DML - Data Manipulation Language

DML is the abbreviation for Data Manipulation Language. DML is a collection of SQL commands that can be used to manipulate a database's data.

DML is part of the SQL language commands, which execute queries with database objects and changes to their contents. The various DML commands can be used to create, edit, evaluate and delete data in a database. DML commands are a subarea of SQL; the range of the SQL language is composed of DML and DDL together.

SIUD

SIUD is the abbreviation for SELECT, INSERT, UPDATE, DELETE, which are the four DML commands used for data manipulation.

See also:

- Create SIUD Procedures
- #INSERTEX (CSV file import)|INSERTEX

SELECT

Please refer to SQL Language Reference / Data Retrieval / SELECT for details.

INSERT

Adds one or more new rows to a specified table. Available in gpre, DSQL, and isql.

Syntax

```
INSERT [TRANSACTION transaction] INTO object [(col [, col …])]
   {VALUES (val [, val …]) | select_expr};

<object> = tablename | viewname

<val> = {:variable | constant | expr
   | function | udf ([val [, val …]])
   | NULL | USER | RDB$DB_KEY | ?} [COLLATE collation]

<constant> = num | 'string' | charsetname 'string'

<function> = CAST (val AS datatype)
   | UPPER (val)
   | GEN_ID (generator, val)
```

Argument	Description
expr	
select_expr	A SELECT that returns zero or more rows and where the number of columns in each row is the same as the number of items to be inserted.

Notes on the INSERT statement

- In SQL and isql, you cannot use val as a parameter placeholder (like "?").
- In DSQL and isql, val cannot be a variable.
- You cannot specify a COLLATE clause for Blob columns.

Important: In SQL statements passed to DSQL, omit the terminating semicolon. In embedded applications written in C and C++, and in isql, the semicolon is a terminating symbol for the statement, so it must be included.

Argument	Description
TRANSACTION transaction	Name of the transaction that controls the execution of the INSERT.
INTO object	Name of an existing table or view into which to insert data.
col	Name of an existing column in a table or view into which to insert values.
VALUES (val [, val ...])	Lists values to insert into the table or view; values must be listed in the same order as the target columns.
select_expr	Query that returns row values to insert into target columns.

Description

INSERT stores one or more new rows of data in an existing table or view. INSERT is one of the database privileges controlled by the GRANT and REVOKE statements. Values are inserted into a row in column order unless an optional list of target columns is provided. If the target list of columns is a subset of available columns, default or NULL values are automatically stored in all unlisted columns. If the optional list of target columns is omitted, the VALUES clause must provide values to insert into all columns in the table.

To insert a single row of data, the VALUES clause should include a specific list of values to insert.

To insert multiple rows of data, specify a select_expr that retrieves existing data from another table to insert into this one. The selected columns must correspond to the columns listed for insert.

Important: It is legal to select from the same table into which insertions are made, but this practice is not advised because it may result in infinite row insertions.

The TRANSACTION clause can be used in multiple transaction SQL applications to specify which transaction controls the INSERT operation. The TRANSACTION clause is not available in DSQL or isql.

Examples

The following statement, from an embedded SQL application, adds a row to a table, assigning values from host-language variables to two columns:

```
EXEC SQL
   INSERT INTO EMPLOYEE_PROJECT (EMP_NO, PROJ_ID)
```

```
      VALUES (:emp_no, :proj_id);
```

The next isql statement specifies values to insert into a table with a SELECT statement:

```
INSERT INTO PROJECTS
   SELECT * FROM NEW_PROJECTS
   WHERE NEW_PROJECTS.START_DATE > '6-JUN-1994';
```

See also:

INSERT SET TRANSACTION UPDATE OR INSERT

back to top of page

UPDATE

Changes the data in all or part of an existing row in a table, view, or active set of a cursor. Available in gpre, DSQL, and isql.

