Manual de SQL

Tutorial

Introducción

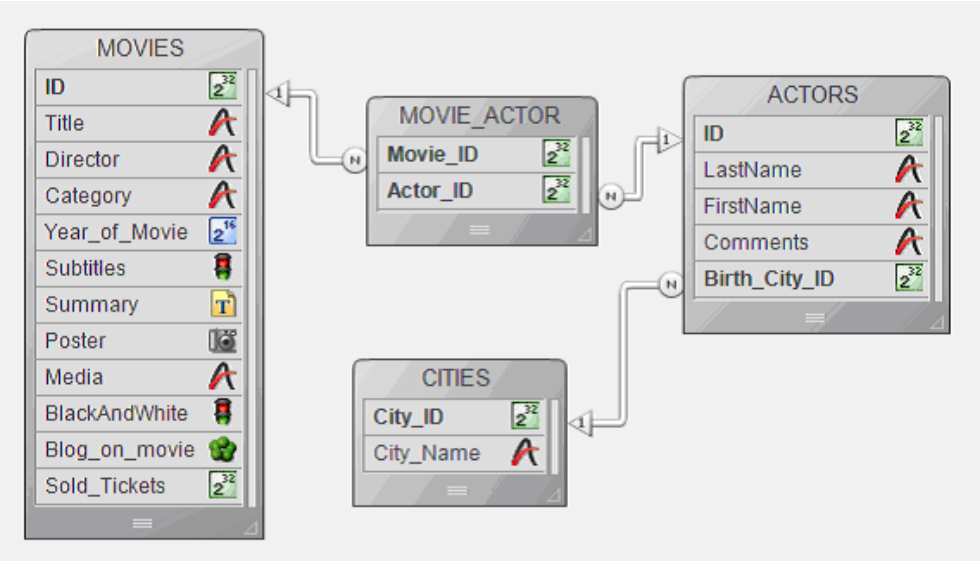
El SQL (Structured Query Language) es una herramienta de creación, organización, gestión y recuperación de datos almacenados por una base de datos informática. SQL no es un sistema de gestión de bases de datos en sí, ni un producto independiente, sin embargo, SQL es una parte integral de un sistema de gestión de base de datos, un lenguaje y una herramienta utilizada para comunicarse con este sistema.

El objetivo de este tutorial no es enseñarte a trabajar con SQL (para esto se puede encontrar documentación y enlaces en Internet), ni a enseñarle a utilizar y/o programar en 4D. En cambio, su propósito es mostrarle cómo trabajar con código SQL dentro de código 4D, cómo recuperar datos utilizando comandos SQL, cómo pasar parámetros y cómo obtener resultados después de una consulta SQL.

Descripción de la base de datos que acompaña este manual

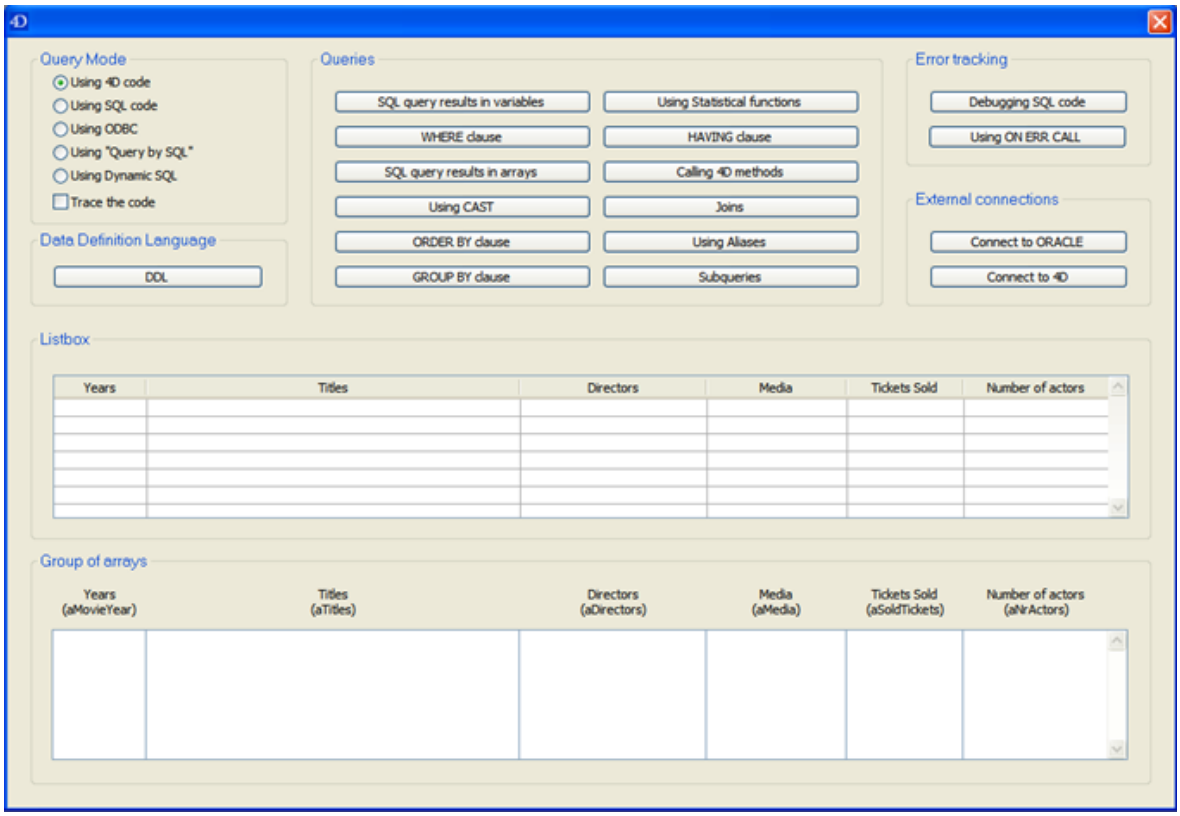
Todos los ejemplos que se detallan en este documento se han probado y verificado plenamente en una de las bases de datos de ejemplo llamada "4D SQL Code Samples" que puede descargar de nuestro servidor ftp (*ftp://ftp-public.4d.fr/Documents/Products_Documentation/LastVersions/Line_12/4D_SQL_Code_Samples.zip*).

