроса.

7.3 Получение описания

Мы можем получить описание результатов запроса, пользуясь методом Query::get_result_description();

Следующий фрагмент кода возвращает нам описание таблицы, имя которой ввел пользователь.

```

char tablename[MAX_TNAME_LEN];
std::cout << "enter table name:";
std::cout.flush();
std::cin.getline(tablename,MAX_TNAME_LEN);
std::ostream ostr;
ostr << "select * from " << tablename;
Query_var query = queryManager->create_query(ostr.str(),"");
ostr.rdbuf()->freeze(0);
RecordDescription_var tableDescription = query->get_result_description();

```

8 Большие объекты (Large Objects)

Особенности работы с большими объектами базы данных (LOB) заключены в:

- трех интерфейсах - Blob, Clob, Wclob
- трех методах интерфейса Query (create_blob, create_clob, create_wclob)

Blob, Clob, Wclob можно указывать как параметры запроса к базе. Объекты этих типов можно получать в строках результата. Для выполнения запроса к базе, параметром которого является LOB, невозможно использовать механизм evaluate. Только Query содержит методы создания LOB-ов для записи. Все LOB, полученные в результате выполнения запросов, предназначены только для чтения. Интерфейсы Blob, Clob и Wclob имеют почти идентичный интерфейс. Они отличаются только в соответствующих типах данных.

- Blob предназначен для работы с массивами бинарных данных, единицей которых является октет, используется UAKGQuery::OctSeq;
- Clob - символьные данные, используется CORBA::String;
- Wclob - символьные данные в unicode, используется CORBA::WString;

Рассмотрим интерфейс LOB на примере Blob:

```

interface Blob {
    ///
    unsigned long length() raises(CosQuery::QueryProcessingError);
    ///
    UAKGQuery::OctSeq fetch_chunk(in unsigned long chunkSize
        , out boolean more)
        raises(CosQuery::QueryProcessingError, ForWritingOnly);
    ///
    void put_chunk(in UAKGQuery::OctSeq data)
        raises(CosQuery::QueryProcessingError, ForReadingOnly);
};

```

Метод `length` возвращает длину LOB, в соответствующих единицах (для Blob - октеты, для Clob - символы, для Wclob - символы unicode). Все размеры указываются в этих единицах. Метод `fetch_chunk` может быть вызван для LOB, который получен в результате выполнения запроса `select` - типа. Он предоставляет механизм для получения фактических данных LOB-а. Аргумент `chunkSize` определяет максимальный требуемый размер считываемой части LOB. При этом аргумент `more` сигнализирует о том, есть ли еще данные для чтения. Нулевое значение `chunkSize` означает требование считать весь LOB целиком. Метод `put_chunk` можно вызывать только для LOB, который создан одним из методов `Query create_<B|C|Wc>lob`. Аргумент `data` - фактические данные для записи. При вызове `fetch_chunk` и `put_chunk` в неверном контексте будет брошено соответствующее исключение (`ForWritingOnly`, `ForReadingOnly`), сигнализирующее о том, что данный LOB предназначен только для записи, либо только для чтения. Методы `Query`

```

/**
 * request Blob for filling query parameters
 * @returns empty Blob for writing only.
 */
CosQueryCollection::Blob create_blob()
    raises(CosQuery::QueryProcessingError);

/**
 * request Clob for filling query parameters
 * @returns empty Clob for writing only.
 */
CosQueryCollection::Clob create_clob()
    raises(CosQuery::QueryProcessingError);

/**
 * request Wclob for filling query parameters
 * @returns empty Wclob for writing only.
 */
CosQueryCollection::Wclob create_wclob()
    raises(CosQuery::QueryProcessingError);

```

возвращают LOB-ы соответствующих типов для записи и дальнейшего связывания с параметрами этой `Query`. Приведем пример чтения:

```

CORBA::String_var sql = CORBA::string_dup("select b from blob_test");
RecordSeq_var rs = new RecordSeq();
CORBA::Boolean more = true;
try {
    Query_var q = queryManager->create_query(sql, "");
    q->prepare(RecordDescriptionAccess::empty());
    q->execute_records(RecordAccess::emptyRecordSeq());
}

```

```

rs = q->fetch_records(1, more);
if (rs->length() > 0) {
    Blob_var bl;
    try {
        bl = FieldValueAccess::getBlob(rs[0][0]);
    } catch (const InvalidFieldTypeException& ) {
        cerr << "InvalidFieldTypeException" << endl;
        return;
    } catch (const FieldValueIsNull& ) {
        cerr << "FieldValueIsNull" << endl;
        return;
    }
    FILE* f = fopen("file_from_blob", "wb");
    OctSeq_var octSeq;
    long n = 0;
    for (more=true; more; n++){
        octSeq = bl->fetch_chunk(chunk_size, more);
        fwrite(octSeq->get_buffer(), 1, octSeq->length(), f);
    }
    fclose(f);
} else {
    cerr << "In database no records" << endl;
}
q->destroy();
} catch(const QueryTypeInvalid& ex) {
    cerr << "QueryTypeInvalid" << endl;
    return;
} catch(const QueryInvalid& ex) {
    cerr << "QueryInvalid" << endl;
    return;
} catch(const QueryProcessingError& ex) {
    cerr << "QueryProcessingError" << endl;
    cerr << ex.why << endl;
    return;
}
}

```

Приведем пример записи:

```

UAKGQuery::OctSeq_var octSeq = new UAKGQuery::OctSeq();
CORBA::ULong chunk_size;

...// fill octSeq and choose chunk_size

RecordDescription_var recordDescription = new RecordDescription;
recordDescription->length(1);
recordDescription[0].name=CORBA::string_dup(":param");
recordDescription[0].type=TypeBlob;

```

```

RecordSeq_var recordSeq = new RecordSeq;
recordSeq->length(1);
recordSeq[0].length(1);
try {
    Query_var q = queryManager->create_query(
        "insert into blob_test(b) values(:param)", "");
    q->prepare(recordDescription);
    Blob_var bl = q->create_blob();
    if (chunk_size == 0) {
        bl->put_chunk(octSeq.in());
    } else {
        CORBA::ULong sentLength = 0;
        CORBA::ULong leftLength = 0;
        const CORBA::Octet* curr;
        while (sentLength < octSeq.length()) {
            curr = octSeq.get_buffer() + sentLength;
            leftLength = octSeq.length() - sentLength;
            if (leftLength >= chunk_size) {
                bl->put_chunk(octSeq(chunk_size, chunk_size, curr, 0));
                sentLength += chunk_size;
            } else {
                bl->put_chunk(octSeq(leftLength, leftLength, curr, 0));
                sentLength = octSeq.length();
            }
        }
    }
}

FieldValueAccess::setBlob(recordSeq[0][0], bl);
q->execute_records(recordSeq);
q->destroy();
} catch(const QueryTypeInvalid& ex) {
    cerr << "QueryTypeInvalid" << endl;
    return;
} catch(const QueryInvalid& ex) {
    cerr << "QueryInvalid" << endl;
    return;
} catch(const QueryProcessingError& ex) {
    cerr << "QueryProcessingError" << endl;
    cerr << ex.why << endl;
    return;
}

```

9 Интерфейс QueryManager

9.1 Последовательности

NAME `get_next_sequence_value` - возвращает следующее значение последовательности.
SYNOPSIS

```
long get_next_sequence_value(in string sequence_name);
```

DESCRIPTION Функция

```
get_next_sequence_value
```

RETURN VALUE `get_next_sequence_value` возвращает следующее значение из последовательности

```
sequence_name
```

10 Свойства

NAME `get_property`

SYNOPSIS

```
string get_property(in string property_name) raises(PropertyNotDefined);
```

DESCRIPTION Функция

```
get_property
```

Доступные свойства :

```
collection
```

Если свойство

```
collection
```

RETURN VALUE `get_property` return value of property if such exist and have a value, else function raise `PropertyNotDefined` exception.

11 Коллекции

11.1 Введение

`AKGQueryCollection` (коллекции) – это семейство интерфейсов, представляющих высокоуровневую объектную модель, предназначенную для работы с множествами данных в БД.

Коллекции позволяют избавиться от рутинного программирования при работе с таблицами, предоставляя объекты более высокого уровня, которые реализуют необходимые механизмы управления данными, скрывая детали реализации от программиста.

11.2 Общее описание механизма работы

11.2.1 "Физический смысл" коллекции

В сухом остатке коллекцию можно рассматривать как специфицированный набор SQL предложений для чтения, записи и модификации данных, каждое из которых связано с одним из методов объекта.

Например, в простейшем случае, для чтения данных из одной таблицы мы, очевидно, должны воспользоваться SQL-предложением следующего типа:

```
SELECT <select-part> FROM <from-part> WHERE <where-part>
```

На практике для этого мы можем создать коллекцию, которая будет специфицирована SELECT, FROM и WHERE частью данного предложения, после чего использовать методы коллекции, например метод `retrieve_by_filter` с параметром `<filter>`, который будет эквивалентен SQL-предложению

```
SELECT <select-part> FROM <from-part> WHERE <where-part> AND <filter>
```

Если потом мы захотим модифицировать считанные данные, мы сможем использовать метод `update_by_filter` той же коллекции, который будет эквивалентен SQL-предложению

```
UPDATE <from-part> SET <set-part> WHERE <where-part> AND <filter>
```

где SET часть автоматически построена на основании SELECT части спецификации коллекции.

Таким образом, на более общем уровне коллекция выступает как комплексный механизм управления данными, ограниченный следующими вещами:

1. Набором полей данных (SELECT часть SQL предложения)
2. Источником данных (FROM часть SQL предложения)
3. Условиями выбора (WHERE часть SQL предложения)
4. Принципом упорядочивания данных (ORDER-BY часть SQL предложения)

Все эти вещи должны быть заданы на этапе создания объекта.

11.2.2 Порядок работы с коллекцией

Порядок работы с `UAKGQueryCollection` выглядит так:

1. Создать коллекцию, воспользовавшись одним из методов `QueryManager`.
2. Пользуясь методами коллекции либо (а) сразу выполнить необходимое преобразование над данными, либо (б) получить объект `Iterator` для последовательного чтения данных и воспользоваться им.
3. В конце работы уничтожить `UAKGCollection`, вызвав метод `destroy`.