Syntax SQL form

```
UPDATE [TRANSACTION transaction] {table | view}
    SET col = val [, col = val ...]
    [WHERE search_condition | WHERE CURRENT OF cursor]
    [ORDER BY order_list]
    [ROWS value [TO upper_value] [BY step_value][PERCENT][WITH TIES]];

 DSQL and isql form:

 UPDATE {table | view}
    SET col = val [, col = val ...]
    [WHERE search_condition
    [ORDER BY order_list]
    [ROWS value [TO upper_value] [BY step_value][PERCENT][WITH TIES]]

 <val> = {
    col [array_dim]
    | :variable
    | constant
    | expr
    | function
    | udf ([val [, val ...]])
    | NULL
    | USER
    | ?}
    [COLLATE collation]

 <array_dim> = [[x:]y [, [x:]y ...]]

 <constant> = num | 'string' | charsetname 'string'

 <function> = CAST (val AS datatype)
    | UPPER (val)
    | GEN_ID (generator, val)
```

```
<expr> = A valid SQL expression that results in a single value.
<search_condition> = See CREATE TABLE for a full description.
```

Notes on the UPDATE statement

- In SQL and isql, you cannot use val as a parameter placeholder (like “?”).

- In DSQL and isql, val cannot be a variable.
- You cannot specify a COLLATE clause for Blob columns.

Argument	Description
TRANSACTION	transaction Name of the transaction under control of which the statement is executed.
table / view	Name of an existing table or view to update.
SET col = val	Specifies the columns to change and the values to assign to those columns.
WHERE search_condition	Searched update only; specifies the conditions a row must meet to be modified.
WHERE CURRENT OF cursor	Positioned update only; specifies that the current row of a cursor's active set is to be modified. Not available in DSQL and isql.
ORDER BY order_list	Specifies columns to order, either by column name or ordinal number in the query, and the sort order (ASC or DESC) for the returned rows.

```
ROWS1 value
[TO upper_value]
[BY step_value]
[PERCENT][WITH TIES]
```

- Value is the total number of rows to return if used by itself.
- Value is the starting row number to return if used with TO.
- Value is the percent if used with PERCENT.
- Upper_value is the last row or highest percent to return.
- If step_value = n, returns every nth row, or n percent rows.
- PERCENT causes all previous ROWS values to be interpreted as percents.
- WITH TIES returns additional duplicate rows when the last value in the ordered sequence is the same as values in subsequent rows of the result set; must be used in conjunction with ORDER BY.

1 Please also refer to ROWS syntax for Firebird 2.0 syntax, description and examples.

New in Firebird 2.0: New extensions to UPDATE and DELETE syntaxes - ROWS specifications and PLAN and ORDER BY clauses can now be used in UPDATE and DELETE statements.

Users can now specify explicit plans for UPDATE/DELETE statements in order to optimize them manually. It is also possible to limit the number of affected rows with a ROWS clause, optionally used in combination with an ORDER BY clause to have a sorted record set.

Syntax

```
UPDATE ... SET ... WHERE ...
[PLAN <plan items>]
[ORDER BY <value list>]
[ROWS <value> [TO <value>]]
```

Description

UPDATE modifies one or more existing rows in a table or view. UPDATE is one of the database privileges controlled by GRANT and REVOKE.

For searched updates, the optional WHERE clause can be used to restrict updates to a subset of rows in the table. Searched updates cannot update array slices.

Important

Without a WHERE clause, a searched update modifies all rows in a table.

When performing a positioned update with a cursor, the WHERE CURRENT OF clause must be specified to update one row at a time in the active set.

Note: When updating a blob column, UPDATE replaces the entire blob with a new value.

Examples

The following isql statement modifies a column for all rows in a table:

```
UPDATE CITIES
   SET POPULATION = POPULATION * 1.03;
```

The next embedded SQL statement uses a WHERE clause to restrict column modification to a subset of rows:

```
EXEC SQL
   UPDATE PROJECT
   SET PROJ_DESC = :blob_id
   WHERE PROJ_ID = :proj_id;
```

See also:

UPDATE UPDATE OR INSERT Firebird 2.5 Release Notes: OldSetClauseSemantics

back to top of page

DELETE

Removes rows in a table or in the active set of a cursor. Available in gpre, DSQL, and isql.

