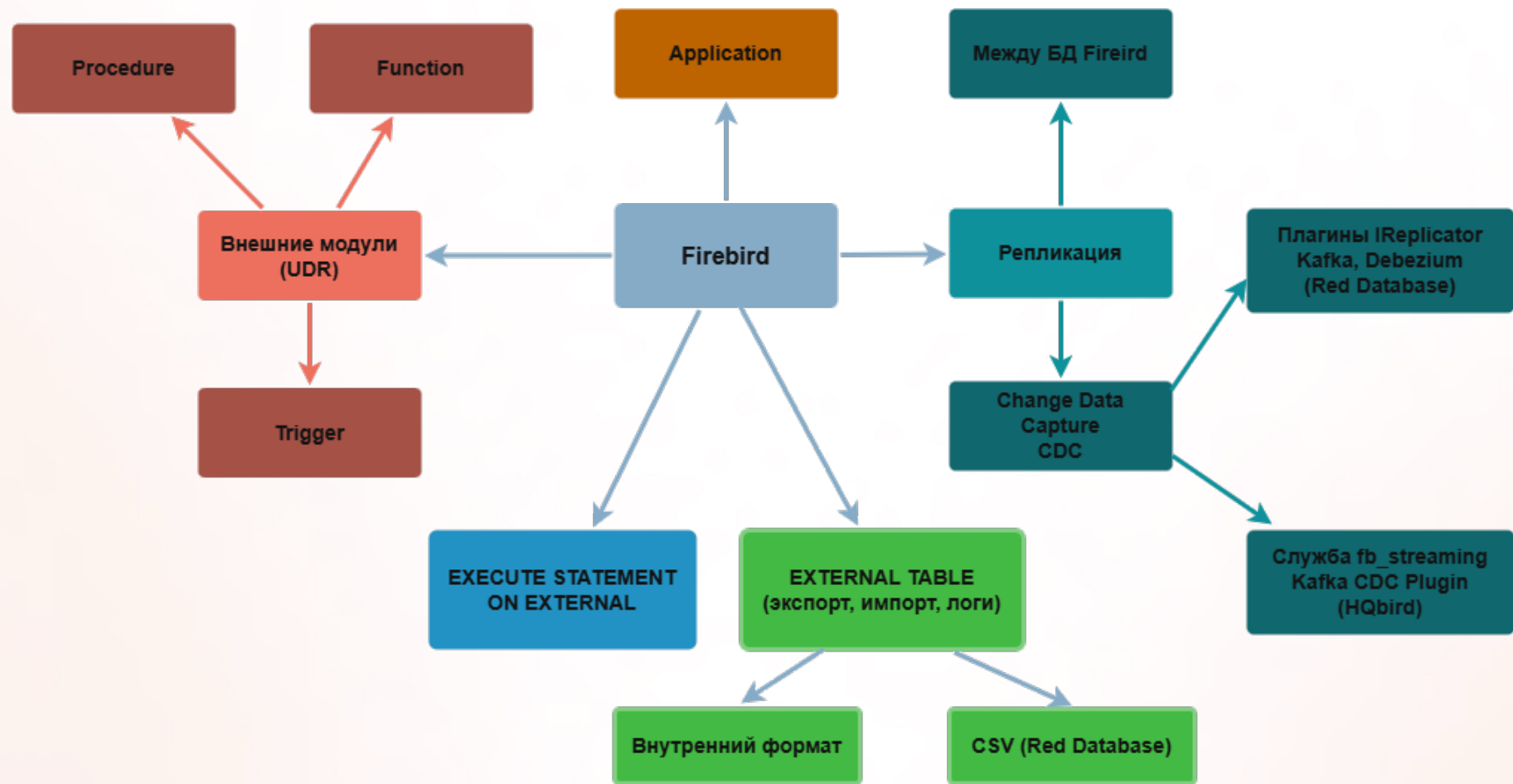


FIREBIRD 🐉 CONF 2025

Денис Симонов

Написание провайдеров для работы с внешними источниками данных

Firebird и внешние источники данных



EDS взаимодействие между БД Firebird или не только Firebird?

