Fyracle 0.8.3

Overview of supported syntax

Contents

Introduction

This booklet is a short introduction to Oracle-mode Firebird, nick named “Fyracle”. Fyracle is a development product, which is currently at release 0.8.3. Although it is functional enough to allow the well-known ERP+CRM package Compiere to start and and to perform basic tasks with it, you should not expect production-quality operation at this time.

This booklet is intended to be used together with the SQL and PL/SQL pocket guides. These guides are included with the full developer install kits. They are ***not*** part of the demo install kits.

1. Supported SQL syntax

Fyracle supports all the common statements from the Data Manipulation Language (DML).

```
sql_statement ::=
      commit_clause     |
      savepoint_clause  |
      rollback_clause   |
      lock_clause       |
      delete_clause     |
      insert_clause     |
      select_clause     |
      set_clause        |
      update_clause
```

Fyracle supports transactions through the COMMIT, SAVEPOINT and ROLLBACK statements. The comment on COMMIT is accepted, but ignored.

```
commit_clause ::=
      COMMIT [WORK] [COMMENT <string>]

save_point_clause ::=
      SAVEPOINT <ident>

rollback_clause ::=
      ROLLBACK [WORK] [TO SAVEPOINT <ident>]
```

Firebird uses the advanced record versioning paradigm to handle concurrent transactions. As a result, it does not have to lock records and tables the way most other databases do. Oracle is designed using traditional locking and only required a sort of record versioning later on. Because of this, it has lock optimization statements. These are accepted, but ignored.

```
set_clause ::=
      SET READ [ONLY|WRITE]                     |
      SET USE ROLLBACK SEGMENT <ident>          |
      SET ISOLATION LEVEL
            [SERIALIZABLE | READ COMMITTED]     |
      SET NAME <string>

lock_clause ::=
      LOCK TABLE table_ref_list
            IN lock_type MODE [NOWAIT]
```

```
lock_type ::=
      ROW [SHARE|EXCLUSIVE]          |
      SHARE [UPDATE|EXCLUSIVE]       |
      EXCLUSIVE
```

The DML statements DELETE and UPDATE accept both searched WHERE clauses and positioned WHERE clauses. The identifier must refer to the name of an open cursor.

```
where_subclause ::=
      WHERE sql_condition            |
      WHERE CURRENT OF <ident>

delete_clause ::=
      DELETE FROM table_ref where_subclause

update_item ::=
      <ident> = sql_expr_list        |
      <ident> = ( subquery_clause )

update_clause ::=
      UPDATE table_ref [[AS] <ident>]
            SET ( update_item [, update_item]... )
            where_subclause
```

The INSERT statement accepts both the simple VALUES clause and (SELECT ...) clause as source. Fyracle does not currently support more than one select item in the subquery clause in version 0.8.3

```
insert_clause ::=
      INSERT INTO table_ref [( <ident> [, <ident>]... )]
            VALUES sql_expr_list           |
      INSERT INTO table_ref [( <ident> [, <ident>]... )]
            subquery_clause
```

The Fyracle SELECT clause has a fairly full implementation of the Oracle syntax for regular datatypes. Collections are not supported in version 0.8.3. The FOR UPDATE OF clause emulates Oracle style pessimistic locking.

```
select_clause ::=
      subquery_clause
      [FOR UPDATE [OF <ident> [(column_list)]

subquery_clause ::=
      SELECT [ALL|DISTINCT|UNIQUE] select_list
            FROM from_list
            [select_into_clause]
            [WHERE sql_condition_clause]
```

```
            [[START WITH sql_condition_clause]
                  CONNECT BY sql_condition_clause]
            [GROUP BY sql_expression_list
                  [HAVING sql_condition_clause]]
            [UNION [ALL] subquery_clause]
            [ORDER BY order_by_list]

select_base ::=
      table_ref.* |
      sql_expression

select_item ::=
      select_base [[AS] <ident>]

select_list ::=
      *                                        |
      select_item [, select_item]...

from_base ::=
      table_ref               |
      ( subquery_clause )

from_item ::=
      from_base [[AS] <ident>]

join_spec ::=
      ON sql_condition_clause             |
      USING ( <ident> [, <ident>]... )

from_join ::=
      from_item                                                 |
      from_item [INNER] JOIN from_item join_spec                |
      from_item LEFT  [OUTER] JOIN from_item join_spec |
      from_item RIGHT [OUTER] JOIN from_item join_spec |
      from_item FULL  [OUTER] JOIN from_item join_spec |
      from_item CROSS JOIN from_item join_spec

from_list ::=
      from_join [, from_list]
```