11.3 Создание Query Collections

UAKGQuery предоставляет два типа коллекций: UAKGCollection и UAKGKeyCollection. UAKGCollection соответствует "простому" набору данных, UAKGKeyCollection - набору данных с ключом.

Существует два способа создания UAKGCollection:

1. создание по заданным полям
2. создание по заданному SQL предложению

К примеру, объект UAKGCollection можно создать так:

```
UAKGCollection_var collection = queryManager->create_collection_by_parts(
    "F1,F2", "UAKGTEST", "F1=1", "F2");
```

или так:

```
UAKGCollection_var collection = queryManager->create_collection_by_query(
    "select F1,F2 from UAKGTEST where F1=1 order by F2");
```

По сути, приведенные участки кода эквивалентны друг другу.

Отметим, что параметры первого вызова суть:

1. "F1,F2" – набор полей (SELECT часть SQL предложения);
2. "UAKGTEST" – источник данных (FROM часть SQL предложения);
3. "F1=1" – условие выборки (WHERE часть SQL предложения);
4. "F2" – параметр упорядочивания (необязательный) : ORDER часть.

11.4 Доступ к данным

11.4.1 Получение данных при помощи методов объекта Iterator

Удобный способ доступа к данным UAKGCollection (например, к результатам запроса) предоставляется объектом `Iterator`.

Общая процедура работы с итератором такова:

1. Получить объект `Iterator`
2. Получать записи при помощи методов объекта
3. Удалить объект `Iterator`

Пример использования объекта `Iterator`:

```

UAKGIterator_var iterator = collection->create_uakgiterator();
Boolean more = true;
while( more )
{
    OctSeq_var octSeq = iterator->fetch_rc(50, more);
    printRC( octSeq );
}
iterator->destroy();

```

Методы объекта Iterator:

- `fetch_rc(ULong n, Boolean& more)` Читает `n` записей и возвращает их как поток байт в RC кодировке.
- `fetch_records(ULong n, Boolean& more)` Читает `n` записей и возвращает их как последовательность строк.
- `skip(ULong n, Boolean& more)` Пропускает `n` записей.

11.4.2 Получение данных при помощи собственных методов коллекции

Еще один механизм получения данных коллекции - это методы

- `retrieve_by_filter(const char* where_filter)`
- `retrieve_by_pattern(const Record& pattern)`

Оба метода возвращают поток байт, содержащий записи, отфильтрованные из общего объема данных коллекции на основании фактического значения полученного ими параметра. При этом первый метод фильтрует записи на основании условия, закодированного в переданной ему текстовой строке, второй - отбирает их по принципу соответствия заданному образцу.

Пример:

```

UAKGCollection_var collection_ = queryManager->create_collection_by_query(
    "select F1,F2 from UAKGTEST where F1=1 order by F2");
OctSeq_var octSeq_ = collection_->retrieve_by_filter("F2='test'");
printRC( octSeq_ );
collection_->destroy();

```

- этот код приведет к распечатке всех записей UAKGTEST, поле F1 которых равно "1", а поле F2 - "test".

11.4.3 Отбор путем сопоставления с образцом

Использование метода `retrieve_by_pattern`, который отбирает записи по принципу их соответствия образцу, требует более подробного комментария.

Вообще говоря, сопоставление с образцом представляет собой широкую концепцию, которая является дальнейшим развитием принципа QBE (Query By Example).

Особенности нашей реализации принципа таковы:

- Структура записи образца должна соответствовать структуре записей в запрашиваемом наборе данных. (т. е. количество полей и их тип должны быть идентичны)
- Запись `r` соответствует образцу `p`, если:
 1. $\forall i \in 0 \dots \text{length}(p) : p[i]! = \text{NIL} \rightarrow p[i] \text{ match } r[i]$ (Т. е. для всех индексов полей, соответствующие поля должны сопоставляться)
 2. Два поля r_i и p_i сопоставимы тогда и только тогда, когда:
 - (a) $\text{type}(r_i) == \text{string} \leftrightarrow (r_i \text{ LIKE } p_i)$
 - (b) $\text{type}(r_i)! = \text{string} \leftrightarrow (r_i == p_i)$(Т. е. поля строковых типов сопоставляются с помощью оператора LIKE, остальных типов – с помощью равенства).

Практически это выглядит так:

Допустим, у нас есть таблица UAKGTEST с двумя полями: F1 - типа number и F2 - типа varchar2. Пусть также мы создали коллекцию, которая работает со строками этой таблицы. Тогда для того, чтобы получить те строки коллекции, у которых F1=5, необходимо:

1. Сформировать запись (образец), у которой первое поле равно 5, а второе установлено в NULL (это поле в фильтрации участвовать не будет);
2. Выбрать нужные строки при помощи `retrieve_by_pattern`, передав образец как параметр.

То же в коде:

```
Record_var pattern = new Record;  
pattern->length(2);  
FieldValueAccess::setLong( pattern[0], 5);  
FieldValueAccess::setNull( pattern[1] );  
OctSeq_var octSeq = collection->retrieve_by_pattern( pattern );  
printRC( octSeq );
```

11.4.4 Добавление, изменение и удаление данных

Кроме действий, описанных выше, при работе с UAKGCollection можно выполнить следующий набор действий:

1. Добавление записей:

- `add_record(const Record& element)` - добавляет запись к коллекции.
- `add_records(const RecordSeq& elements)` - добавить последовательность записей.
- `add_rc(const OctSeq& rc)` - добавить последовательность записей в RC кодировке.

Заметим, что при добавлении запись добавляется непосредственно в базу данных и принадлежность ее условиям выборки коллекции не проверяется. Рассмотрим, к примеру, следующий фрагмент кода:

```
UAKGCollection_var collection=qm->create_collection(
    "select * from emp where deptno=22"
);

ULong n=collection->number_of_records();
cout << "now number of records is:" << n << endl;
RecordDescription_var rd = collection->get_record_description();
Record_var record=CosQueryFacade::RecordAccess::createRecordByDescription(rd);
CosQueryFacade::RecordAccess::setShortByName(record.inout(), "EMPNO", 11, rd);
CosQueryFacade::RecordAccess::setShortByName(record.inout(), "DEPTNO", 11, rd);
collection->add_record(record);
n=collection->number_of_records();
cout << "now number of records is:" << n << endl;
```

будет выводить 2 одинаковых числа, хотя в БД добавится одна запись.

2. Обновление записей:

- `update_by_pattern(const Record& newRecord, const Record& pattern)`
- установить в `newRecord` те записи, которые соответствуют образцам `pattern`. Например:

```
SRecord sr1, sr2;
collection->update_by_pattern(
    sr1._short(1)._short(2).in(), sr2._nil()._short(2).in()
);
```

установить в (1,2) все записи, у которых второе поле было равно 2.

- `update_by_filter(const Record& newRecord, const char* filter)`
- установить в `newRecord` те записи, которые соответствуют SQL выражению. Например:

```
collection_->update_by_filter(sr._short(1)._short(2),"x2=2");
```

установит в (1,2) все записи, у которых поле x2 было ранее равно 2-м.

3. Удаление записей:

- `remove_record(const Record& record)` - удалить записи, совпадающие с данной записью.
- `remove_records_by_filter(const char* filter)` - удалить записи, соответствующие некоторому условию.
- `remove_records_by_pattern(const Record& pattern)` - удалить записи, соответствующие образцу.
- `remove_all_records()` - удалить все записи из коллекции.

Например:

```
UAKGCollection_var collection_ = queryManager->create_collection_by_query(
    "select F1,F2 from UAKGTEST where F1=1 order by F2");
Record_var inpRecord = new Record;
inpRecord->length(2);
FieldValueAccess::setLong( inpRecord[0], 5);
FieldValueAccess::setString( inpRecord[1], "test" );
collection_->add_record(inpRecord);
FieldValueAccess::setNull( inpRecord[1] );
OctSeq_var octSeq_ = collection_->retrieve_by_pattern(inpRecord);
printRC( octSeq_ );
collection_->remove_records_by_pattern( inpRecord );
collection_->destroy();
```

11.5 Запросы к коллекции

Приведенного выше набора методов не всегда достаточно, для комфортабельной работы с коллекциями записей. Иногда нам может потребоваться изменить набор запрашиваемых полей, или получить доступ к данным из связанной таблицы.

Для подобных случаев коллекции предоставляют семейство методов `evaluate`. (как `QueryManager`, т. е. и коллекции, и менеджер очередей реализуют интерфейс `QueryEvaluator`).

К примеру:

```
result = collection->evaluate_rc_e("select x1,x3 from @ where x1=x2","")
```

выберет из коллекции поля `x1`, `x3` тех записей, у которых `x1=x2`, а запрос:

```
collection=queryManager->create_collection("select * from orders","");
result = collection->evaluate_rc(
```

```
"select @,CUSTOMERS.NAME from @,CUSTOMERS where CUSTOMER.ID=customer_id",
""
);
```

добавит к набору записей коллекции заказов имя заказчика из связанной таблицы.

Следующий запрос:

```
result = collection->evaluate_rc("select @ from @ where NOT @","");
```

Проинвертитует коллекцию: т. е. выберет нам все записи из источника данных, которые не принадлежат коллекции.

Теперь, определим семантику нашего расширения SQL значком @ более формально: Если в основании коллекции лежит выражение вида:

```
select <select-part> from <from-part> where <where-part>
```

и в evaluate_xx передается выражение

```
select [@,][new-select-part]
      from @[,new_from_part] [where new_where_part] ,
```

то результирующее выражение, которое будет обработано, будет имеет вид:

```
select <select-part>,<new-select-part>
      from <from-part>,<new-from-part>
      where (<where-part>) AND (<new-where-part>)".
```

Сокращенные формы обрабатываются более или менее очевидным способом.

11.5.1 Ограничения

- Коллекция может обрабатывать только SELECT запросы без клауз HAVING_BY и GROUP_BY Полностью грамматика интерпретируемых нами SQL выражений приведена в приложении 15.