- HQbird 2024 взаимодействие с MySQL, ODBC

FOR

```
EXECUTE STATEMENT (:sql) [<params>]
```

```
ON EXTERNAL '<conn_string>'
```

```
AS USER 'root' PASSWORD '12345'
```

```
INTO ...
```

DO

Варианты реализации

- Расширить механизм EDS в ядре Firebird
- Расширить с помощью плагинов

Какой тип плагина нужно реализовать?

Устройство EDS

- Реализация EDS расположена в /src/jrd/extds/
- В EDS есть тоже провайдеры:
 - Firebird (EXECUTE STATEMENT для внешних БД Firebird)
 - Internal (EXECUTE STATEMENT для текущей БД)
- Можно расширять см. /src/jrd/extds/ExtDS.cpp

```
void Manager::addProvider(Provider* provider)
{
    // TODO: if\when usage of user providers will be implemented,
    // need to check provider name for allowed chars (file system rules ?)
    ...
}
```

Префиксы EDS провайдеров

```
SQL = 'SELECT MON$ATTACHMENT_NAME FROM MON$ATTACHMENTS';  
FOR EXECUTE STATEMENT :SQL  
    ON EXTERNAL 'Firebird::inet://localhost/ext_db'  
    AS USER 'SYSDBA' PASSWORD 'masterkey'  
    INTO NAME DO SUSPEND;  
-- вернёт ext_db  
  
ON EXTERNAL 'Internal::inet://localhost/ext_db'  
-- вернёт self_db (игнорируется строка коннекта)  
  
-- что если  
ON EXTERNAL 'ODBC::<conn_string>'
```

Проблемы EDS провайдеров

- Не реализованы как динамические библиотеки (плагины)
- Требуется модификация ядра Firebird
- Синхронизация кода с основной веткой Firebird

Особенности реализации EDS

- Провайдер Firebird:: работает через обычно API, которое используется в приложениях, то есть EDS выступает клиентом
- Что если API внешнего источника данных обернуть в API Firebird?

Плагин типа Provider

Краткие сведения в <doc/README.providers.html>

В `firebird.conf`

`Providers = Remote,Engine13,Loopback`

`Providers = Remote,Engine13,Engine12,Loopback`

`Providers = Remote,Engine13,ODBCEngine,Loopback`

`Providers = ODBCEngine,Remote,Engine13,Loopback`

`Providers = MySQLEngine,Remote,Engine13,Loopback`

Не путайте с провайдерами EDS!!!

Какие интерфейсы надо реализовать?

- IProvider
- IAttachment
- ITransaction
- IStatement
- IResultSet
- IBlob

Не все методы нужны. Реализуем только те, что могут использоваться в EDS. Для остальных делаем заглушки.

Регистрация фабрики IProvider

Посмотреть как реализовано для EngineXX в /src/jrd/jrd.cpp (.h)

```
void registerEngine(IPluginManager* iPlugin)
{
    UnloadDetectorHelper* module = getUnloadDetector();
    module->setCleanup(shutdownBeforeUnload);
    module->setThreadDetach(threadDetach);
    iPlugin->registerPluginFactory(IPluginManager::TYPE_PROVIDER, ODBC_ENGINE_NAME, &engineFactory);
    module->registerMe();
}

extern "C" FB_DLL_EXPORT void FB_PLUGIN_ENTRY_POINT(IMaster * master)
{
    CachedMasterInterface::set(master);
    registerEngine(PluginManagerInterfacePtr());
}
```

Интерфейс IProvider (ODBCProvider, MySQLProvider)

- IAttachment* attachDatabase(...) (обязателен)
- IAttachment* createDatabase(...) (бросаем isc_unavailable)
- IService* attachServiceManager(...) (для EDS не нужен)
- void shutdown(...) (не обязателен)
- void setDbCryptCallback (для EDS не нужен)

