New SQL Features in Firebird

Remember Firebird 2.1

- **Common SQL**
  - COMMON TABLE EXPRESSIONS
  - INSERT OR UPDATE
  - MERGE
  - RETURNING
  - Built-in functions
- **Procedural SQL**
  - Domains at procedures and triggers
    - *TYPE OF <domain name>*
- **DDL**
  - DATABASE TRIGGER's
    - *CONNECT | DISCONNECT*
    - *TRANSACTION START | COMMIT | ROLLBACK*
  - GLOBAL TEMPORARY TABLE
- **Monitoring**
  - Monitoring tables
  - Ability to break execution of user query

Whats New in Firebird 2.5

- **Common SQL**
  - SIMILAR TO
  - SQLSTATE
  - Hexadecimal constants
  - UUID binary to\from char conversions
- **Procedural SQL**
  - AUTONOMOUS TRANSACTIONS
  - EXECUTE STATEMENT
  - TYPE OF COLUMN
- **DDL**
  - ALTER VIEW
  - ALTER computed fields
  - CREATE\ALTER\DROP user
  - ALTER ROLE
  - GRANTED BY
- **Monitoring**
  - New MONITORING TABLES
  - Terminate user connection

Common SQL : SIMILAR TO

Regular expressions support per SQL standard specification

New SIMILAR TO predicate

More powerful version of LIKE with regexp syntax

Example : is given string represents a valid number ?

```
Value SIMILAR TO '[\+\-]?[0-9]*([0-9].|.[0-9])?[[:DIGIT:]]*'
           ESCAPE '\'

[\+\-]                  + or -
?                       0 or 1 times
[0-9]                   any digit
*                       0 or more times
([0-9].|.[0-9])         <digit and point> or <point and digit>
?                       0 or 1 times
[[:DIGIT:]]             any digit
*                       0 or more times
```


Common SQL : SQLSTATE

SQLSTATE is standard compliant generic error code for use by generic applications

SQLCODE is deprecated (but still supported) and it is recommended to use SQLSTATE

To obtain SQLSTATE value use new API function : fb_sqlstate

SQLSTATE is not available (yet) for WHEN block of PSQL exception handling

isql since v2.5 used SQLSTATE in error messages :

```
FB25>isql
SQL> connect 'not_exists';
Statement failed, SQLSTATE = 08001
I/O error during "CreateFile (open)" operation for file "not_exists"
-Error while trying to open file
-The system cannot find the file specified.

FB21>isql
SQL> connect 'not_exists';
Statement failed, SQLCODE = -902
I/O error for file "...\not_exists"
-Error while trying to open file
-The system cannot find the file specified.
```


Common SQL : HEX Literals

Hexadecimal numeric and binary string literals

Numeric literals : *0xHHHHHHHH*

Prefix *0x* or *0X*

Up to 16 hexadecimal digits

1 - 8 digits : data type is - signed integer

9 - 16 digits : data type is - signed bigint

```
SQL> SELECT 0xF0000000, 0x0F0000000 FROM RDB$DATABASE;

    CONSTANT                 CONSTANT
============ ====================
  -268435456           4026531840
```


Common SQL : HEX literals

Hexadecimal numeric and binary string literals

String literals : *x'HH...H'*

Prefix *x* or *X*

Data type - CHAR(N / 2) CHARACTER SET OCTETS, where N – number of digits

```
SQL> SELECT 'First line' || _ASCII x'0D0A09' || 'Second line'
CON> FROM RDB$DATABASE;

CONCATENATION
=======================
First line
        Second line
```


Common SQL : UUID <-> CHAR

UUID binary to\from CHAR conversion

CHAR_TO_UUID

Converts the CHAR(32) ASCII representation of an UUID
(*XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX*)
to the binary (CHAR(16) OCTETS) representation (optimized for storage)

```
SQL> SELECT CHAR_TO_UUID('A96B285B-4629-45A1-9A86-A8ECCF6561F4')
        FROM RDB$DATABASE;

CHAR_TO_UUID
================================
A96B285B462945A19A86A8ECCF6561F4
```


Common SQL : UUID <-> CHAR

UUID binary to\from CHAR conversion

UUID_TO_CHAR

Converts a binary (CHAR(16) OCTETS) UUID to the string (CHAR(32) ASCII) representation

```
SQL> SELECT UUID_TO_CHAR(x'A96B285B462945A19A86A8ECCF6561F4')
        FROM RDB$DATABASE;

UUID_TO_CHAR
====================================
A96B285B-4629-45A1-9A86-A8ECCF6561F4
```


PSQL : Autonomous Transactions

Syntax

```
IN AUTONOMOUS TRANSACTION DO
    <simple statement | compound statement>
```

Parameters ?