11.5.2 Список методов

- evaluate_rc(const char* queryText, const char* queryFlags, const RecordDescription& recordDescription, const OctSeq& params)
- evaluate_rc_e(const char* queryText, const char* queryFlags);
- evaluate_rc_inout(const char* queryText, const char* queryFlags, const RecordDescription& recordDescription, OctSeq& params)

- `evaluate_record(const char* queryText, const char* queryFlags,
const RecordDescription& recordDescription, const Record& params)`
- `evaluate_records_inout(const char* queryText, const char* queryFlags,
const RecordDescription& recordDescription,
RecordSeq& params)`
- `evaluate_records(const char* queryText, const char* queryFlags,
const RecordDescription& recordDescription,
const RecordSeq& params)`

11.6 Подколлекции (subcollections)

Как видно из названия, `SubCollection` - это "подколлекция". Т. е. мы можем запросить какой-то набор данных из коллекции, организованный в коллекцию, которая является подмножеством оригинала.

Способ работы с `SubCollection`:

1. Создать подколлекцию:

- `create_subcollection(const char* subquery)` Здесь `subquery` это SELECT-запрос к коллекции на языке, так-же расширенном знаком @, и с такими-же ограничениями, как и для семейства методов коллекции `evaluate`.

либо

- `create_subcollection_by_pattern(const Record& pattern)` Здесь возвращаемая коллекция это просто ограничение исходной коллекции по образцу `pattern`.

2. Работать с ней, как с обычной `UAKGCollection`

3. В конце работы – удалить ее при помощи метода `destroy()`

Например:

```
UAKGCollection_var collection_ = queryManager->
    create_collection_by_fields("tname, tabtype", "tab", "1=1", "");
UAKGIterator_var iterator = collection_->create_uakgiterator();
Boolean more;
RecordSeq_var recordSeq = iterator->fetch_records(0, more);
FieldValueAccess::setNull(recordSeq[0][0]);
iterator->destroy();
UAKGCollection_var new_collection_ = collection_->
    create_subcollection_by_pattern(recordSeq[0]);
```

```

iterator = new_collection_->create_uakgiterator();
recordSeq = iterator->fetch_records(0, more);
printRecordSeq(cout,recordSeq.in());
iterator->destroy();
new_collection_->destroy();
collection_->destroy();

```

11.7 UAKGKeyCollection

Интерфейс `UAKGKeyCollection` это специализация `UAKGCollection`, для свойства "иметь первичный ключ" ¹.

Дополнительные методы позволяют получать, обновлять и удалять записи из БД по заданному значению ключевых полей.

11.7.1 Создание UAKGKeyCollection

При создании `UAKGKeyCollection` дополнительно к описанным выше параметрам необходимо задать список ключевых полей. Для создания `KeyCollection` вы можете использовать 2 метода `UAKGQueryManager`:

- `create_key_collection_by_parts` – который создает ключевую коллекцию по соответствующим частям SQL предложений с дополнительной частью: список имен полей, из которых состоят ключи. Пример:

```

UAKGKeyCollection_var collection = queryManager->
create_key_collection_by_parts("F1,F2", "UAKGTEST", "F1=1", "F2", "F1");

```

- `create_key_collection_by_query` – Который создает ключевую коллекцию с помощью SQL запросов с нашим расширением: клаузой 'with key'.

```

UAKGKeyCollection_var collection = queryManager->
create_key_collection_by_query(
    "select F1,F2 from UAKGTEST where F1=1 order by F2 with key F1"
);

```

Пример:

```

UAKGKeyCollection_var collection = queryManager->
create_key_collection_by_parts("F1,F2", "UAKGTEST", "F1=1", "F2", "F1");

```

или

```

UAKGKeyCollection_var collection = queryManager->
create_key_collection_by_query(
    "select F1,F2 from UAKGTEST where F1=1 order by F2 with key F1");

```

¹т. е. ключ, имеющий уникальное значение для каждой записи

- эти два участка кода эквивалентны, они создают коллекцию UAKGTest, где F1 является первичным ключом. Как мы видим, у нас есть еще одно расширение SQL - клауза WITH KEY. Она может использоваться только для получения коллекции с ключами, синтаксис предложения с этой клаузой следующий:

```
SELECT <selection> <table-expr> WITH KEY <selection>
```

Как и в SQL92, последовательность полей тоже может образовывать ключ:

```
UAKGKeyCollection_var collection = queryManager->
    create_key_collection_by_query(
        "select * from X with key x1,x2");
```

Заметим, что правильность указания ключей лежит на совести прикладного программиста: UAKGQueryService использует эту информацию, но никак не проверяет ее соответствие действительной организации базы данных. При этом ключ - это список полей из тех (и только), которые заданы в базовой структуре UAKGKeyCollection, т.е. keyDescription - это подмножество recordDescription.

11.7.2 Работа с UAKGKeyCollection

Для того, чтобы получать, обновлять и удалять записи из БД по заданному значению ключевых полей можно использовать следующие методы:

- `retrieve_record_with_key(const Record& key)` – выбрать запись, соответствующую ключу `key`.
- `retrieve_records_with_keys(const OctSeq& keys)` – выбрать последовательность записей по последовательности соответствующих ключей.
- `update_record_with_key(const Record& newRecord, const Record& key)`
- занести новое значение в запись с ключом `key`
- `update_records_with_keys(const OctSeq& records)`
- `remove_records_with_keys(const OctSeq& keys)` удалить записи с соответствующими ключами.

Кроме того, UAKGKeyCollection предоставляет некоторые вспомогательные методы:

- `get_key_description()` - получить описание ключа.
- `extract_keys(const OctSeq& records)` - выделить ключевые поля из последовательности записей.

Пример:

```

UAKGKeyCollection_var collection_ =
    queryManager->create_key_collection_by_query(
        "select F1,F2 from UAKGTEST where F1=1 with key F2");
Record_var inpRecord_ = new Record;
inpRecord_->length(1);
FieldValueAccess::setString(inpRecord_[0], "test");
collection_->remove_record_with_key( inpRecord );
collection_->destroy();

```

11.8 Использование UAKGCollectionListeners

В прикладных программах часто бывает удобно получать нотификации о значащих событиях в жизни коллекции: получении или удалении новых элементов. Для этого предназначен механизм "Слушателей" коллекции: пользователь может добавить свой интерфейс обратного вызова, который будет вызываться при наступлении подобных событий.

```

interface UAKGCollectionListener
{
    // вызывается при добавлении элементов
    void elements_added(in OctSeq elements);
    // вызывается при изменении элементов
    void elements_updated(in OctSeq prev_elements,
        in OctSeq new_elements);
    // вызывается при удалении элементов
    void elements_removed(in OctSeq elements);
    // вызывается при удалении всех элементов
    void all_elements_removed();
    // вызывается при разрушении коллекции, как CORBA объекта.
    void collection_destroyed();
};

```

Прикладной программист должен реализовать этот интерфейс и подключить его к коллекции для получения модификаций с помощью вызова метода:

```

unsigned long UAKCollection::add_listener(
    in UAKGCollectionListener listener,
    in unsigned short eventMask);

```

Этот метод возвращает число, которым коллекция идентифицирует этот слушатель. Им можно воспользоваться для того, чтобы отсоединиться от коллекции, вызвав метод:

```

boolean UAKCollection::remove_listener(in unsigned long listenerIndex);

```

Наконец, `eventMask` – это маска событий, которые хочет получать ваш слушатель, которая должна быть битовой комбинацией

При использовании механизма нотификаций следует помнить, что накладные расходы сервиса на сбор данных для нотификации относительно велики и что синхронные вызовы callback функций клиентов - не самое масштабируемое решение. Если вы хотите, чтобы несколько клиентов получали нотификации, лучше не регистрируйте для каждого клиента свой слушатель, а используйте дополнительное звено, которое получает нотификации и рассылает их клиентам асинхронными методами (например - с помощью CORBA Event Service)

11.9 Ограничения

В настоящее время один экземпляр коллекции может использоваться в только одной транзакции в одно и то же время.

12 Транзакции

В UAKGQueryService реализованы 2 режима транзакций:

1. XA транзакции, использующие XA ресурс нижележащей БД и XA монитор ORBacus Transaction Service.
2. Собственный менеджер транзакций, использующий прямые транзакции БД.

XA менеджер транзакций удобно применять в том случае, если ваш сервер приложений работает с одной DB в XA окружении, с другой стороны, менеджер транзакций UAKGQuery более эффективный.

Рассмотрим каждый из этих режимов подробнее:

12.0.1 XA транзакции

Использование XA транзакций инициируется следующими ключами:

- ORACLE_XA=<xa-open-string> - для Oracle
- INTERBASE_XA=<xa-open-string> - для Interbase.

Где <xa-open-string> – строка XA, в которой указаны параметры соединения базы данных. Для более подробного рассмотрения XA строки, обратитесь к документации соответствующей БД.

- Oracle: http://technet.oracle.com/doc/server.815/a68003/01_app1x.htm#619504
- Interbase InterBase Programming Guide.

При работе с XA транзакциями существуют следующие ограничения:

1. так как соединения с сервером производятся XA монитором, то только одни параметры подключения (login и пароль пользователя базы) могут использоваться. Параметры `username` и `password` `DBConnectionManager::createQueryManager` игнорируются. Если вы хотите добиться одновременной работы нескольких подключений в XA режиме, вы должны запустить несколько копий `UAKGQueryService`.
2. В XA приложении нельзя использовать DDL предложения.
3. В XA приложении нельзя вызывать методы `UAKGQuery` вне контекста транзакции.
4. Так как XA монитор должен быть осведомлен о создании каждого нового потока исполнения, XA CORBA сервер должен использовать режим потоков исполнения `thread_per_request`

12.0.2 UAKG OTS транзакции

Режим UAKG OTS транзакций используется по умолчанию.

В отличие от XA транзакций, при использовании UAKG транзакций параметры подключения к базе данных определяются на этапе создания `QueryManager`-а. При этом методы работы с БД можно вызывать как в контексте глобальной транзакции, так и без: в этом случае обращение к БД будет рассматриваться как локальная транзакция, которая автоматически завершается по окончании работы запроса.