Необходимо сделать фабрику провайдера

IProvider::attachDatabase

- Анализировать строку соединения и выделить в ней префикс
- Если префикс совпадает с нашим префиксом, то создаём соединение с БД, в противном случае бросаем ошибку `isc_unavailable`.
- Префикс необходим, чтобы быстро определить надо ли делать попытку соединения, которая может быть не дешёвой, или пусть следующий провайдер пробует соединиться с БД.
- Предусмотрены следующие префиксы:
 - Для ODBC это: `':odbc:'` или `'odbc://'`
 - Для MySQL это: `':mysql:'` или `'mysql://'`

Интерфейс IAttachment (MySQLAttachment, ODBCAttachment)

- void getInfo(status, ...)
- ITransaction* startTransaction(status, ...)
- IBlob* createBlob(status, ...)
- IBlob* openBlob(status, ...)
- IStatement* prepare(status, ...)
- ITransaction* execute(status, ...)
- IResultSet* openCursor(status, ...) ???
- void detach(status)
- void dropDatabase(status) ???

IAttachment::getInfo

Обработать запрос `isc_info_db_sql_dialect` и `fb_info_features`, чтобы вернуть флаги поддерживаемой функциональности:

```
enum info_features // response to fb_info_features
{
    fb_feature_multi_statements = 1,    // Multiple prepared statements in single attachment
    fb_feature_multi_transactions= 2,   // Multiple concurrent transaction in single attachment
    fb_feature_named_parameters= 3,     // Query parameters can be named
    fb_feature_session_reset= 4,        // ALTER SESSION RESET is supported
    fb_feature_read_consistency= 5,     // Read consistency TIL is supported
    fb_feature_statement_timeout= 6,    // Statement timeout is supported
    fb_feature_statement_long_life  = 7, // Prepared statements are not dropped on transaction end
    fb_feature_max                  // Not really a feature. Keep this last.
};
```

IAttachment::startTransaction

- Вернуть экземпляр **ITransaction**, даже если транзакции не поддерживаются
- Инкрементировать внутренний номер транзакции
- Обработать Transaction Parameter Buffer (tpb)
 - Установить параметры транзакции в целевом драйвере (уровень изолированности, read/write mode, параметры ожидания разрешения блокировок ...)

Интерфейс ITransaction (MySQLTransaction, ODBCTransaction)

- void commit(status)
- void commitRetaining(status)
- void rollback(status)
- void rollbackRetaining(status)

Необходимо освободить внутренние ресурсы связанные с транзакцией, например BLOBs

IAttachment::prepare

- Создать экземпляр IStatement
- Определить тип запроса
 - `isc_info_sql_stmt_select`, `isc_info_sql_stmt_ddl`,
`isc_info_sql_stmt_exec_procedure`, `isc_info_sql_stmt_insert`
- Подготовить флаги запроса
 - `IStatement::FLAG_HAS_CURSOR` – если это курсор
 - `IStatement::FLAG_REPEAT_EXECUTE` – если есть входные параметры
- Подготовить входные и выходные сообщения

Соответствие типов данных

Firebird	MySQL	ODBC
VARCHAR(N), размер < 32765 байт	VARCHAR(N), BIT(N)	SQL_VARCHAR, SQL_WVARCHAR
CHAR(N), размер < 32767 байт	CHAR(N), BIT(N)	SQL_CHAR, SQL_WCHAR
VARBINARY(N), размер < 32765 байт	VARBINARY(N)	SQL_VARBINARY
BINARY(N), размер < 32767 байт	BINARY(N)	SQL_BINARY
SMALLINT (INTEGER беззнаковый)	TINYINT, SMALLINT, YEAR	SQL_TINYINT, SQL_SMALLINT
INTEGER (BIGINT беззнаковый)	MEDIUMINT, INTEGER	SQL_INTEGER
BIGINT (VARCHAR(20) беззнаковый)	BIGINT	SQL_BIGINT
FLOAT	FLOAT	SQL_REAL
DOUBLE PRECISION	DOUBLE	SQL_DOUBLE, SQL_FLOAT
DATE	DATE	SQL_TYPE_DATE
TIME	TIME	SQL_TYPE_TIME
TIMESTAMP	TIMESTAMP, DATETIME	SQL_TYPE_TIMESTAMP
VARCHAR(N), где N = precision + 2	DECIMAL	SQL_DECIMAL, SQL_NUMERIC
BLOB SUB_TYPE TEXT	TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT, JSON	SQL_LONGVARCHAR, SQL_WLONGVARCHAR
BLOB SUB_TYPE 0	TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB	SQL_LONGVARBINARY
BINARY(16)		SQL_GUID
BOOLEAN		SQL_BIT