Same as outer transaction (isolation level, read\write, wait mode, etc)
Configurable ? Not yet... may be later

How it ends ?

```
if (statement executed ok)
then commit
else rollback
```

Notes

Autonomous transaction and its outer transaction fully independent and isolated from each other as any other two transactions.

Temporary BLOBs, created in autonomous transaction, attached to outer transaction. This is done to allow usage of such blobs after autonomous transaction ends. This behavior may be a source of unexpected additional memory consumption.

PSQL : EXECUTE STATEMENT

- **New implementation of EXECUTE STATEMENT :**
  - **Input parameters**
  - **May run with privileges of caller PSQL object**
  - **Autonomous transactions**
  - **Query another Firebird database**
  - **Full backward compatibility**

- **Syntax**

```
[FOR] EXECUTE STATEMENT <query_text> [(<input_parameters>)]
  [ON EXTERNAL [DATA SOURCE] <connection_string>]
  [WITH AUTONOMOUS | COMMON TRANSACTION]
  [AS USER <user_name>]
  [PASSWORD <password>]
  [ROLE <role_name>]
  [WITH CALLER PRIVILEGES]
  [INTO <variables>]
```


PSQL : EXECUTE STATEMENT

Input parameters

Named input parameters

```
EXECUTE STATEMENT
    ('INSERT INTO TABLE VALUES (:a, :b, :a)')
    (a := 100, b := CURRENT_CONNECTION)
```

Not named input parameters

```
EXECUTE STATEMENT
    ('INSERT INTO TABLE VALUES (?, ?, ?)')
    (100, CURRENT_CONNECTION, 100)
```


PSQL : EXECUTE STATEMENT

Caller privileges

```
-- logon as SYSDBA
CREATE TABLE A (ID INT);
CREATE USER VLAD PASSWORD 'vlad';

CREATE PROCEDURE P1 RETURNS (CNT INT)
AS
BEGIN
  EXECUTE STATEMENT 'SELECT COUNT(*) FROM A' INTO :CNT;
  SUSPEND;
END;

GRANT SELECT ON TABLE A TO PROCEDURE P1;
GRANT EXECUTE ON PROCEDURE P1 TO USER VLAD;

-- logon as VLAD
SELECT * FROM P1;
```

```
Statement failed, SQLSTATE = 42000
Execute statement error at jrd8_prepare :
335544352 : no permission for read/select access to TABLE A
Statement : SELECT COUNT(*) FROM A
Data source : Internal::
-At procedure 'P1'
```


PSQL : EXECUTE STATEMENT

Caller privileges

```
-- logon as SYSDBA
CREATE PROCEDURE P2 RETURNS (CNT INT)
AS
BEGIN
  EXECUTE STATEMENT 'SELECT COUNT(*) FROM A'
    WITH CALLER PRIVILEGES
    INTO :CNT;
  SUSPEND;
END;

GRANT SELECT ON TABLE A TO PROCEDURE P2;
GRANT EXECUTE ON PROCEDURE P2 TO USER VLAD;

-- logon as VLAD
SELECT * FROM P2;

          CNT
============
            0
```

Dynamic statement is executed with that set of privileges as it would have if its executed immediately by caller PSQL object (procedure or trigger)

PSQL : EXECUTE STATEMENT

Transactions

Common transaction (default)

statement executed in the current transaction

```
EXECUTE STATEMENT '...'
    WITH COMMON TRANSACTION
```

Autonomous transaction

statement executed in the separate new transaction

```
EXECUTE STATEMENT '...'
    WITH AUTONOMOUS TRANSACTION
```

parameters the same as current trancation, not (yet) configurable

PSQL : EXECUTE STATEMENT

Query another Firebird database

EXTERNAL DATA SOURCE clause :

<connection_string> is usual Firebird's connection string

Query another database using user\password

```
EXECUTE STATEMENT '...'
    ON EXTERNAL DATA SOURCE 'host:path'
    USER 'VLAD' PASSWORD 'dontKnow'
```

When user\password is not set

Trusted autentication (Windows only) : Firebird's process account name is effective user name at remote database (if TA is supported by remote side)

```
EXECUTE STATEMENT '...'
    ON EXTERNAL DATA SOURCE 'host:path'
```

CURRENT_USER is effective user name at local database

```
EXECUTE STATEMENT '...'
    ON EXTERNAL DATA SOURCE 'path_to_the_current_database'
```


PSQL : TYPE OF COLUMN

Syntax

```
data_type ::= <builtin_data_type>
      |  <domain_name>
      |  TYPE OF <domain_name>
      |  TYPE OF COLUMN <table_name>.<column_name>
```