12.1 Типичное использование транзакций

Ниже приведен фрагмент кода, иллюстрирующий типичное использование OTS:

```
Object_var obj = orb->resolve_initial_references("TransactionCurrent");
CosTransactions::Current_var current =
    CosTransactions::Current::_narrow(obj);

current->begin();
try {

    // do something with db:
    queryManger->evaluate_query(query1,"SQL92",query1ParamsDescriptions,
                                query1Params);
    queryManger->evaluate_query(query2,"SQL92",query2ParamsDescriptions,
                                query2Params);

    current->commit();
}
```

```

}catch(const QueryProcessingError& ex){

    current->rollback();

}catch(...){
    cerr << "Fatal: unknown exception";
    current->rollback();
}

```

Т. е. типичное применение транзакций – обеспечение атомарности последовательности операции над базой данных: операция либо полностью завершится, либо будет полностью отменена. Для более подробной информации, рекомендуем [4] , [8].

13 Приложение 1: IDL спецификации

13.1 CosQueryCollection

```

#ifndef __COSQUERYCOLLECTION_IDL
#define __COSQUERYCOLLECTION_IDL

/*
 * module CosQueryCplection.
 * writeln from specifications of OMG froup for CORBA Query Service.
 * (C) Ruslan Shevchenko <Ruslan@Shevchenko.Kiev.UA>
 * (C) GradSoft <info@gradsoft.com.ua>
 * 1998, 1999, 2000, 2001
 * $Id: ProgrammingGuide_rus.tex,v 1.29 2002/01/30 14:18:52 rssh Exp $
 */

#ifndef __CosQueryIDLConfigV2_idl
#include <CosQueryIDLConfigV2.idl>
#endif

#ifdef HAVE_ORB_IDL
#include <orb.idl>
#endif

#pragma prefix "gradsoft.kiev.ua"

/**
 *
 */
module UAKGQuery
{
    ///

```

```

    /**
     * UAKGQuery sequence of octets
     **/
    typedef sequence<octet> OctSeq;

};

#include <CosQuery.idl>

#pragma prefix "omg.org"

/**
 * module CosQueryCpllection.
 * data definitions for CORBA Query Service
 * writeln from specifications of OMG group .
 * (C) Ruslan Shevchenko <Ruslan@Shevchenko.Kiev.UA>
 * 1998, 1999, 2000
 **/
module CosQueryCollection {

    ///
    exception ElementInvalid {};
    ///
    exception IteratorInvalid {};
    ///
    exception PositionInvalid {};

    /**
     * possible DB field types
     **/
    enum FieldType {
        ///
        TypeBoolean,
        ///
        TypeChar,
        ///
        TypeOctet,
        ///
        TypeShort,
        ///
        TypeUShort,
        ///
        TypeLong,
    };
};

```

```

    ///
    TypeULong,
    ///
    TypeFloat,
    ///
    TypeDouble,
    ///
    TypeString,
    ///
    TypeObject,
    ///
    TypeSmallInt,
    ///
    TypeInteger,
    ///
    TypeReal,
    ///
    TypeDoublePrecision,
    ///
    TypeCharacter,
    ///
    TypeDecimal,
    ///
    TypeNumeric,
    ///
    TypeDateTime,
    ///
    TypeRaw,
    ///
    TypeWString,
    ///
    TypeBlob,
    ///
    TypeClob,
    ///
    TypeWclob
};

/**
 * decimal field.
 **/
struct Decimal {
    /**
     * precision of number.
     **/
    long precision;

```

```

/**
 * scale of number
 */
long scale;
/**
 * value in BCD format.
 */
sequence<octet> value;
};

/**
 * type, corresponding to DATE field.
 * (all values are start from 1)
 */
struct DateTime {
    ///
    short year;
    ///
    octet month;
    ///
    octet day;
    ///
    octet hour;
    ///
    octet minute;
    ///
    octet second;
};
///
exception ForReadingOnly {};

///
exception ForWritingOnly {};

/**
 *
 */
interface Blob {
    ///
    unsigned long length() raises(CosQuery::QueryProcessingError);
    ///
    UAKGQuery::OctSeq fetch_chunk(in unsigned long chunkSize
        , out boolean more)
        raises(CosQuery::QueryProcessingError, ForWritingOnly);
    ///
    void put_chunk(in UAKGQuery::OctSeq data)

```

```

        raises(CosQuery::QueryProcessingError, ForReadingOnly);
};

/**
 *
 **/
interface Clob {
    ///
    unsigned long length() raises(CosQuery::QueryProcessingError);
    ///
    string fetch_chunk(in unsigned long chunkSize, out boolean more)
        raises(CosQuery::QueryProcessingError
            , ForWritingOnly);
    ///
    void put_chunk(in string data) raises(CosQuery::QueryProcessingError
        , ForReadingOnly);
};

/**
 *
 **/
interface Wclob {
    ///
    unsigned long length() raises(CosQuery::QueryProcessingError);
    ///
    wstring fetch_chunk(in unsigned long chunkSize, out boolean more)
        raises(CosQuery::QueryProcessingError
            , ForWritingOnly);
    ///
    void put_chunk(in wstring data) raises(CosQuery::QueryProcessingError
        , ForReadingOnly);
};

/**
 * what can be not null value in DB:
 **/
union Value switch(FieldType) {
    ///
    case TypeBoolean: boolean b;
    ///
    case TypeChar: char c;
    ///

```

```

case TypeOctet: octet o;
///
case TypeShort : short s;
///
case TypeUShort : unsigned short us;
///
case TypeLong : long l;
///
case TypeULong : unsigned long ul;
///
case TypeFloat : float f;
///
case TypeDouble : double d;
///
case TypeString : string str;
///
case TypeObject : Object obj;
///
case TypeSmallInt : short si;
///
case TypeInteger : long i;
///
case TypeReal : float r;
///
case TypeDoublePrecision : double dp;
///
case TypeCharacter : string ch;
///
case TypeDecimal : Decimal dec;
///
case TypeNumeric : Decimal n;
///
case TypeDateTime : DateTime dt;
///
case TypeRaw : sequence<octet> raw;
///
case TypeWString : wstring wstr;
///
case TypeBlob : Blob bl;
///
case TypeClob : Clob cl;
///
case TypeWclob : Wclob wcl;
};

```

```

///  

typedef boolean Null;  

  

/**  

 * this union represent one field in DB  

 **/  

union FieldValue switch(Null){  

    case FALSE : Value v;  

};  

  

/**  

 * one record in DB  

 **/  

typedef sequence<FieldValue> Record;  

  

/**  

 *  

 **/  

typedef string Istring;  

  

};  

  

#endif

```

13.2 CosQuery

```

#ifndef __COSQUERY_IDL
#define __COSQUERY_IDL
  

/*  

 * module CosQuery.  

 * from specifications of OMG group for CORBA Query Service.  

 * (C) Ruslan Shevchenko <Ruslan@Shevchenko.Kiev.UA>, 1998, 1999, 2000  

 * (C) GradSoft, 2001  

 * $Id: ProgrammingGuide_rus.tex,v 1.29 2002/01/30 14:18:52 rssh Exp $  

 */  

  

#include <CosQueryCollection.idl>  

  

#pragma prefix "omg.org"  

  

/**  

 * CosQuery: legacy definitions from OMG Query Service.  

 **/

```

```

module CosQuery {

    ///
    exception QueryInvalid
    {
        ///
        string why;
    };

    ///
    exception QueryProcessingError
    {

        ///
        string why;
    };

    ///
    exception QueryTypeInvalid { };

    ///
    enum QueryStatus
    {
        ///
        complete,
        ///
        incomplete
    };

};

#endif

```

13.3 UAKGQuery

```

#ifndef __UAKGQUERY_IDL
#define __UAKGQUERY_IDL

/*
 * IDL Definitions for UAKGQueryService
 * (C) Ruslan Shevchenko <Ruslan@Shevchenko.Kiev.UA>, 1998 - 2002
 * (C) Grad-Soft Ltd <info@gradsoft.kiev.ua> 2001 - 2002
 * $Id: ProgrammingGuide_rus.tex,v 1.29 2002/01/30 14:18:52 rss Exp $
 */

#ifdef CORBA_HAVE_OTS

```

```

#ifndef __COSTRANSACTIONS_IDL
#include <CosTransactions.idl>
#endif
#endif

#ifndef __COSQUERY_IDL
#include <CosQuery.idl>
#endif

#ifndef __RC_IDL
#include <RC.idl>
#endif

#pragma prefix "gradsoft.kiev.ua"

/**
 * UAKGQuery module
 * (GradSoft-specific type of UAKGQuery implementation).
 */
module UAKGQuery
{

    ///
    typedef sequence<string> StringSeq;

    /**
     * struct for description of field size.
     * name: name of field in DB.
     * ValueType: field type.
     * size: size of field in bytes. (for strings: include \0, i. e.
     *       for VARCHAR(x) size is x+1
     * precision (have sense only for NUMERIC types) - precision.
     * scale (have sense only for NUMERIC types) - scale, as signed byte.
     */
    struct FieldDescription{
        /// name of field in db
        string      name;
        /// field type
        CosQueryCollection::FieldType      type;
        /// size of field in bytes (for strings: include trailing \0, i. e.
        /// for VARCHAR2(x) size is x+1
        unsigned long  size;
        /// precision (have sense only for numeric types)
        unsigned short precision;
        /// scale (have sense only for numeric types)
        short          scale;
    };
}

```

```

};

///
typedef sequence<FieldDescription> RecordDescription;

/*
struct ParameterDescription
{
    FieldDescription    field;
    CORBA::ParameterMode mode;
};
typedef sequence<ParameterDescription> ParametersDescription;
*/

///
struct QueryError
{
    /// error code: 0 is OK.
    long    errorCode;
    /// error message
    string  errorMessage;
    /// sql string, during execution of which error causes.
    string  sqlString;
    /// db name
    string  dbName;
    /// error code from underlying database
    long    dbErrorCode;
};

///
exception QueryNotPrepared {};
///
exception InvalidParameterName{};
///
exception InvalidParameterType{};
///
exception PropertyNotDefined{};

/**
 * Hight level interface for evaluationg SQL queries
 **/
interface QueryEvaluator
#ifdef CORBA_HAVE_OTS
    //      :CosTransactions::TransactionalObject
    // in all ORB-s context is passing unconditionally