Интерфейс IStatement (MySQLStatement, ODBCStatement)

- void getInfo(status, ...)
- void free(status)
- ISC_UINT64 getAffectedRecords(status)
- IMessageMetadata* getOutputMetadata(status)
- IMessageMetadata* getInputMetadata(status)
- unsigned getType(status)
- ITransaction* execute(status, ...)
- IResultSet* openCursor(status, ...)
- unsigned getFlags(status)
- unsigned int getTimeout(status)
- void setTimeout(status, timeout)

IStatement::execute

- Преобразовать входные параметры к нужным типам
- Скопировать входные параметры типа BLOB
- Выполнить SQL запрос
- Преобразовать выходные параметры к нужным типам
- Скопировать выходные параметры некоторых типов в BLOB

ISstatement::openCursor

- Преобразовать входные параметры к нужным типам
- Скопировать входные параметры типа BLOB
- Открыть курсор
- Создать экземпляр IResultSet

Интерфейс `ResultSet` (`MySQLResultSet`, `ODBCResultSet`)

- `int fetchNext(Status* status, void* message)`
- `IMessageMetadata* getMetadata(Status* status)`
- `void close(Status* status)`
- `void setDelayedOutputFormat(Status* status, IMessageMetadata* format)`

IResult::fetchNext

- Считать очередную запись из курсора
- Преобразовать выходные параметры к нужным типам
- Скопировать выходные некоторые параметры в BLOB

Интерфейс IBlob (MySQLBlob, ODBCBlob)

- int getSegment(Status* status, ...)
- void putSegment(Status* status, ...)
- void close(Status* status)
- void cancel(Status* status)
- void seek(Status* status, int mode, int offset)

Особенности работы с BLOB

- Создание нового BLOB (приходит из Firebird, приходит из вне)
IBlob* IAttachment::createBlob(status, ...)
- Открытие существующего
IBlob* IAttachment::openBlob(status, ...)
- Храниться на уровне IAttachment
- Идентификатор ISC_QUAD
 - gds_quad_high – номер транзакции
 - gds_quad_low – номер BLOB (инкремент)
- При подтверждении транзакции все созданные в ней BLOB очищаются

Оптимизация BLOB

- При передаче BLOB из нашего провайдера в EDS содержимое полностью копируется во временный BLOB (Firebird)
- EDS реализует только однонаправленные курсоры
- Это значит при `IRResultSet::fetchNext` можно стирать ранее переданные BLOB из памяти своего провайдера

Какие реализации Provider существуют?

- ODBCEngine, MySQLEngine (HQBird)
- jdbc_provider (Red Database)
- Magpie (ODBC Дмитрий Сибиряков)

Примеры MySQL

```
dsn_mysql = 'mysql://host=localhost;port=3306;database=employees';
```

```
for execute statement (q'{
  select
    emp_no, birth_date,
    first_name, last_name,
    gender, hire_date
  from employees
  where emp_no = ?
}') (10020)
on external dsn_mysql
as user 'root' password 'sa'
into
  emp_no, birth_date, first_name, last_name, gender, hire_date
do suspend;
```

Примеры MySQL

```
dsn_mysql = 'mysql://host=localhost;port=3306;database=employees';
```

```
for execute statement (q'{
  select
    emp_no, birth_date,
    first_name, last_name,
    gender, hire_date
  from employees
  where emp_no = :emp_no
}') (emp_no := 10020)
on external dsn_mysql
as user 'root' password 'sa'
into
  emp_no, birth_date, first_name, last_name, gender, hire_date
do suspend;
```