Usage

- Input and output parameters

```
CREATE PROCEDURE
  MY_PROC (IN_PARAM  TYPE OF COLUMN <table_name>.<column_name>)
  RETURNS (OUT_PARAM TYPE OF COLUMN <table_name>.<column_name>)
```

- Variable declaration

```
DECLARE VARIABLE VAR1 TYPE OF COLUMN <table_name>.<column_name>
```

- CAST statement

```
OUT_PARAM = CAST(VAR1 AS TYPE OF COLUMN <table_name>.<column_name>)
```


ALTER VIEW \ COMPUTED FIELD

Syntax

```
{CREATE OR ALTER | ALTER} VIEW <view_name> [(<field list>)]
    AS <select statement>

ALTER TABLE <table_name>
  ALTER <field_name> [TYPE <data type>] COMPUTED BY (<expression>)
```

Task

- change view (or computed field) definition

- **Before Firebird 2.5**
  - drop and create (or alter two times) again all dependent objects
  - restore all privileges granted to all dependent objects which was re-created
- **Firebird 2.5**
  - just execute **ALTER VIEW** and enjoy :-)

ALTER VIEW \ COMPUTED FIELD

Example

```
CREATE TABLE T (NAME CHAR(20), CF COMPUTED BY (LEFT(NAME, 10)) );
CREATE VIEW V AS SELECT CF FROM T;
CREATE PROCEDURE P ... SELECT ... FROM V ...; -- dependent procedure
```

Try to change both view and computed field definitions

Before Firebird 2.5 :

```
ALTER PROCEDURE P AS BEGIN END;  -- empty body for ALL dependent objects
DROP VIEW V;
ALTER TABLE T DROP CF;
ALTER TABLE T
    ADD CF COMPUTED BY (SUBSTRING(NAME FOR 15));
CREATE VIEW V AS
    SELECT NAME, CF FROM T;
GRANT SELECT ON TABLE T TO VIEW V;            -- restore ALL dependent
ALTER PROCEDURE P ... SELECT ... FROM V ...; -- objects and privileges
```

What if you have hundreds dependent objects ?

ALTER VIEW \ COMPUTED FIELD

Example

```
CREATE TABLE T (NAME CHAR(20), CF COMPUTED BY (LEFT(NAME, 10)) );
CREATE VIEW V AS SELECT CF FROM T;
CREATE PROCEDURE P ... SELECT ... FROM V ...; -- dependent procedure
```

Try to change both view and computed field definitions

Firebird 2.5 :

```
ALTER TABLE T
    ALTER CF COMPUTED BY (RIGHT(NAME, 10));

ALTER VIEW V AS
    SELECT NAME, CF FROM T;
```


DDL : USER MANAGEMENT

Syntax

```
CREATE USER name PASSWORD 'password'
    [FIRSTNAME 'firstname']
    [MIDDLENAME 'middlename']
    [LASTNAME 'lastname']

ALTER USER name PASSWORD 'password'
    [FIRSTNAME 'firstname']
    [MIDDLENAME 'middlename']
    [LASTNAME 'lastname']

DROP USER name
```


DDL : SYSDBA and SYSADMIN's

When Windows Administrator is connected using trusted authentication :