Syntax SQL and DSQL form

Important: Omit the terminating semicolon for DSQL.

```
DELETE [TRANSACTION transaction] FROM table
   {[WHERE search_condition] | WHERE CURRENT OF cursor}
   [ORDER BY order_list]
   [ROWS value [TO upper_value] [BY step_value][PERCENT][WITH TIES]];
```

<search_condition> = Search condition as specified in SELECT.

isql form

```
DELETE FROM TABLE [WHERE search_condition];
```

Argument	Description
TRANSACTION transaction	Name of the transaction under control of which the statement is executed; SQL only.
table	Name of the table from which to delete rows.
WHERE search_condition	Search condition that specifies the rows to delete; without this clause, DELETE affects all rows in the specified table or view.
WHERE CURRENT OF cursor	Specifies that the current row in the active set of cursor is to be deleted.
ORDER BY order_list	Specifies columns to order, either by column name or ordinal number in the query, and the sort order (ASC or DESC) for the returned rows.

```
ROWS1 value
[TO upper_value]
[BY step_value]
[PERCENT][WITH TIES]
```

- Value is the total number of rows to return if used by itself.
- Value is the starting row number to return if used with TO.
- Value is the percent if used with PERCENT.
- Upper_value is the last row or highest percent to return.
- If step_value = n, returns every nth row, or n percent rows.
- PERCENT causes all previous ROWS values to be interpreted as percents.
- WITH TIES returns additional duplicate rows when the last value in the ordered sequence is the same as values in subsequent rows of the result set; must be used in conjunction with ORDER BY.

1 Please also refer to ROWS syntax for Firebird 2.0 syntax, description and examples.

New in Firebird 2.0: New extensions to UPDATE and DELETE syntaxes- ROWS specifications and PLAN and ORDER BY clauses can now be used in UPDATE and DELETE statements.

Users can now specify explicit plans for UPDATE/DELETE statements in order to optimize them manually. It is also possible to limit the number of affected rows with a ROWS clause, optionally used in combination with an ORDER BY clause to have a sorted recordset.

Syntax

```
DELETE ... FROM ...
[PLAN <plan items>]
[ORDER BY <value list>]
[ROWS <value> [TO <value>]]
```

Description

DELETE specifies one or more rows to delete from a table or . DELETE is one of the database privileges controlled by the GRANT and REVOKE statements.

The TRANSACTION clause can be used in multiple transaction SQL applications to specify which transaction controls the DELETE operation. The TRANSACTION clause is not available in DSQL or isql.

For searched deletions, the optional WHERE clause can be used to restrict deletions to a subset of rows in the table.

Important

Without a WHERE clause, a searched delete removes all rows from a table.

When performing a positioned delete with a cursor, the WHERE CURRENT OF clause must be specified to delete one row at a time from the active set.

Examples

The following isql statement deletes all rows in a table:

```
DELETE FROM EMPLOYEE_PROJECT;
```

The next embedded SQL statement is a searched delete in an embedded application. It deletes all rows where a host-language variable equals a column value.

```
EXEC SQL
   DELETE FROM SALARY_HISTORY
   WHERE EMP_NO = :emp_num;
```

The following embedded SQL statements use a cursor and the WHERE CURRENT OF option to delete rows from CITIES with a population less than the host variable, min_pop. They declare and open a cursor that finds qualifying cities, fetch rows into the cursor, and delete the current row pointed to by the cursor.

```
EXEC SQL
   DECLARE SMALL_CITIES CURSOR FOR
   SELECT CITY, STATE
   FROM CITIES
   WHERE POPULATION < :min_pop;

EXEC SQL
   OPEN SMALL_CITIES;

EXEC SQL
   FETCH SMALL_CITIES INTO :cityname, :statecode;
   WHILE (!SQLCODE)
     {EXEC SQL
        DELETE FROM CITIES
        WHERE CURRENT OF SMALL_CITIES;
     EXEC SQL
        FETCH SMALL_CITIES INTO :cityname, :statecode;}
EXEC SQL
  CLOSE SMALL_CITIES;
```

See also: DELETE

MERGE

MERGE is used to combine the data of multiple tables. It is something of a combination of the INSERT

and UPDATE elements.