Примеры ODBC

```
conn_str = 'odbc://DRIVER={MariaDB ODBC 3.1  
Driver};SERVER=server;PORT=3306;DATABASE=test;CHARSET=utf8mb4';
```

```
sql = Q'{  
SELECT  
  id, title,  
  body, bydate  
FROM article  
}';  
for execute statement (:sql)  
  on external :conn_str  
  as user 'root' password 'play'  
  into id, title, body, bydate  
do suspend;
```

Примеры ODBC

```
xConnStr = 'odbc://DRIVER={MariaDB ODBC 3.1  
Driver};SERVER=127.0.0.1;PORT=3306;DATABASE=test;CHARSET=utf8mb4';
```

```
xSQL = '  
SELECT  
    CODE_SEX, NAME, NAME_EN  
FROM sex  
WHERE CODE_SEX = :A_CODE_SEX  
';  
for execute statement (:xSQL) (A_CODE_SEX := xCODE_SEX)  
    on external xConnStr  
    as user 'test' password '12345'  
    into CODE_SEX, NAME, NAME_EN  
do suspend;
```

Недостатки текущей реализации

- Префикс в строке соединения зависит от конфигурации
- Передача параметров аутентификации зависит от префикса
- Обработка ошибок соединения
- Не все СУБД умеют возвращать типы входных параметров
- Именованные параметры обрабатываются только для операторов SQL синтаксис которых похож на Firebird
- Не работает оператор CALL ХП возвращающими набор данных
- Необходимо следить за соответствием кодировки строковых параметров

Недостатки текущей реализации (префиксы)

- Провайдеры пробуют открыть соединение в порядке указанном в Provider (firebird.conf)
- Если попытка соединения неудачная пробуются следующий провайдер
- Префикс в строке соединения позволяет быстро определить подходит провайдер или нет
- Префиксы имеют вид ‘:odbc:’ или ‘odbc://’ и ‘:mysql:’ или ‘mysql://’
 - Двоеточие в начале префикса позволяет не интерпретировать его как имя хоста
 - Префикс вида <prefix>:// требует чтобы провайдер был до провайдера Remote Providers = ODBCEngine,Remote,Engine13,Loopback

Передача параметров аутентификации

- Провайдер должен быть расположен до провайдера Remote в списке Providers для использования префикса вида <prefix>://
- В противном случае имя пользователя приходится передавать в строке соединения, а не в параметре EDS USER. Вместо этого в USER передаём NULL. И используем префиксы вида :<prefix>:
- Причина: если передаётся имя пользователя и в строке коннекта не выделен хост, то EDS считает что это БД Firebird, которая находится на текущем сервере, а потому пытается применить к ней аутентификацию Firebird. То есть пользователь ищется в security.db (см. /src/jrd/extds/ValidatePassword.cpp)

Обработка ошибок соединения

- Если попытка соединения не удачна переходим к следующему провайдеру
- Реальная ошибка возникшая при попытке соединения теряется, возвращается ошибка из последнего провайдера в списке

Решение (костыль):

- Экземпляр псевдо-соединения `ErrorAttachment : IAttachment`
- Ошибка выбрасывается при вызове любого метода этого соединения

Не все СУБД умеют возвращать типы входных параметров

- MySQL API имеет функцию `mysql_stmt_param_count`
- Функция для описание параметров `mysql_stmt_param_metadata` – заглушка
- Костыль: использовать для параметра тип наибольшей вместимости - `VARCHAR(32764)`

Не все СУБД умеют возвращать типы входных параметров

- ODBC имеет функцию SQLDescribeParam
- Не все драйверы возвращают реальное описание. Например для myodbc для всех параметров возвращает VARCHAR(255)