Firebird 2.1

CURRENT_USER is a SYSDBA
SYSDBA privileges

Firebird 2.5

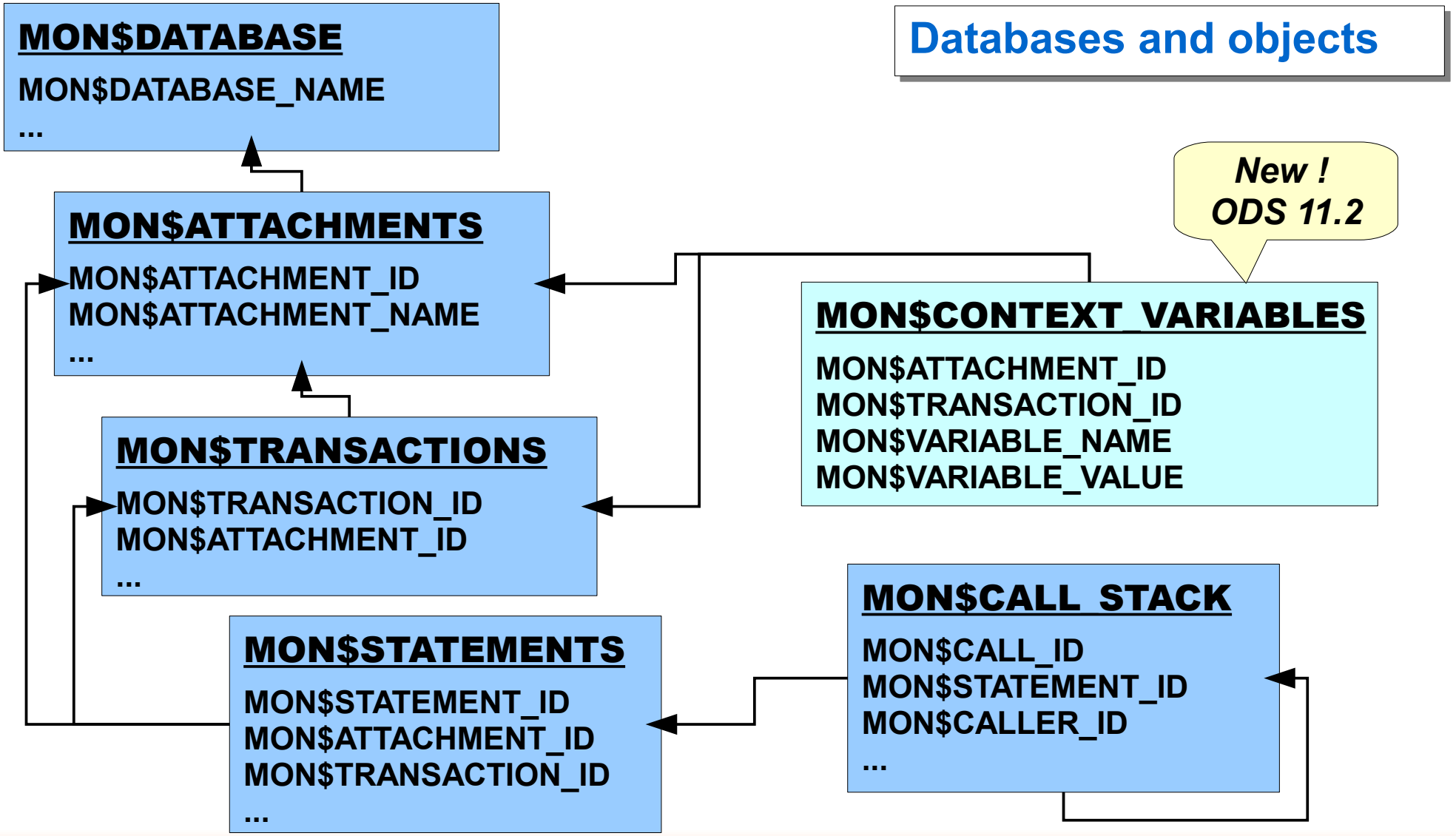
CURRENT_USER is always a Domain\User

Auto-admin mapping (per database)