The select INTO clause is only permitted in PL/SQL blocks, not in dynamic SQL statements from a client.

```
select_into_clause ::=
      INTO pls_expression_list

order_by_list ::=
      <ident> [[ASC] | DESC] [, order_by_list]
```

SQL conditions in Fyracle follow the Oracle precedence and association rules. All the core syntax is supported:

```
sql_condition_clause ::=
      membership_clause                          |
      NOT membership_clause                      |
      membership_clause OR membership_clause     |
      membership_clause AND membership_clause

membership_clause ::=
      EXISTS ( subquery_clause )                          |
      comparison_clause IS [NOT] NULL                     |
      comparison_clause BEWTEEN
            sql_expression_clause AND
            sql_expression_clause                         |
      comparison_clause [NOT] IN
            ( sql_primary_clause )                        |
      comparison_clause LIKE sql_expression_clause

comparison_clause ::=
      sql_expression = sql_expression     |
      sql_expression != sql_expression    |
      sql_expression >= sql_expression    |
      sql_expression <= sql_expression    |
      sql_expression < sql_expression     |
      sql_expression > sql_expression
```

SQL conditions in Fyracle follow the Oracle precedence and association rules. All the core syntax for scalar datatypes is supported:

```
sql_expression_list ::=
      sql_expression [, sql_expression]...

sql_expression ::=
      sql_factor                          |
      sql_factor + sql_expression         |
      sql_factor - sql_expression         |
      sql_factor || sql_expression

sql_factor ::=
      sql_unary                     |
      sql_unary * sql_factor        |
      sql_unary / sql_factor
```

The PRIOR keyword is only permitted inside a CONNECT BY clause.

```
sql_unary ::=
      [[+]|-] sql_builtin           |
      PRIOR sql_builtin             |
      sql_unary ^ sql_builtin
```

```
sql_case_list ::=
      WHEN sql_expression
            THEN  sql_expression [sql_case_list]      |
      WHEN sql_expression
            THEN  sql_expression ELSE sql_expression

sql_builtin ::=
      builtin_fun ( sql_expression_list )       |
      CASE sql_expression sql_case_list END     |
      CASE sql_case_list END                    |
      sql_primary
```

The outer join operator “(+)” is only supported inside the where clause of a SELECT statement. The identifier must refer to a join table.

```
sql_primary ::=
      SUM([[ALL]|DISTINCT] sql_expression)      |
      MIN([[ALL]|DISTINCT] sql_expression)      |
      MAX([[ALL]|DISTINCT] sql_expression)      |
      AVG([[ALL]|DISTINCT] sql_expression)      |
      COUNT([[ALL]|DISTINCT] sql_expression)    |
      COUNT(*)                                  |
      ( subquery_clause )                       |
      ( expression_list )                       |
      : <ident>                                 |
      <ident>.CURVAL                            |
      <ident>.NEXTVAL                           |
      <ident>( sql_expression_list )            |
      <ident>(+)                                |
      <ident>
```

The list of supported built-in functions is still relatively short; we welcome all community contributed code suggestions for implementing additional Oracle built-in's.

```
builtin_fun ::=
      NULLIF | NVL | COALESCE | DECODE | SUBSTR |
      TO_CHAR | TO_DATE | ROUND | TRUNC | ADD_MONTHS |
      REPLACE | TO_NUMBER | INITCAP | TRIM | UPPER |
      ABS | LENGTH | USER | EMPTY_BLOB | EMPTY_CLOB
```