Именованные параметры

```
dsn_mysql = 'mysql://host=localhost;port=3306;database=employees';  
execute statement  
( 'CALL sp_conn_audit(:A_CONN_ID, :A_USER, :A_DT)' )  
(  
    A_CONN_ID := current_connection,  
    A_USER := current_user,  
    A_DT := localtime  
)  
on external dsn_mysql  
as user 'root' password 'sa';
```

Statement failed, SQLSTATE = 42000

Execute statement error at isc_dsql_prepare :

335544382 : You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near ':A_CONN_ID, :A_USER, :A_DT)' at line 1

Statement : CALL sp_conn_audit(:A_CONN_ID, :A_USER, :A_DT)

Data source : Firebird::mysql:host=localhost;port=3306;database=employees

-At block line: 7, col: 3

Взгляд в будущее

- External data management
ISO/IEC 9075-9:2023(E) (SQL/MED) standard.
- Data foreign wrapper (провайдеры)
- Foreign server (внешние серверы)
- User Mapping (отображение пользователей)
- Access to foreign data (EXECUTE STATEMENT)
- Foreign table (внешние таблицы)

Foreign Server (внешние серверы)

```
CREATE SERVER [IF NOT EXISTS] <foreign server name>  
    [{FOREIGN DATA WRAPPER | USING PLUGIN} <provider (plugin)>]  
    [OPTIONS (<option> [,<option> ...] )]
```

```
<option> ::= {  
    CONNECTION_STRING = '<connection string>'  
    | USER = '<user name>'  
    | PASSWORD = '<password>'  
    | ROLE = '<role>'  
    | <option name> = 'value'  
    | <option name> 'value' }
```

Foreign Server (внешние серверы)

```
CREATE SERVER TEST_SERVER
FOREIGN DATA WRAPPER "Remote" OPTIONS (
    CONNECTION_STRING '172.17.0.2:/db/external.fdb',
    USER 'ext_user',
    PASSWORD 'ext_password');

FOR
EXECUTE STATEMENT q'{
select
    emp_no, birth_date, first_name,
    last_name, gender, hire_date
from employees }'
ON EXTERNAL SERVER TEST_SERVER
INTO
    emp_no, birth_date, first_name, last_name, gender, hire_date
```

Foreign Server (внешние серверы)

```
CREATE SERVER ODBC_SERVER
FOREIGN DATA WRAPPER "ODBCEngine" OPTIONS (
    CONNECTION_STRING 'odbc://DRIVER={MariaDB ODBC 3.1
Driver};SERVER=192.168.1.112;DATABASE=employees;CHARSET=utf8mb4',
    USER 'ext_user',
    PASSWORD 'ext_password');

FOR
EXECUTE STATEMENT q'{
select
    emp_no, birth_date, first_name,
    last_name, gender, hire_date
from employees }'
ON EXTERNAL SERVER ODBC_SERVER
INTO
    emp_no, birth_date, first_name, last_name, gender, hire_date
```

Foreign Server (переопределение учётных данных)

FOR

```
EXECUTE STATEMENT q'{  
select  
    emp_no, birth_date, first_name,  
    last_name, gender, hire_date  
from employees }'
```

ON EXTERNAL SERVER ODBC_SERVER

AS USER 'root' PASSWORD 'play'

INTO

```
emp_no, birth_date, first_name, last_name, gender, hire_date
```

User Mapping (отображение пользователей)

```
CREATE USER MAPPING FOR <user name> SERVER <server name>  
[OPTIONS (<option> [, <option> ...] )]
```

```
<option> ::= {  
    CONNECTION_STRING = '<connection string>'  
    | USER = '<user name>'  
    | PASSWORD = '<password>'  
    | ROLE = '<role>'  
    | <option name> = 'value'  
    | <option name> 'value' }
```

User Mapping

```
CREATE USER MAPPING FOR SYSDBA  
SERVER ODBC_SERVER  
OPTIONS (USER 'ext_user', PASSWORD 'ext_password');
```

```
CREATE USER MAPPING FOR ADMIN  
SERVER ODBC_SERVER  
OPTIONS (USER 'ext_user2', PASSWORD 'ext_password');
```