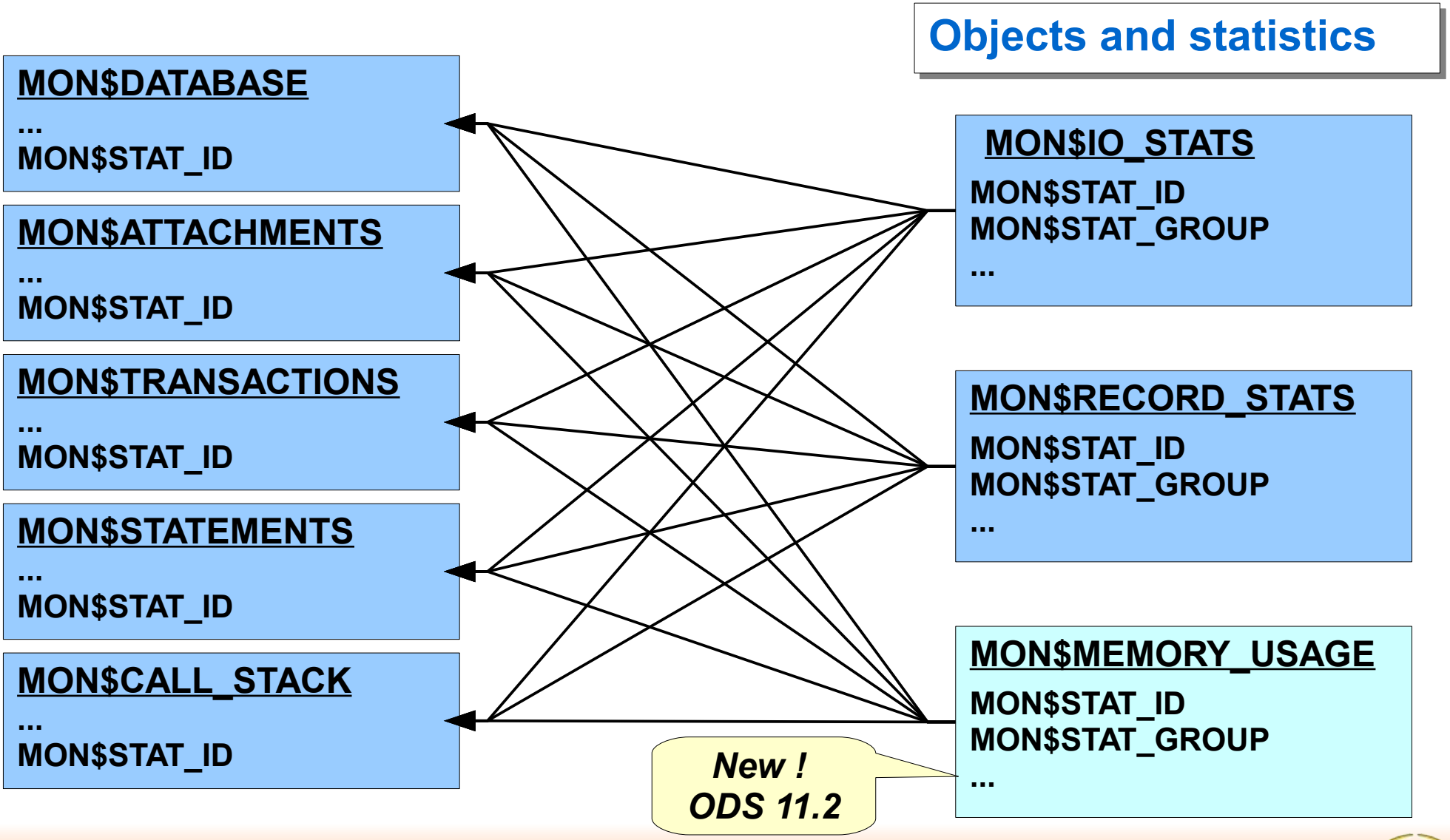
```
ALTER ROLE RDB$ADMIN SET|DROP AUTO ADMIN MAPPING;
```

Auto-admin mapping	ROLE	Privileges
ON	*NONE or RDB$ADMIN*	*SYSDBA*
OFF	*NONE or RDB$ADMIN*	*ordinary*
ON	*Other*	
OFF		

MONITORING TABLES



MONITORING TABLES



MONITORING TABLES

Some monitoring tables is “active”

1) Cancel current activity of connection № 32:

```
DELETE FROM MON$STATEMENTS
 WHERE MON$ATTACHMENT_ID = 32
```

2) Cancel all queries running more than 20 minutes:

```
DELETE FROM MON$STATEMENTS
 WHERE DATEADD(20 MINUTE TO MON$TIMESTAMP) < CURRENT_TIMESTAMP
   AND MON$STATE <> 0;
```

3) Disconnect everybody but ourselves (new in Firebird 2.5):

```
DELETE FROM MON$ATTACHMENTS
 WHERE MON$ATTACHMENT_ID <> CURRENT_CONNECTION
```


What's next ?

Whats new in Firebird 3

- **Common SQL**
  - Full syntax of **MERGE** (per SQL 2008)
    - support for **DELETE** substatement
    - additional condition in **WHEN** clause
    - multiply **WHEN** clauses
  - **MERGE ... RETURNING**
  - Window (analytical) functions
  - **SUBSTRING** with regular expressions
  - **BOOLEAN** data type
  - Cursor stability with data modification queries
  - Global temporary tables are improved

Whats new in Firebird 3

- **Procedural SQL**
  - **SQL functions**
  - **Sub-routines**
  - **External functions, procedures and triggers on C\C++\Pascal\Java etc.**
  - **Packages**
  - **Exceptions with parameters : EXCEPTION ... USING (...)**
  - **SQLSTATE in WHEN handler**
  - **CONTINUE in loops**

Whats new in Firebird 3

- **DDL**
  - **Manage nullability of column**
    - **ALTER DOMAIN ... {NULL | NOT NULL}**
    - **ALTER COLUMN ... {NULL | NOT NULL}**
  - **ALTER DATABASE ... SET DEFAULT CHARACTER SET**
  - **IDENTITY columns**

    (hello, migrants from MSSQL\MySQL ;)
  - **RECREATE SEQUENCE, RECREATE GENERATOR**
  - **DDL triggers**

Whats new in Firebird 3

- **More complete system of SQL grants**
  - **GRANT CREATE | ALTER | DROP <object> TO <user> | <role>**
  - **GRANT ROLE TO ROLE**
- **Monitoring**
  - **Extended statistics, queries plan's, ...**
- **To be continued :-)**