```

```

#endif
{

/**
 * evaluate query <code> queryText </code> and return result as
 * RC-coded octet sequence.
 * @param queryText -- text of query
 * @param queryFlags -- flags for query executing
 * @param recordDescription -- description of input parameters.
 * @param params -- input parameters as RC-coded octet sequence
 * @return result of query
 */
OctSeq evaluate_rc(in string queryText, in string queryFlags,
                  in RecordDescription recordDescription_,
                  in OctSeq params)
    raises(CosQuery::QueryTypeInvalid,
           CosQuery::QueryInvalid,
           CosQuery::QueryProcessingError);

/**
 * evaluate query <code> queryText </code> and return result as
 * sequence of records.
 * @param queryText -- text of query
 * @param queryFlags -- flags for query executing
 * @param recordDescription -- description of input parameters.
 * @param params -- input parameters as record sequence.
 * @return result of query
 */
RC::RecordSeq evaluate_records(in string queryText, in string queryFlags,
                              in RecordDescription recordDescription_,
                              in RC::RecordSeq params)
    raises(CosQuery::QueryTypeInvalid,
           CosQuery::QueryInvalid,
           CosQuery::QueryProcessingError);

/**
 * evaluate query <code> queryText </code> and return result as
 * RC-coded octet sequence.
 * @param queryText -- text of query
 * @param queryFlags -- flags for query executing
 * @param recordDescription_ -- description of input parameters.
 * @param params -- input parameters as record .
 * @return result of query
 */

```

```

RC::RecordSeq evaluate_record(in string queryText,
                              in string queryFlags,
                              in RecordDescription recordDescription_,
                              in CosQueryCollection::Record params)
    raises(CosQuery::QueryTypeInvalid,
           CosQuery::QueryInvalid,
           CosQuery::QueryProcessingError);

/**
 * evaluate query <code> queryText </code> without bind parameters
 * and return result as RC-coded octet sequence.
 *@param queryText -- text of query
 *@param queryFlags -- flags for query executing
 *@return result of query
 **/
OctSeq evaluate_rc_e(in string queryText, in string queryFlags)
    raises(CosQuery::QueryTypeInvalid,
           CosQuery::QueryInvalid,
           CosQuery::QueryProcessingError);

/**
 * evaluate query <code> queryText </code> without bind parameters
 * and return result as sequence of records.
 *@param queryText -- text of query
 *@param queryFlags -- flags for query executing
 *@return result of query
 **/
RC::RecordSeq evaluate_records_e(in string queryText, in string queryFlags)
    raises(CosQuery::QueryTypeInvalid,
           CosQuery::QueryInvalid,
           CosQuery::QueryProcessingError);

/**
 * evaluate query <code> queryText </code> and fill out and inout
 * parameters of query, return result as RC-coded octet sequence.
 *@param queryText -- text of query
 *@param queryFlags -- flags for query executing
 *@param recordDescription_ -- description of input parameters.
 *@param params -- input parameters as record .
 *@return result of query
 **/
OctSeq evaluate_rc_inout(in string queryText, in string queryFlags,
                        in RecordDescription recordDescription_,
                        inout OctSeq params)
    raises(CosQuery::QueryTypeInvalid,

```

```

        CosQuery::QueryInvalid,
        CosQuery::QueryProcessingError);

/**
 * evaluate query <code> queryText </code> and fill out and inout
 * parameters of query, return result as sequence of records.
 * @param queryText -- text of query
 * @param queryFlags -- flags for query executing
 * @param recordDescription_ -- description of input parameters.
 * @param params -- input parameters as record .
 * @return result of query
 */
RC::RecordSeq evaluate_records_inout(in string queryFlags,
                                     in string queryType,
                                     in RecordDescription recordDescription_,
                                     inout RC::RecordSeq params)
    raises(CosQuery::QueryTypeInvalid,
           CosQuery::QueryInvalid,
           CosQuery::QueryProcessingError);

};

interface Query;
interface QueryManager;

/**
 * this is interface for UAKG Query
 * Query is SQL text with set of parameters: prepare parameters and
 * execute parameters.
 * prepare parameters are descriptions of appropriate execute parameters
 * execute parameters are SQL host variables.
 * i. e. let we have query (SELECT * from T where x=:x and y=:y);
 * than prepare query have type RecordDescription and consist from
 * FieldDescription of :x and :y.
 * execute query are values of :x and :y (or sequence of pair of values
 * for multiple evaluated query).
 */
interface Query
{

    /**
     * @return owner of query
     */

```

```

readonly attribute QueryManager query_mgr;

/**
 *@return text of query.
 */
readonly attribute string queryText;

/**
 * return status of query: i.e:
 * complete when query is executed, otherwise incomplete
 */
CosQuery::QueryStatus get_status ();

/**
 * prepare query for executing.
 * if query have no parameters, paramsDescription must be empty
 * sequence.
 */
void prepare_query(in RecordDescription paramsDescription)
    raises(CosQuery::QueryProcessingError);

/**
 * synonym for prepare_query
 */
void prepare(in RecordDescription paramsDescription)
    raises(CosQuery::QueryProcessingError);

/**
 * execute query
 *@params octSeq_ records of execute parameters, coded as RCSeq
 * (note, that prepare parameters is record descriptio of execute
 * record).
 */
void execute_rc(in OctSeq octSeq_)
    raises(CosQuery::QueryProcessingError);

/**
 * execute query with inout parameters
 *@params octSeq_ records of execute parameters, coded as RCSeq
 */
void execute_rc_inout(inout OctSeq octSeq_)
    raises(CosQuery::QueryProcessingError);

```

```

/**
 * execute query
 * @params records -- query host parameters in RecordSeq
 * (query will be evaluated records.length() times)
 */
void execute_records(in RC::RecordSeq records)
    raises(CosQuery::QueryProcessingError);

/**
 * execute query
 * @params record_ -- query host parameters in one record
 */
void execute_record(in CosQueryCollection::Record record_)
    raises(CosQuery::QueryProcessingError);

///
void execute_records_inout(inout RC::RecordSeq recordSeq_)
    raises(CosQuery::QueryProcessingError);

///
RecordDescription get_result_description()
    raises(CosQuery::QueryProcessingError,
           QueryNotPrepared);

/**
 * get description of records parameters
 * @precondition
 * must be called after prepare
 */
RecordDescription get_parameters_description()
    raises(CosQuery::QueryProcessingError);

///
RC::RecordSeq get_all_parameters_records()
    raises(CosQuery::QueryProcessingError);

///
RC::RecordSeq get_parameters_records(in StringSeq neededFields)
    raises(CosQuery::QueryProcessingError,
           InvalidParameterName);

///
OctSeq get_all_parameters_rc()
    raises(CosQuery::QueryProcessingError);

///

```

```

OctSeq get_parameters_rc(in StringSeq fieldNames)
                        raises(CosQuery::QueryProcessingError,
                               InvalidParameterName);

/**
 * @returns number of fetched rows.
 */
unsigned long get_row_count()
                raises(CosQuery::QueryProcessingError);

/**
 * fetch query result in records.
 * @param numberOfRecords -- number of records to fetch.
 *      0 means, that we want to fetch all records.
 * @param more -- true, if status is incomplete (i.e. we can query
 * more results), otherwise false.
 * @returns fetched rows packed in RC coding to octet sequence.
 */
OctSeq fetch_rc(in unsigned long numberOfRecords, out boolean more)
                raises(CosQuery::QueryProcessingError);

/**
 * synonym for fetch_rc.
 */
OctSeq get_result_rc(in unsigned long numberOfRecords)
                    raises(CosQuery::QueryProcessingError);

/**
 * fetch query result in records.
 * @param numberOfRecords -- number of records to fetch.
 *      0 means, that we want to fetch all records.
 * @param more -- true, if status is incomplete (i.e. we can query
 * more results), otherwise false.
 * @returns fetched records.
 */
RC::RecordSeq fetch_records(in unsigned long numberOfRecords,
                            out boolean more)
                        raises(CosQuery::QueryProcessingError);

/**
 * synonym for fetch_records
 */
RC::RecordSeq get_result_records(in unsigned long numberOfRecords)
                                raises(CosQuery::QueryProcessingError);

```

```

/**
 * skip N records without retrieving.
 * @returns actual number of skipped records.
 */
unsigned long skip(in unsigned long numberOfRecords,
                  out boolean more)
                  raises(CosQuery::QueryProcessingError);

/**
 * request Blob for filling query parameters
 * @returns empty Blob for writing only.
 */
CosQueryCollection::Blob create_blob()
    raises(CosQuery::QueryProcessingError);

/**
 * request Clob for filling query parameters
 * @returns empty Clob for writing only.
 */
CosQueryCollection::Clob create_clob()
    raises(CosQuery::QueryProcessingError);

/**
 * request Wclob for filling query parameters
 * @returns empty Wclob for writing only.
 */
CosQueryCollection::Wclob create_wclob()
    raises(CosQuery::QueryProcessingError);

/**
 * @return last error.
 * if Query is ok, code in error is 0.
 */
QueryError get_last_error();

/**
 * destroy query, which not longer needed
 */
void destroy();

};

//
// UAKGQueryCollections

```

```

//

interface UAKGCollectionListener;
interface UAKGIterator;

///
exception ReadOnlyCollection {};
///
exception ReadOnlyIterator {};
///
exception KeyNotFound {};

///
interface UAKGCollection: QueryEvaluator
{

    ///
    readonly attribute string  selectQueryText;
    ///
    readonly attribute string  selectDistinctQueryText;
    ///
    readonly attribute string  selectRangeQueryText;
    ///
    readonly attribute string  countQueryText;
    ///
    readonly attribute string  insertQueryText;
    ///
    readonly attribute string  removeAllQueryText;
    ///
    readonly attribute string  orderByText;

    ///
    RecordDescription  getRecordDescription()
                        raises(CosQuery::QueryProcessingError);

    ///
    void  set_readonly(in boolean ronly)
          raises(ReadOnlyCollection);

    ///
    boolean  is_readonly();

    /**
     * true, is select collection is ordered.
     **/
}