The scalar SQL types are supported. Most type synonyms (e.g. NATURAL, STRING, etc.) have not been implmented yet. Collections are not implmented either. The same datatypes are also accepted in the Data Definition Language (DDL), see chapter 3.

```
sql_type_clause ::=
      CHAR( <int> [BYTE|CHAR] )
      NCHAR( <int> [BYTE|CHAR] )
      VARCHAR( <int> [BYTE|CHAR] )
      NVARCHAR( <int> [BYTE|CHAR] )
      VARCHAR2( <int> [BYTE|CHAR] )
      NVARCHAR2( <int> [BYTE|CHAR] )
      NUMBER
      NUMBER( <int> [, <int>] )
      DATE
      TIMESTAMP [( <int> )] [WITH [LOCAL] TIME ZONE]
      LONG [RAW]
      BLOB
      CLOB
      NCLOB
      ROWID
```

2. Supported PL/SQL syntax

In version 0.8.2, the range of datatypes supported by Fyracle's PL/SQL is the same as is supported for the SQL statements:

```
pls_type_clause ::=
      CHAR( <int> [BYTE|CHAR] )
      NCHAR( <int> [BYTE|CHAR] )
      VARCHAR( <int> [BYTE|CHAR] )
      NVARCHAR( <int> [BYTE|CHAR] )
      VARCHAR2( <int> [BYTE|CHAR] )
      NVARCHAR2( <int> [BYTE|CHAR] )
      NUMBER
      NUMBER( <int> [, <int>] )
      DATE
      TIMESTAMP [( <int> )] [WITH [LOCAL] TIME ZONE]
      LONG [RAW]
      BLOB
      CLOB
      NCLOB
      ROWID
```

Datatypes can be specified in terms of already defined objects, and user defined types are suypported too.

```
pls_datatype ::=
      pls_type_clause         |
      <ident>                 |
      <ident>%TYPE            |
      <ident>%ROWTYPE
```

Every declaration block starts with non-code declarations, followed by procedure and function definitions.

```
declaration_list ::=
      [type_var_decls] [proc_func_decls]

type_var_decls ::=
      non_code_decl                        |
      non_code_decl ; type_var_decls

non_code_decl ::=
      variable_decl     |
      exception_decl    |
      type_decl         |
      subtype_decl      |
      cursor_decl
```

```
variable_decl ::=
      <ident> [CONSTANT] pls_datatype
            [NOT NULL] [ [:=|DEFAULT] pls_expression ]

exception_decl ::=
      <ident> EXCEPTION

type_del ::=
      TYPE <ident> IS RECORD
            ( variable_decl [, variable_decl]... )

subtype_decl ::=
      SUBTYPE <ident> IS pls_datatype [NOT NULL]

cursor_decl ::=
      CURSOR <ident> [( param_list )]
            IS subquery_clause
            [FOR UPDATE [OF tablecol_list]]
```

Code declarations consist of either procedure or function declarations. Forward declarations are supported, to allow for mutually recursive code.

```
proc_func_decls ::=
      proc_clause                         |
      func_clause                         |
      proc_clause ; proc_func_clause      |
      func_clause ; proc_func_clause

param_decl ::=
      <ident> [IN|OUT|IN OUT] [NOCOPY]
            pls_datatype [ [:=|DEFAULT] pls_expression ]

param_list ::=
      param_decl              |
      param_decl , param_list

proc_clause ::=
      PROCEDURE <ident> [( param_list )]              |
      PROCEDURE <ident> [( param_list )] [IS|AS]
            [PRAGMA AUTONOMOUS_TRANSACTION]
            plsql_block_clause
```

```
func_clause ::=
      FUNCTION <ident> [( param_list )]
            RETURN <pls_data_type>                    |
      FUNCTION <ident> [( param_list )]
            RETURN <pls_data_type> [IS|AS]
            [PRAGMA PARALLEL_ENABLE]
            [PRAGMA DETERMINISTIC]
            plsql_block_clause
```