Common SQL : MERGE

Full SQL 2008 syntax

```
MERGE INTO <table>
   USING <table_or_join>
      ON <search_condition>

   [WHEN MATCHED [AND <search_condition>] THEN
      UPDATE SET col1 = val1, ..., colN = valN
     |
      DELETE]

   [WHEN NOT MATCHED [AND <search_condition>] THEN
      INSERT [(col1, ..., colN)] VALUES (val1, ..., valN)]
```


Common SQL : MERGE

DELETE substatement and multiply WHEN clauses

```
MERGE INTO TABLE
   USING LOG
      ON TABLE.PK = LOG.PK

   WHEN MATCHED AND LOG.ACTION = 'D' THEN
      DELETE

   WHEN MATCHED THEN                    -- second WHEN MATCHED clause
      UPDATE SET col1 = LOG.val1, ...

   WHEN NOT MATCHED THEN
      INSERT (col1, ...) VALUES (LOG.val1, ...)
```


Common SQL : MERGE

RETURNING clause

```
MERGE INTO <table>
   USING <table_or_join>
      ON <search_condition>

   [WHEN MATCHED [AND <search_condition>] THEN
      UPDATE SET col1 = val1, ..., colN = valN
     |
      DELETE]

   [WHEN NOT MATCHED [AND <search_condition>] THEN
      INSERT [(col1, ..., colN)] VALUES (val1, ..., valN)]

   [RETURNING ... [INTO ...]]
```


Common SQL : WINDOW FUNCTIONS

Syntax

```
<window function> ::=
    <window function type> OVER (<window specification>)

<window function type> ::=
     <aggregate function>                -- aggregate
   | <rank function type>                -- ranking
   | ROW_NUMBER                          -- yes, it is row number ;)
   | <lead or lag function>              -- navigational
   | <first or last value function>
   | <nth value function>

<window specification> ::=
    [PARTITION BY column1, ...]
    [ORDER BY column1 [ASC|DESC] [NULLS {FIRST|LAST}], ... ]
```


Common SQL : WINDOW FUNCTIONS

Syntax

```
<aggregate function> ::=
    AVG | MAX | MIN | SUM | COUNT | LIST

<rank function type> ::=
    RANK | DENSE_RANK

<lead or lag function> ::=
    LEAD | LAG

<first or last value function> ::=
    {FIRST_VALUE | LAST_VALUE} (<value>)

<nth value function> ::=
    NTH_VALUE (<value>, <nth row>)
```


Common SQL : WINDOW FUNCTIONS

Example

```
SELECT A, B, C,
   SUM(C) OVER(),
   SUM(C) OVER(ORDER BY A, B),
   SUM(C) OVER(PARTITION BY A),
   SUM(C) OVER(PARTITION BY A ORDER BY B)
```

A	B	C	SUM	SUM1	SUM2	SUM3
1	1	30	141	30	60	30
1	2	20	141	50	60	50
1	3	10	141	60	60	60
2	1	25	141	85	40	25
2	2	15	141	100	40	40
3	1	41	141	141	41	41

Common SQL : substring search using REGEXP

Syntax

```
SUBSTRING(<string> SIMILAR <pattern> ESCAPE <char>)
```

Rules

- **Pattern**
  - `R = <R1> <E> " <R2> <E> " <R3>`
- **Search**
  - `S = <S1> <S2> <S3>`

```
1)<S>       SIMILAR TO <R1> <R2> <R3> ESCAPE <E>

2)<S1>      SIMILAR TO <R1>          ESCAPE <E> AND
  <S2> <S3> SIMILAR TO <R2> <R3>     ESCAPE <E>

3)<S2>      SIMILAR TO <R2>          ESCAPE <E> AND
  <S3>      SIMILAR TO <R3>          ESCAPE <E>
```

- **Result**
  - S2

Common SQL : substring search using REGEXP

Syntax

```
SUBSTRING(<string> SIMILAR <pattern> ESCAPE <char>)
```

Example

```
SUBSTRING('abc-12b34xyz' SIMILAR '%\"[\+\-]?[0-9]+\"%' ESCAPE '\')

R1 = %
R2 = [\+\-]?[0-9]+
R3 = %

1)  'abc-12b34xyz' SIMILAR TO '%[\+\-]?[0-9]+%' ESCAPE '\'

2)  'abc'          SIMILAR TO '%'                ESCAPE '\' AND
    '-12b34xyz'    SIMILAR TO '[\+\-]?[0-9]+%'   ESCAPE '\'

3)  '-12'          SIMILAR TO '[\+\-]?[0-9]+'    ESCAPE '\' AND
    'b34xyz'       SIMILAR TO '%'                ESCAPE '\'
```

Result

```
'-12'
```


Common SQL : BOOLEAN data type

Syntax

```
<data_type> ::= BOOLEAN
<boolean_literal> ::= TRUE | FALSE | UNKNOWN
```