```

```

readonly attribute boolean sorted;

/**
 * add record
 */
void add_record(in CosQueryCollection::Record element)
    raises(CosQueryCollection::ElementInvalid,
           CosQuery::QueryProcessingError,
           ReadOnlyCollection);

/**
 * add records
 */
void add_records(in RC::RecordSeq elements)
    raises(CosQueryCollection::ElementInvalid,
           CosQuery::QueryProcessingError,
           ReadOnlyCollection);

/**
 * add records coded in RC sequence
 */
void add_rc(in OctSeq rc)
    raises(CosQueryCollection::ElementInvalid,
           CosQuery::QueryProcessingError,
           ReadOnlyCollection);

//
// retrieve record number

/**
 *return number of records in collection
 *@returns number of records in collection
 */
unsigned long get_number_of_records()
    raises(CosQuery::QueryProcessingError);

//
// retrieve records

/**
 * retrieve records by filter.
 *@param where-filter : logical expression for selection of records
 * to delete (in SQL-like DBs is context of where clause)

```

```

*TODO: what it return is it correct ?
*/
OctSeq    retrieve_by_filter(in string where_filter)
           raises(CosQuery::QueryProcessingError);

/**
 * retrieve records by pattern.
 * @param : pattern
 *TODO: what it return is it correct ?
 */
OctSeq    retrieve_by_pattern(in CosQueryCollection::Record pattern)
           raises(CosQuery::QueryProcessingError,
                  CosQueryCollection::ElementInvalid);

//
// replacing
//

/**
 * update records by pattern
 * @param newRecord -- new record instead pattern matched
 * @param pattern -- pattern for matching
 */
void      update_by_pattern(in CosQueryCollection::Record newRecord,
                           in CosQueryCollection::Record pattern )
           raises(CosQuery::QueryProcessingError,
                  CosQueryCollection::ElementInvalid,
                  ReadOnlyCollection);

/**
 * update records by filter
 * @param newRecord -- new record instead filter matched
 * @param filter -- condition
 */
void      update_by_filter( in CosQueryCollection::Record newRecord,
                           in string filter )
           raises(CosQuery::QueryProcessingError,
                  CosQueryCollection::ElementInvalid,
                  ReadOnlyCollection);

//
// removing
//

```

```

/**
 * remove all records from collection
 */
void      remove_all_records()
           raises(CosQuery::QueryProcessingError,
                 ReadonlyCollection);

/**
 * remove records with same value as <code> record_ </code>
 * @param record_ - value of record to be removed.
 */
void      remove_record(in CosQueryCollection::Record record_)
           raises(CosQuery::QueryProcessingError,
                 CosQueryCollection::ElementInvalid,
                 ReadonlyCollection);

/**
 * remove records with are satisficated to <code> filter </code>
 * @param filter - logical expression for selectiong removed records.
 */
void      remove_records_by_filter(in string filter)
           raises(CosQuery::QueryProcessingError,
                 ReadonlyCollection);

/**
 * remove records with are match pattern <code> pattern </code>
 * @param pattern - pattern to match.
 */
void      remove_records_by_pattern(in CosQueryCollection::Record pattern)
           raises(CosQuery::QueryProcessingError,
                 ReadonlyCollection);

//
// elements ordering
//

/**
 * sort - set new order expression
 * @param order_expressinon - new expresiion for ORDER BY clause
 */
void      sort(in string order_expression)
           raises(CosQuery::QueryProcessingError);

//
// access interfaces factories

```

```

//

/**
 * create iterator
 */
UAKGIterator    create_iterator();

/**
 * create iterator which iterate records, matched for pattern
 */
UAKGIterator    create_iterator_by_pattern(
                    in CosQueryCollection::Record pattern)
                raises(CosQueryCollection::ElementInvalid,
                    CosQuery::QueryProcessingError);

/**
 *
 * subquery must be specified in next form:
 * <code>
 *   select <field_list> from <table_list>
 *     where <conditions> [order by <field_list>]
 * </code>
 */
UAKGCollection create_subcollection(in string subquery)
                raises(CosQuery::QueryInvalid,
                    CosQuery::QueryProcessingError);

///
UAKGCollection create_subcollection_by_pattern(
                    in CosQueryCollection::Record pattern)
                raises(CosQuery::QueryInvalid,
                    CosQuery::QueryProcessingError,
                    CosQueryCollection::ElementInvalid);

/**
 * add listener to collection events
 */
unsigned long  add_listener(in UAKGCollectionListener listener,
                            in unsigned short eventMask);

/**
 * remove listener
 */
boolean       remove_listener(in unsigned long listenerIndex);

```

```

/**
 * destroy collection and free server resources, associated with
 * this collection.
 */
void      destroy();
};

///
interface UAKGCollectionListener
{
    ///
    void  elements_added(in OctSeq elements);
    ///
    void  elements_updated(in OctSeq prev_elements,
                          in OctSeq new_elements);
    ///
    void  elements_removed(in OctSeq elements);
    ///
    void  all_elements_removed();
    ///
    void  collection_destroyed();
};

struct ListenersSeqStruct
{
    UAKGCollectionListener listener;
    unsigned short         mask;
};

typedef sequence<ListenersSeqStruct> UAKGCollectionListeners;

/**
 * Iterator for retrieving data
 */
interface UAKGIterator
{
    /**
     * are we situated at the end of data set ?
     */
    readonly attribute boolean  end;

    /**
     * fetch n records as RC-coded octet sequence
     * @param n - number of records to fetch
     * @param more - set to true, if we not at end of collection.
     * @returns fetched records.

```

```

    */
    OctSeq      fetch_rc(in unsigned long n, out boolean more);

/**
 * fetch n records as records sequence
 * @param n - number of records to fetch
 * @param more - set to true, if we not at the end of collection.
 * @returns fetched records.
 */
    RC::RecordSeq  fetch_records(in unsigned long n, out boolean more);

/**
 * skip n records
 * @param n - number of records to skip
 * @param more - set to true, if we not at the end of collection.
 * @returns actual number of skipped records.
 */
    unsigned long  skip(in unsigned long n, out boolean more);

/**
 * destroy iterator and free associated server resources.
 */
    void destroy();
};

/**
 * Collection of records with unique keys.
 */
interface UAKGKeyCollection: UAKGCollection
{

    ///
    RecordDescription  get_key_description();

    ///
    CosQueryCollection::Record
        retrieve_record_with_key(in CosQueryCollection::Record key)
            raises(CosQuery::QueryProcessingError);

    ///
    void update_record_with_key(in CosQueryCollection::Record newRecord,
                               in CosQueryCollection::Record key)
        raises(CosQuery::QueryProcessingError, KeyNotFound);

    ///

```

```

void remove_record_with_key(in CosQueryCollection::Record key)
    raises(CosQuery::QueryProcessingError);

///
OctSeq retrieve_records_with_keys(in OctSeq keys)
    raises(CosQuery::QueryProcessingError);

///
void update_records_with_keys(in OctSeq records)
    raises(CosQuery::QueryProcessingError);

///
void remove_records_with_keys(in OctSeq keys)
    raises(CosQuery::QueryProcessingError);

};

/**
 * factory for collection interfaces
 */
interface UAKGCollectionFactory
{

/**
 * queryText - select <field_list> from <table_list> where <conditions> [order by <field.
 */
UAKGCollection create_collection( in string queryText )
    raises(CosQuery::QueryInvalid,
          CosQuery::QueryProcessingError);

/**
 * queryText - select <field_list> from <table_list> where <conditions> [order by <field.
 */
UAKGKeyCollection create_key_collection(
    in string queryText
    )
    raises(CosQuery::QueryInvalid,
          CosQuery::QueryProcessingError);

///
UAKGCollection create_collection_by_parts(
    in string selectPartText,
    in string fromPartText,
    in string wherePartText,

```

```

        in string orderByPartText)
        raises(CosQuery::QueryInvalid,
              CosQuery::QueryProcessingError);

    ///
    UAKGKeyCollection create_key_collection_by_parts(
        in string selectPartText,
        in string fromPartText,
        in string wherePartText,
        in string orderByPartText,
        in string keysPartText)
        raises(CosQuery::QueryInvalid,
              CosQuery::QueryProcessingError);

};

/**
 * interface for our QueryManager.
 */
interface QueryManager: QueryEvaluator,
                       UAKGCollectionFactory
{
    ///
    string get_username() raises(CosQuery::QueryProcessingError);
    ///
    string get_dblink() raises(CosQuery::QueryProcessingError);

    ///
    readonly attribute unsigned long number_of_queries;

    ///
    Query create_query(in string query, in string flags)
        raises(CosQuery::QueryTypeInvalid,
              CosQuery::QueryInvalid);

    ///
    Query create(in string query, in string flags)
        raises(CosQuery::QueryTypeInvalid,
              CosQuery::QueryInvalid);

    ///
    string get_property(in string property_name)
        raises(PropertyNotDefined);

    ///
    long get_next_sequence_value(in string sequence_name)

```

```

        raises(CosQuery::QueryProcessingError,
              CosQuery::QueryInvalid,
              CosQuery::QueryTypeInvalid);

    ///
    void destroy();

};

///
exception QueryManagerNotFound {};

typedef sequence<QueryManager> UAKGQueryManagerSeq;

///
interface DBConnectionManager
{
    ///
    QueryManager createQueryManager(in string login, in string password,
                                    in string db_name, in string drv_name,
                                    in string implementation_specific_data)
        raises(QueryManagerNotFound,
              CosQuery::QueryProcessingError);

    /**
     * shutdown query service.
     */
    void shutdown();

};

};

#endif

```

13.4 RC.idl

```

#ifndef __RC_IDL
#define __RC_IDL
/*
 * definitions and pseudo-interfaces for custom Record Marshalling.
 * (C) Ruslan Shevchenko <Ruslan@Shevchenko.Kiev.UA>, 1999
 * (C) GradSoft, 2001
 * $Id: ProgrammingGuide_rus.tex,v 1.29 2002/01/30 14:18:52 rssh Exp $
 */