Fyracle supports all common PL/SQL statements:

```
statement ::=
      <<label>> statement ;   |
      statement ;

plsql_statement ::=
      sql_statement           |
      assignment_clause       |
      call_clause             |
      case_clause             |
      close_clause            |
      execute_clause          |
      exit_clause             |
      fetch_clause            |
      goto_clause             |
      if_clause               |
      for_clause              |
      while_clause            |
      loop_clause             |
      null_clause             |
      open_clause             |
      raise_clause            |
      return_clause           |
      begin_clause            |
      declare_clause

statement_list ::=
      statement               |
      statement , statement_list

assignment_clause ::=
      <ident> := pls_expression

call_clause ::=
      <ident> [(actuals_list)]

case_list ::=
      WHEN <pls_expression>
            THEN statement_list [case_list]           |
      WHEN <pls_expression>
```

```
            THEN statement_list ELSE statement_list

case_clause ::=
      CASE case_list END CASE [<ident>]

open_clause ::=
      OPEN <ident> [( actuals_list )]

fetch_clause ::=
      FETCH <ident> INTO pls_expression_list

close_clause ::=
      CLOSE <ident>

execute_clause ::=
      EXECUTE IMMEDIATE pls_expression

exit_clause ::=
      EXIT [<ident>] [WHEN pls_expression]

goto_clause ::=
      GOTO <ident>

if_clause ::=
      IF pls_expression THEN statement_list
            [ELIF pls_expression THEN statment_list]...
            [ELSE statement_list]

loop_clause ::=
      LOOP statement_list END LOOP [<ident>]

for_clause ::=
      FOR <ident> IN pls_expr..pls_expr loop_clause  |
      FOR <ident> IN
            REVERSE pls_expr..pls_expr loop_clause   |
      FOR <ident> IN
            <ident>[(actuals_list)] loop_clause

while_clause ::=
      WHILE pls_expression loop_clause

raise_caluse ::=
      RAISE <ident>

null_clause ::=
      NULL

return_clause ::=
      RETURN <pls_expression>
```

```
declare_clause ::=
      DECLARE declaration_list begin_clause

begin_clause ::=
      BEGIN statement_list [exception_handlers] END

handler ::=
      WHEN <ident> THEN statement_list    |
      WHEN OTHERS THEN statement_list

exception_handlers ::=
      EXCEPTION [handler;]...
```

Like in Oracle, Fyracle does not distinguish between conditions and expressions in PL/SQL:

```
pls_expression_list ::=
      pls_expression                              |
      pls_expression , pls_expression_list

pls_expression ::=
      pls_membership_clause                                |
      NOT pls_membership_clause                            |
      pls_membership_clause OR pls_membership_clause       |
      pls_membership_clause AND Pls_membership_clause

pls_membership_clause ::=
      pls_comparison_clause IS [NOT] NULL
                  |
      pls_comparison_clause BEWTEEN
pls_expression_clause AND pls_expression_clause      |
      pls_comparison_clause [NOT] IN
( pls_primary_clause )                     |
      pls_comparison_clause LIKE pls_expression_clause

pls_comparison_clause ::=
      pls_expression = pls_expression     |
      pls_expression != pls_expression    |
      pls_expression >= pls_expression    |
      pls_expression <= pls_expression    |
      pls_expression < pls_expression     |
      pls_expression > pls_expression

pls_expression ::=
      pls_factor                          |
      pls_factor + pls_expression         |
      pls_factor - pls_expression         |
      pls_factor || pls_expression
```

```
pls_factor ::=
      pls_unary                     |
      pls_unary * pls_factor        |
      pls_unary / pls_factor

pls_unary ::=
      [[+]|-] pls_builtin           |
      pls_unary ^ pls_builtin

pls_case_list ::=
      WHEN pls_expression
            THEN  pls_expression [pls_case_list]      |
      WHEN pls_expression
            THEN  pls_expression ELSE pls_expression

pls_builtin ::=
      builtin_fun ( pls_expression_list )       |
      CASE pls_expression pls_case_list END     |
      CASE pls_case_list END                    |
      pls_primary
```

As a special case DBMS_OUTPUT.PUT_LINE is hardcoded as a predefined function. Use of the implicit cursor “SQL” is supported. Use of named parameters (“=>” syntax) is supported.

```
pls_primary ::=
      ( expression_list )                       |
      SQL%OPEN                                  |
      SQL%FOUND                                 |
      SQL%NOTFOUND                              |
      SQL%ROWCOUNT                              |
      <ident>%OPEN                              |
      <ident>%FOUND                             |
      <ident>%NOTFOUND                          |
      <ident>%ROWCOUNT                          |
      <ident>( actuals_list )                   |
      DBMS_OUTPUT.PUT_LINE( pls_expression )    |
      <literal>

actuals_list ::=
      actual                    |
      actual , actual_list

actual ::=
      pls_expression                |
      <ident> => pls_expression
```