Storage

```
1 byte
```

Client support (XSQLDA)

```
#define SQL_BOOLEAN 32764
```


Common SQL : BOOLEAN data type

Truth tables

AND	True	False	Unknown
True	*True*	*False*	*Unknown*
False	*False*	*False*	*False*
Unknown	*Unknown*	*False*	*Unknown*

OR	True	False	Unknown
True	*True*	*True*	*True*
False	*True*	*False*	*Unknown*
Unknown	*True*	*Unknown*	*Unknown*

IS	True	False	Unknown
True	*True*	*False*	*False*
False	*False*	*True*	*False*
Unknown	*False*	*False*	*True*

NOT	
True	*False*
False	*True*
Unknown	*Unknown*

Common SQL : BOOLEAN data type

Examples

```
CREATE TABLE TBOOL (ID INT, BVAL BOOLEAN);
COMMIT;

INSERT INTO TBOOL VALUES (1, TRUE);
INSERT INTO TBOOL VALUES (2, 2 = 4);
INSERT INTO TBOOL VALUES (3, NULL = 1);
COMMIT;
```

```
SELECT * FROM TBOOL

            ID     BVAL
============= =======
             1 <true>
             2 <false>
             3 <null>
```


Common SQL : BOOLEAN data type

Examples

1. Test for TRUE value

```
SELECT * FROM TBOOL
 WHERE BVAL

            ID     BVAL
============ =======
             1 <true>
```

2. Test for FALSE value

```
SELECT * FROM TBOOL
 WHERE BVAL IS FALSE

            ID     BVAL
============ =======
             2 <false>
```

3. Tests for UNKNOWN value

```
SELECT * FROM TBOOL
 WHERE BVAL IS UNKNOWN

            ID     BVAL
============ =======
             3 <null>


SELECT * FROM TBOOL
 WHERE BVAL = UNKNOWN


SELECT * FROM TBOOL
 WHERE BVAL <> UNKNOWN
```


Common SQL : BOOLEAN data type

Examples

4. Boolean values in SELECT list

```
SELECT ID, BVAL, BVAL AND ID < 2
  FROM TBOOL

           ID     BVAL
============ ======= =======
           1 <true>  <true>
           2 <false> <false>
           3 <null>  <false>
```


Common SQL : cursor stability

The issue

- Famous infinite insertion circle (CORE-92)

```
INSERT INTO T
SELECT * FROM T
```

- DELETE more rows than expected (CORE-634)

```
DELETE FROM T
 WHERE ID IN (SELECT FIRST 1 ID FROM T)
```

- All DML statements is affected (INSERT, UPDATE, DELETE, MERGE)
- Common ticket at tracker CORE-3362

Common SQL : cursor stability

The reason of the issue

- DML statements used implicit cursors
  - `INSERT INTO T SELECT ... FROM T`
    works as

```
FOR SELECT <values> FROM T INTO <tmp_vars>
  DO INSERT INTO T VALUES (<tmp_vars>)
```

  - `UPDATE T SET <fields> = <values> WHERE <conditions>`
    works as

```
FOR SELECT <values> FROM T WHERE <conditions> INTO <tmp_vars>
    AS CURSOR <cursor>
DO UPDATE T SET <fields> = <tmp_vars>
       WHERE CURRENT OF <cursor>
```

  - `DELETE` works like `UPDATE`

Common SQL : cursor stability

The “standard” way

- Rows to be inserted\updated\deleted should be marked first
- Marked rows is inserted\updated\deleted then
- Pros
  - rowset is stable and is not affected by DML statement itself
- Cons
  - Marks should be saved somewhere and rows will be visited again, or
  - Set of marked rows should be saved somewhere and this store will be visited again
- Note : this could be reached in Firebird using (well known) workaround: ***force query to have SORT in PLAN - it will materialize implicit cursor and make it stable***

Common SQL : cursor stability

The Firebird 3 way

- Use undo-log to see if record was already modified by current cursor
  - if record was inserted - ignore it
  - if record was updated or deleted - read backversion
- Pros
  - No additional bookkeeping required
  - No additional storage required
  - Relatively easy to implement
- Cons
  - Inserted records could be visited (but ignored, of course)
  - Backversions of updated\deleted records should be read
  - Not works with SUSPEND in PSQL

Common SQL : cursor stability

PSQL notes

- **PSQL cursors with SUSPEND inside still not stable !**

  This query still produced infinite circle

```
FOR SELECT ID FROM T INTO :ID
  DO BEGIN
    INSERT INTO T (ID) VALUES (:ID);
    SUSPEND;
  END
```