```

```

#ifndef __COSQUERYCOLLECTION_IDL
#include <CosQueryCollection.idl>
#endif

#pragma prefix "gradsoft.kiev.ua"

/**
 * pseudo-interfaces for custom Record Marshalling
 * The main entity is: RC-coded octet sequence.
 * We provide 2 pseudo-interfaces: RCReader and RCWriter
 * for reading/writing from/to RCSeq.
 */
module RC
{
    ///
    typedef sequence<octet> OctetSeq;
    //typedef CosQueryCollection::Record Record;
    ///
    typedef sequence<CosQueryCollection::Record> RecordSeq;

    //typedef CosQueryCollection::Decimal Decimal;

    /**
     * thrown, when Reader discovered error in OctSeq.
     */
    exception BadOctSeq
    {
        /**
         * position of read failure (in bytes).
         */
        long pos;
        /**
         * what was happened ?
         */
        string reason;
    };

    ///
    exception TypeNotImplemented
    {
        ///
        CosQueryCollection::FieldType fieldType;
    };
};

```

```

///
exception FieldValueIsNull {};
///
exception InvalidPosition {};

/**
 * header of RC-coded octet sequence.
 */
struct RCHeader
{
    ///
    octet version;
    /// number of records in sequence.
    /// (if -1, than number of records is unknown).
    long nRecords;
    /// number of fields in one record.
    unsigned long nFields;
};

///
exception InvalidHeadData {};

/**
 * this pseudointerface must be mapped to RCWriter static class
 * in host language.
 */
interface Writer // pseudo
{

    /**
     *write header of Octet Sequence to octSeq_.
     *@param nRecords - number of records to be coded.
     *@param nFields - number of fields in one record.
     *@param pos - position (input really ignored, on output it
     * is settet to first position after header).
     *@param octSeq_ - sequence, in which we code.
     */
    void writeHeader(in long nRecords, in unsigned long nFields,
                    inout unsigned long pos, inout OctetSeq octSeq_)
        raises (InvalidHeadData);

    ///
    void writeHead(inout unsigned long pos, inout OctetSeq octSeq_)
        raises (InvalidHeadData);
}

```

```

///
void writeRecord(in CosQueryCollection::Record record,
                inout unsigned long pos,
                inout OctetSeq octSeq_ )
                raises(TypeNotImplemented);

///
void writeRecordSeq(in RecordSeq recordSeq_)
                raises(TypeNotImplemented);

///
void writeBoolean(in boolean value, inout unsigned long pos,
                 inout OctetSeq octSeq_);

///
void writeChar(in char value, inout unsigned long pos,
               inout OctetSeq octSeq_);

///
void writeShort(in short value, inout unsigned long pos,
                inout OctetSeq octSeq_);

///
void writeLong(in long value, inout unsigned long pos, inout OctetSeq octSeq_);

///
void writeULong(in unsigned long value, inout unsigned long pos,
                inout OctetSeq octSeq_);

///
void writeFloat(in float value, inout unsigned long pos,
                inout OctetSeq octSeq_);

///
void writeDouble(in float value, inout unsigned long pos,
                 inout OctetSeq octSeq_);

///
void writeString(in string value, inout unsigned long pos,
                 inout OctetSeq octSeq_);

///
void writeWString(in wstring value, inout unsigned long pos,
                  inout OctetSeq octSeq_);

///

```

```

void writeObject(in Object value, inout unsigned long pos,
                inout OctetSeq octSeq_);

///
void writeDecimal(in CosQueryCollection::Decimal value,
                 inout unsigned long pos,
                 inout OctetSeq octSeq_);

///
void writeRaw(in OctetSeq value, inout unsigned long pos,
             inout OctetSeq octSeq_);

///
void writeDateTime(in CosQueryCollection::DateTime value,
                  inout unsigned long pos,
                  inout OctetSeq octSeq_);

///
void writeFieldValue(in CosQueryCollection::FieldValue value,
                    inout unsigned long pos, inout OctetSeq octSeq_);

///
void writeNullField(inout unsigned long pos, inout OctetSeq octSeq_);

///
void writeBooleanField(in boolean value,
                       inout unsigned long pos, inout OctetSeq octSeq_);

///
void writeCharField(in char value,
                   inout unsigned long pos, inout OctetSeq octSeq_);

///
void writeOctetField(in char value,
                    inout unsigned long pos, inout OctetSeq octSeq_);

///
void writeShortField(in short value,
                    inout unsigned long pos, inout OctetSeq octSeq_);

///
void writeUShortField(in unsigned short value,
                     inout unsigned long pos, inout OctetSeq octSeq_);

///
void writeLongField(in long value,

```

```

        inout unsigned long pos, inout OctetSeq octSeq_);

///
void writeUlongField(in unsigned long value,
                    inout unsigned long pos, inout OctetSeq octSeq_);

///
void writeFloatField(in float value,
                    inout unsigned long pos, inout OctetSeq octSeq_);

///
void writeDoubleField(in double value,
                    inout unsigned long pos, inout OctetSeq octSeq_);

///
void writeStringField(in string value,
                    inout unsigned long pos, inout OctetSeq octSeq_);

///
void writeObjectField(in Object value,
                    inout unsigned long pos, inout OctetSeq octSeq_);

///
void writeDecimalField(in CosQueryCollection::Decimal value,
                    inout unsigned long pos, inout OctetSeq octSeq_);

///
void writeRawField(in OctetSeq value,
                    inout unsigned long pos, inout OctetSeq octSeq_);

///
void writeLongRawField(in OctetSeq value,
                    inout unsigned long pos, inout OctetSeq octSeq_);

///
void writeBlob(in CosQueryCollection::Blob value,
                    inout unsigned long pos, inout OctetSeq octSeq_);

///
void writeClob(in CosQueryCollection::Clob value,
                    inout unsigned long pos, inout OctetSeq octSeq_);

///
void writeWclob(in CosQueryCollection::Wclob value,
                    inout unsigned long pos, inout OctetSeq octSeq_);

///
OctetSeq copyStream(in unsigned long from_pos, in unsigned long to_pos,
                    in OctetSeq octSeq_)

```

```

        raises(InvalidPosition);

};

/**
 * this pseudointerface must be mapped to RCReader static class
 * in host language.
 */
interface Reader
{

    ///
    void    readHeader(inout RCHheader header, inout unsigned long pos,
                      in OctetSeq octSeq_)
                raises(BadOctSeq);

    ///
    CosQueryCollection::Record readRecord(inout unsigned long pos, in OctetSeq octSeq_)
                raises(BadOctSeq);

    ///
    RecordSeq readRecordSeq(inout unsigned long pos, in OctetSeq octSeq_)
                raises(BadOctSeq);

    ///
    CosQueryCollection::FieldValue readField(inout unsigned long pos,
                                             in OctetSeq octSeq_)
                raises(BadOctSeq);

    /**
     * return true and skip null value, if return was null, otherwise
     * return false and not touch pos.
     */
    boolean nextFieldIsNull(inout unsigned long pos, in OctetSeq octSeq_)
                raises(BadOctSeq);

    ///
    CosQueryCollection::FieldType nextFieldType(inout unsigned long pos,
                                                in OctetSeq octSeq_)
                raises(BadOctSeq);

    ///
    boolean readBooleanField(inout unsigned long pos, in OctetSeq octSeq_)
                raises(BadOctSeq,FieldValueIsNull);

    ///

```

```

void      readBooleanField_inout(inout boolean value,
                                inout unsigned long pos, in OctetSeq octSeq_)
                                raises(BadOctSeq,FieldValueIsNull);

///
char      readCharField(inout unsigned long pos, in OctetSeq octSeq_)
                                raises(BadOctSeq,FieldValueIsNull);

///
void      readCharField_inout(inout char value,
                               inout unsigned long pos, in OctetSeq octSeq_)
                               raises(BadOctSeq,FieldValueIsNull);

///
octet     readOctetField(inout unsigned long pos, in OctetSeq octSeq_)
                                raises(BadOctSeq,FieldValueIsNull);

///
short     readShortField(inout unsigned long pos, in OctetSeq octSeq_)
                                raises(BadOctSeq,FieldValueIsNull);

///
unsigned short  readUShortField(inout unsigned long pos, in OctetSeq octSeq_)
                                raises(BadOctSeq,FieldValueIsNull);

///
long      readLongField(inout unsigned long pos, in OctetSeq octSeq_)
                                raises(BadOctSeq,FieldValueIsNull);

///
unsigned long  readULongField(inout unsigned long pos, in OctetSeq octSeq_)
                                raises(BadOctSeq,FieldValueIsNull);

///
float      readFloatField(inout unsigned long pos, in OctetSeq octSeq_)
                                raises(BadOctSeq,FieldValueIsNull);

///
double     readDoubleField(inout unsigned long pos, in OctetSeq octSeq_)
                                raises(BadOctSeq,FieldValueIsNull);

///
string     readStringField(inout unsigned long pos, in OctetSeq octSeq_)
                                raises(BadOctSeq,FieldValueIsNull);

///

```

```

Object readObjectField(inout unsigned long pos, in OctetSeq octSeq_)
                        raises(BadOctSeq,FieldValueIsNull);

///
CosQueryCollection::Decimal readDecimalField(
                        inout unsigned long pos,
                        in OctetSeq octSeq_)
                        raises(BadOctSeq,FieldValueIsNull);

///
CosQueryCollection::Decimal readNumericField(
                        inout unsigned long pos,
                        in OctetSeq octSeq_)
                        raises(BadOctSeq,FieldValueIsNull);

///
CosQueryCollection::DateTime readDateTimeField(
                        inout unsigned long pos,
                        in OctetSeq octSeq_)
                        raises(BadOctSeq,FieldValueIsNull);

///
OctetSeq readRawField(inout unsigned long pos, in OctetSeq octSeq_)
                        raises(BadOctSeq,FieldValueIsNull);

///
void readRawField_inout(inout OctetSeq value,
                        inout unsigned long pos, in OctetSeq octSeq_)
                        raises(BadOctSeq,FieldValueIsNull);

///
OctetSeq readLongRawField(inout unsigned long pos, in OctetSeq octSeq_)
                        raises(BadOctSeq,FieldValueIsNull);

///
void readLongRawField_inout(inout OctetSeq value,
                        inout unsigned long pos, in OctetSeq octSeq_)
                        raises(BadOctSeq,FieldValueIsNull);

///
string readLongStringField(inout unsigned long pos, in OctetSeq octSeq_)
                        raises(BadOctSeq,FieldValueIsNull);

///
void readLongStringField_inout(inout string value,
                        inout unsigned long pos, in OctetSeq octSeq_)