3. Supported Data Definition syntax

In version 0.8.2, the range of supported DDL syntax is pretty narrow. This is not a reflection of limitation in the underlying Firebird RDBMS. We simply did not yet get around to implementing much more.

```
DDL_clause ::=
      create_clause     |
      drop_clause       |
      alter_clause      |
      comment_clause

create_clause ::=
      create_proc_clause      |
      create_func_clause      |
      create_index_clause     |
      create_seq_clause       |
      create_table_clause     |
      create_trigger_clause   |
      create_view_clause

drop_clause ::=
      drop_proc_clause        |
      drop_func_clause        |
      drop_index_clause       |
      drop_seq_clause         |
      drop_table_clause       |
      drop_trigger_clause     |
      drop_view_clause

alter_clause ::=
      alter_index_clause      |
      alter_table_clause

comment_clause ::=
      COMMENT ON [ TABLE | COLUMN ] IS <string>

create_proc_clause ::=
      CREATE [OR REPLACE] <plsql_proc_clause>

drop_proc_clause ::=
      DROP PROCEDURE <ident>

creat_func_clause ::=
      CREATE [OR REPLACE] <plsql_func_clause>

drop_func_clause ::=
      DROP FUNCTION <ident>
```

```
creat_seq_clause ::=
      CREATE [OR REPLACE] SEQUENCE <ident>
            [START WITH <int>] [INCREMENT BY <int>]

drop_seq_clause ::=
      DROP SEQUENCE <ident>

create_view_clause ::=
      CREATE [OR REPLACE] [[NO] FORCE]
            VIEW [( <ident> [,<ident>]... )] AS subquery

drop_view_clause ::=
      DROP VIEW <ident> [CASCADE CONSTRAINTS]
```

Note: statement level triggers are not supported

```
create_trigger_clause ::=
      CREATE [OR REPLACE] TRIGGER <ident>
            [BEFORE | AFTER | INSTEAD OF] trigger_event
            ON <ident> FOR EACH ROW
            [WHEN sql_condition]
            plsql_block_clause

trigger_event ::=
      [INSERT | UPDATE | DELETE]                    |
      trigger_event OR [INSERT | UPDATE | DELETE]

drop_trigger_clause ::=
      DROP TRIGGER <ident>

create_table ::=
      CREATE [OR REPLACE] [GLOBAL TEMPORARY] TABLE
      ( property_clause [, property_clause] )
      [ON COMMIT [DELETE | PRESERVE]]

property_clause ::=
      column_def_clause |
      [CONSTRAINT <ident>] table_constraint_clause
            [USING INDEX TABLESPACE <ident>]

column_def_clause ::=
      <ident> sql_type_clause
            [DEFAULT <sql_expr_clause>]
             [[CONSTRAINT <ident>] constraint_clause]

constraint_clause ::=
      [NOT] NULL        |
      UNIQUE            |
      PRIMARY           |
```

```
      CHECK ( sql_condition ) |
      REFERENCES <ident> [ON DELETE [CASCADE|SET NULL]]

table_constraint_clause ::=
      CHECK ( sql_condition )
      UNIQUE ( <ident> [, <ident]... )                |
      PRIMARY ( <ident> [, <ident]... )               |
      REFERENCES <ident> ( <ident> [, <ident]... )
            [ON DELETE [CASCADE|SET NULL]]            |
      FOREIGN KEY ( <ident> [, <ident]... )
            REFERENCES <ident> ( <ident> [, <ident]... )
            [ON DELETE [CASCADE|SET NULL]]

alter_table_clause ::=
      ALTER TABLE ADD CONSTRAINT <ident>
            table_constraint_clause
            [USING INDEX TABLESPACE <ident>]

drop_table_clause ::=
      DROP TABLE <ident>

create_index_clause ::=
      CREATE [OR REPLACE] [UNIQUE|BITMAP]
            INDEX <ident> ON <ident> [<ident>]
            (index_col [, index_col]... )

index_col ::=
      <ident> [ASC|DESC]

drop_index_clause ::=
      DROP INDEX <ident>
```