Common SQL : cursor stability

PSQL notes

- **PSQL cursors without SUSPEND inside is stable now, this could change old behavior**

```
FOR SELECT ID FROM T WHERE VAL IS NULL INTO :ID
  DO BEGIN
    UPDATE T SET VAL = 1
     WHERE ID = :ID;
  END
```


Common SQL : improvements in GTT

- Global temporary tables is writable even in read-only transactions
  - Read-only transaction in read-write database
    - Both ON COMMIT PRESERVE ROWS and ON COMMIT DELETE ROWS
  - Read-only transaction in read-only database
    - GTT ON COMMIT DELETE ROWS only
- Faster rollback for GTT ON COMMIT DELETE ROWS
  - No need to backout records on rollback
- Garbage collection in GTT is not delayed by active transactions of another connections
- All this improvements is backported into v2.5.1 too

PSQL : SQL functions

Syntax

```
{CREATE [OR ALTER] | ALTER | RECREATE} FUNCTION <name>
    [(param1 [, ...])]
    RETURNS <type>
AS
BEGIN
    ...
END
```

Example

```
CREATE FUNCTION F(X INT) RETURNS INT
AS
BEGIN
  RETURN X+1;
END;

SELECT F(5) FROM RDB$DATABASE;
```


PSQL : SQL sub-routines

Syntax

- **Sub-procedures**

```
DECLARE PROCEDURE <name> [(param1 [, ...])]
    [RETURNS (param1 [, ...])]
AS
...
```

- **Sub-functions**

```
DECLARE FUNCTION <name> [(param1 [, ...])]
    RETURNS <type>
AS
...
```


PSQL : SQL sub-routines

Example

```
EXECUTE BLOCK RETURNS (N INT)
AS
  DECLARE FUNCTION F(X INT) RETURNS INT
  AS
  BEGIN
    RETURN X+1;
  END;
BEGIN
  N = F(5);
  SUSPEND;
END
```


PSQL : Packages

```
-- package header, declarations only
CREATE OR ALTER PACKAGE TEST
AS
BEGIN
  PROCEDURE P1(I INT) RETURNS (O INT);   -- public procedure
END

-- package body, implementation
RECREATE PACKAGE BODY TEST
AS
BEGIN
  FUNCTION F1(I INT) RETURNS INT;    -- private function

  PROCEDURE P1(I INT) RETURNS (O INT)
  AS
  BEGIN
  END;

  FUNCTION F1(I INT) RETURNS INT
  AS
  BEGIN
    RETURN 0;
  END;
END
```


DDL : IDENTITY columns

Syntax

```
<column definition> ::=
  <name> <type> GENERATED BY DEFAULT AS IDENTITY <constraints>
```

Rules

- Column type – INTEGER or NUMERIC(P, 0)
- Implicit NOT NULL
- Not guarantees uniqueness
- Can not have DEFAULT clause
- Can not be COMPUTED BY

DDL : DDL triggers

Syntax

```
<ddl-trigger> ::=

    {CREATE | RECREATE | CREATE OR ALTER} TRIGGER <name>
       [ACTIVE | INACTIVE] {BEFORE | AFTER} <ddl event>
       [POSITION <n>]

<ddl event> ::=

    ANY DDL STATEMENT
 | <ddl event item> [{OR <ddl event item>}...]

<ddl event item> ::=

    {CREATE | ALTER | DROP}

    {TABLE | PROCEDURE | FUNCTION | TRIGGER | EXCEPTION |
     VIEW  | DOMAIN    | SEQUENCE | INDEX   | ROLE      |
     USER  | COLLATION | PACKAGE  | PACKAGE BODY        |
     CHARACTER SET }
```


DDL : DDL triggers

New context variables (RDB$GET_CONTEXT)

- Namespace *DDL_TRIGGER*
- Defined inside DDL trigger only
- Read only
- Predefined variables:
  - *DDL_EVENT* - kind of DDL event
  - *OBJECT_NAME* – name of metadata object
  - *SQL_TEXT* - text of SQL query

PSQL : EXCEPTION with parameters

Syntax

```
EXEPTION <name> USING (param1 [, ...]);
```

Example

```
CREATE EXCEPTION EX_BAD_SP_NAME
    'Name of procedures must start with ''@1'' : ''@2''';

CREATE TRIGGER TRG_SP_CREATE BEFORE CREATE PROCEDURE
AS
DECLARE SP_NAME VARCHAR(255);
BEGIN
  SP_NAME = RDB$GET_CONTEXT('DDL_TRIGGER', 'OBJECT_NAME');

  IF (SP_NAME NOT STARTING 'SP_')
  THEN EXCEPTION EX_BAD_SP_NAME USING ('SP_', SP_NAME);
END;
```


THANK YOU FOR ATTENTION

Questions ?

Firebird official web site

Firebird tracker

hvlad@users.sf.net