```

```

                                                    raises(BadOctSeq,FieldValueIsNull);
    ///
    void readBlob(inout CosQueryCollection::Blob value,
                 inout unsigned long pos, in OctetSeq octSeq_)
                 raises(BadOctSeq,FieldValueIsNull);
    ///
    void readClob(inout CosQueryCollection::Clob value,
                 inout unsigned long pos, in OctetSeq octSeq_)
                 raises(BadOctSeq,FieldValueIsNull);
    ///
    void readWclob(inout CosQueryCollection::Wclob value,
                  inout unsigned long pos, in OctetSeq octSeq_)
                  raises(BadOctSeq,FieldValueIsNull);

};

};

#endif

```

14 Приложение 2:

```

RCStream:: Version, RecordArray

Version:: 0x01

RecordArray :: NumberOfRecords[4], RecordHeader , RecordData<1..infinity> ;

Record :: RecordHeader,RecordData;

RecordHeader :: NumberOfFields[1];

RecordData :: FieldBlock<NumberOfFields> ;

FieldBlock :: DataType , DataValue;

DataType ::
    TypeNull      0x00
|   TypeBoolean  0x01
|   TypeChar     0x02
|   TypeOctet    0x03
|   TypeShort    0x04
|   TypeUShort   0x05

```

```

|     TypeLong      0x06
|     TypeULong     0x07
|     TypeFloat     0x08
|     TypeDouble    0x09
|     TypeString    0x0A
|     TypeObject    0x0B
|     TypeAny       0x0C
|     TypeSmallInt  0x0D
|     TypeInteger   0x0E
|     TypeDecimal   0x0F
|     TypeNumeric   0x10
|     TypeRaw       0x11
|     TypeLongRaw   0x12
|         TypeLongString 0x13
|         TypeWString  0x14
|         TypeDateTime 0x15
;

```

```

DataValue ::
    ValueNull[0],
|     ValueBoolean[1]
|     ValueChar[1]
|     ValueWchar[2]
|     ValueShort[2] // network order
|     ValueUShort[2] // network order
|     ValueLong[4] // network order
|     ValueULong[4] // network order
|     ValueFloat[4] // network order
|     ValueDouble[8] // network order
|     ValueString
|     ValueOctets
|     ValueWString
|     ValueOctet[1]
|         ValueDecimal
|         ValueAny
|         ValueObject
|         ValueDateTime
;

```

```

ValueDecimal:: ValueLong, ValueLong, ValueRaw
// precision, scale, value

```

```

ValueDateTime:: ValueShort, ValueOctet, ValueOctet, ValueOctet, ValueOctet, ValueOctet
//      year      , month      , day      , hour      , minute      , second

```

```

ValueString :: Length[4] //network order ,ValueChar<Length> ;

```

```

ValueWString :: Length[4] //network order ,ValueWChar<Length> ;
ValueOctets  :: Length[4] //network order ,ValueOctet<Length> ;
ValueAny     :: Length[4], TypeCode id as String, value as OctetSeq
ValueObject  :: Length[4], GIOP ObjectReference

```

15 Приложение 3 Грамматика SQL для коллекций:

```

%%

startTerm: query
          { $$ = $1; }
          ;

query:
  SELECT select_opt selection table_expr opt_order_by_clause opt_key_clause
      |
      was_error
      ;

select_opt:
  /* empty */
  |
  ALL
  |
  DISTINCT
  ;

selection:
select_scalar_expr_comma_list
|
'*,
;

table_expr:
  from_clause
  opt_where_clause
  opt_group_by_clause
  opt_having_clause
  ;

```

```

from_clause:
    FROM table_ref_commalist
    |
    FROM '(' query ')',
    ;

opt_where_clause:
    /* empty */
    |
    WHERE search_condition
    ;

opt_group_by_clause:
    /* empty */
    |
    GROUP BY column_ref_commalist
    ;

opt_having_clause:
    /* empty */
    |
    HAVING search_condition
    ;

opt_key_clause:
    /* empty */
    |
    WITH KEY selection
    ;

opt_order_by_clause:
    /* empty */
    |
    ORDER BY ordering_spec_commalist
    ;

search_condition:
    search_condition OR search_condition
    |
    search_condition AND search_condition
    |
    NOT search_condition
    |

```

```

        '(' search_condition ')'
        |
        predicate
        |
        '@'
        ;

predicate:
    comparison_predicate
    |
    between_predicate
    |
    like_predicate
    |
    test_for_null
    |
    in_predicate
    |
    all_or_any_predicate
    |
    existence_predicate
    ;

comparison_predicate:
    scalar_expr LESS subquery
    |
    scalar_expr LESS scalar_expr
    |
    scalar_expr LESS_EQ subquery
    |
    scalar_expr LESS_EQ scalar_expr
    |
    scalar_expr GT scalar_expr
    |
    scalar_expr GT subquery
    |
    scalar_expr GT_EQ scalar_expr
    |
    scalar_expr GT_EQ subquery
    |
    scalar_expr EQ subquery
    |
    scalar_expr EQ scalar_expr
    |
    scalar_expr NEQ subquery
    |

```

```

                scalar_expr NEQ scalar_expr
                ;

test_for_null:
scalar_expr IS NOT DBNULL
|
    scalar_expr IS DBNULL
;

in_predicate:
    scalar_expr NOT IN subquery
|
    scalar_expr IN subquery
|
    scalar_expr NOT IN '(' atom_commalist ')'
|
    scalar_expr IN '(' atom_commalist ')'
;

all_or_any_predicate:
    scalar_expr LESS any_all_some subquery
|
    scalar_expr LESS_EQ any_all_some subquery
|
    scalar_expr GT any_all_some subquery
|
    scalar_expr GT_EQ any_all_some subquery
|
    scalar_expr EQ any_all_some subquery
|
    scalar_expr NEQ any_all_some subquery
;

any_all_some:
    ANY
|
    ALL
|
    SOME
;

existence_predicate:
    EXISTS subquery
;

```

```

between_predicate:
    scalar_expr NOT BETWEEN scalar_expr AND scalar_expr
    |
    scalar_expr BETWEEN scalar_expr AND scalar_expr
    ;

like_predicate:
    scalar_expr NOT LIKE atom opt_escape
    |
    scalar_expr LIKE atom opt_escape
    ;

opt_escape:
    /* empty */
    { $$=NULL; }
    |
    ESCAPE atom
    ;

ordering_spec_commalist:
    ordering_spec
    |
    ordering_spec_commalist ',' ordering_spec
    ;

ordering_spec:
    INTNUM opt_ordering
    |
    column_ref opt_ordering
    ;

opt_ordering:
    /* empty */
    |
    ASC
    |
    DESC
    ;

subquery:
    '(' SELECT select_opt selection table_expr ')'
    ;

select_scalar_expr_comma_list:
    select_scalar_expr
    |

```

```

                select_scalar_expr_comma_list ',' select_scalar_expr
            ;

select_scalar_expr:
    '@'
    |
    scalar_expr
    ;

scalar_expr:
scalar_expr '+' scalar_expr
    |
    scalar_expr '-' scalar_expr
    |
    scalar_expr '*' scalar_expr
    |
    scalar_expr '/' scalar_expr
    |
    scalar_expr CONCAT_OP scalar_expr
    |
    '-' scalar_expr %prec UMINUS
    |
    '+' scalar_expr %prec UMINUS
    |
    atom
    |
    column_ref
    |
    function_ref
    |
    '(' scalar_expr ')'
    ;

table_ref_comma_list:
    table_ref
    |
    table_ref_comma_list ',' table_ref
    ;

atom_comma_list:
    atom
    |
    atom_comma_list ',' atom
    ;

```

```

atom:
parameter_ref
    |
    literal
    |
    { $$=yy_createConstStringTerm("USER"); }
    ;

column_ref_commalist:
    column_ref
    |
    column_ref_commalist ',' column_ref
    ;

column_ref:
    IDENT
    |
    IDENT '.' IDENT
    |
    IDENT '.' IDENT '.' IDENT
    ;

function_ref:
    compound_name '(' '*' ')'
    |
    compound_name '(' DISTINCT column_ref ')'
    |
    compound_name '(' ALL scalar_expr ')'
    |
    compound_name '(' scalar_expr ')'
    ;

compound_name:
    IDENT
    |
    IDENT '.' IDENT
    ;

table_ref:
    compound_name
    |
    compound_name IDENT
    |
    '@'
    ;

```

```
parameter_ref:
parameter
|
parameter parameter
|
parameter INDICATOR parameter
;
```

```
parameter:
',' IDENT
;
```

```
literal:
STRING
|
INTNUM
|
APPROXNUM
;
```

```
was_error:
error
;
```

16 Перечень изменений

- 24.01.2002 - грамматические правки, компиляция под Windows NT
- 21.01.2002 - описана работа с большими объектами (LOB)
- 18.01.2002 - добавлены некоторые пространства имен , приведено в соответствие с текущими idl.
- 19.06.2001 - добавлено описание инициализации.
- 06.06.2001 - отредактирован.
- 08.05.2001 - добавленна грамматика, просмотр.
- 16.04.2001 - добавлены приложения, первая публичная версия.
- 26.03.2000 - первая версию.

References

- [1] GradSoft. *UAKGQuery API reference*.
- [2] Object Management Group, editor. *The Common Object Request Broker: Architecture & Specification*. OMG, 1999. formal/99-10-07.
- [3] Object Management Group, editor. *Common Object Services Specification*, chapter 14. Query. OMG, 2000. formal/2000-06-23.
- [4] Object Management Group, editor. *Common Object Services Specification*, chapter Transaction Service. OMG, 2000. formal/2000-06-28.
- [5] Michi Henning and Steve Vinoski. *Advanced CORBA Programming with C++*. Addison-Wesley, 1999. ISBN 0201379279.
- [6] Anatoliy Doroshenko Ruslan Shevchenko. Techniques to uncreasing performance of corba parallel distributed applications. *Processing of PACT-2001*, 2001. in process.
- [7] Ruslan Shevchenko. Analysis of methods of creating effecient distributed applications, based on corba standart. *Processing of UKRPROG-2000*, 2000.
- [8] X/Open. *X/Open CAE specification - Distributed Transactions Processing*. ISBN 1-872630-24-3.