Foreign Table

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] <table name>
    (<foreign column definition> [, {<foreign column definition>
    | <table constraint>}...])
    SERVER <server name> [OPTIONS (<option> [, <option> ...] )]
```

```
<foreign column definition> ::=
    <regular column definition>
    | <identity column definition>
    [OPTIONS (<option> [, <option> ...] )]
```

```
<regular column definition> ::=
    <column name> { <data type> | <domain name>}
    [DEFAULT {<literal> | NULL | <context variable>}]
    [NOT NULL]
    [<column constraint>]
    [COLLATE <collation name>]
```

Foreign Table

```
<identity column definition> ::=  
    <column name> [<data type>]  
    GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY  
    [(<identity column option> [<identity column option>])]  
    [<column constraint>]
```

```
<identity column option> ::=  
    START WITH <start value>  
    | INCREMENT [BY] <increment value>
```

```
<column constraint> ::=  
    [CONSTRAINT <constraint name>] CHECK (<check condition>)
```

```
<table constraint> ::=  
    [CONSTRAINT <constraint name>] CHECK (<check condition>)
```

```
<option> ::= {  
    <option name> [= 'value']  
    | <option name> ['value'] }
```

Foreign Table

```
RECREATE FOREIGN TABLE ODBC_EMPLOYEES(  
    EMP_NO BIGINT GENERATED ALWAYS AS IDENTITY OPTIONS(COLUMN_NAME 'emp_no'),  
    BIRTH_DATE DATE OPTIONS(COLUMN_NAME 'birth_date'),  
    FIRST_NAME VARCHAR(14) OPTIONS(COLUMN_NAME 'first_name'),  
    LAST_NAME VARCHAR(14) OPTIONS(COLUMN_NAME 'last_name'),  
    GENDER CHAR(1) OPTIONS(COLUMN_NAME 'gender'),  
    HIRE_DATE DATE OPTIONS(COLUMN_NAME 'hire_date')  
)  
SERVER ODBC_SERVER OPTIONS(TABLE_NAME 'employees');
```

```
SELECT *  
FROM ODBC_EMPLOYEES  
WHERE EMP_NO = 10001; -- Локальный предикат пробрасывается
```

Foreign Table

```
UPDATE ODBC_EMPLOYEES  
SET LAST_NAME = 'Dow'  
WHERE EMP_NO = 40245;
```

```
INSERT INTO ODBC_EMPLOYEES (  
    BIRTH_DATE, FIRST_NAME, LAST_NAME, GENDER  
)  
VALUES (date '1991-03-11', 'Marie', 'Dow', 'F');
```

```
DELETE FROM ODBC_EMPLOYEES WHERE EMP_NO = 40245;
```

Foreign Table

```
SELECT COUNT(*)  
FROM  
    ODBC_EMPLOYEES  
    JOIN MYSQL_EMPLOYEES ON ODBC_EMPLOYEES.EMP_NO = MYSQL_EMPLOYEES.EMP_NO  
WHERE ODBC_EMPLOYEES.EMP_NO BETWEEN 10000 AND 10010;
```

Select Expression

- > Aggregate

- > Filter

- > Hash Join (inner) (keys: 1, total key length: 8)

- > Table "ODBC_EMPLOYEES" Full Scan

- > Record Buffer (record length: 33)

- > Table "MYSQL_EMPLOYEES" Full Scan

Взгляд в будущее

- EXECUTE STATEMENT ON EXTERNAL не надо постоянно копировать строку соединения/учётные данные
- ESOES для сторонних провайдеров используется точно также как и для встроенных провайдеры
- Не нужно менять конфигурацию firebird.conf
- Отображение пользователей позволяет иметь разные учётные данные в зависимости от текущего пользователя
- Foreign Table позволяют выполнять гетерогенные запросы, и работать с внешними данными в более привычном виде, то есть без использования динамических запросов

Вопросы

29 МАЯ | МОСКВА

FIREBIRD CONF 2025