La estructura es la siguiente:



La tabla MOVIES contiene información sobre 50 películas, incluyendo título, director, categoría (Acción, Animación, Comedia, Crimen, Drama, etc), el año de salida, si tiene o no tiene subtítulos, un breve resumen , una foto de su cartel, el tipo de medio (DVD, VHS, DivX), si es en blanco y negro, un blog guardado en un BLOB y el número de entradas vendidas.
La tabla ACTORS contiene información sobre los actores de las películas tales como identificación, nombres y apellidos, comentarios y el ID de la ciudad donde nació el actor.
La tabla CITIES contiene información sobre el nombre e ID de las ciudades donde los actores nacieron.
La tabla MOVIE_ACTOR se utiliza para establecer una relación de Muchos a Muchos entre las tablas MOVIES y ACTORS.

Toda la información que necesita para lanzar los ejemplos descritos en el tutorial se encuentra en la siguiente ventana principal que puede acceder seleccionando el comando de menú **Demo SQL>Show samples**:


Query Mode
Using 4D code
Using SQL code
Using ODBC
Using "Query by SQL"
Using Dynamic SQL
Trace the code
Data Definition Language
DDL
Queries
SQL query results in variables
Using Statistical functions
WHERE clause
HAVING clause
SQL query results in arrays
Calling 4D methods
Using CAST
Joins
ORDER BY clause
Using Aliases
GROUP BY clause
Subqueries
Error tracking
Debugging SQL code
Using ON ERR CALL
External connections
Connect to ORACLE
Connect to 4D
Listbox
Years
Titles
Directors
Media
Tickets Sold
Number of actors
Group of arrays
Years (aMovieYear)
Titles (aTitles)
Directors (aDirectors)
Media (aMedia)
Tickets Sold (aSoldTickets)
Number of actors (aNrActors)

Recibir el resultado de una petición SQL en una variable

Comencemos por una búsqueda simple: queremos saber cuántas películas están en nuestra videoteca. En el lenguaje 4D, el código sería:

```
C_LONGINT($AllMovies)
$AllMovies:=0
ALL RECORDS([MOVIES])
$AllMovies:=Records in selection([MOVIES])
ALERT("La videoteca contiene "+String($AllMovies)+"películas ")
```

- La primera forma de interactuar de una manera similar con el motor SQL es ubicar la petición entre las etiquetas **Begin SQL** y **End SQL**. De esta forma, la búsqueda anterior se convierte en:

```
C_LONGINT($AllMovies)
$AllMovies:=0
Begin SQL
    SELECT COUNT(*)
    FROM MOVIES
    INTO <<$AllMovies>>
End SQL
ALERT("La videoteca contiene "+String($AllMovies)+" películas")
```

- Como puede ver, puede recuperar el resultado de la búsqueda en una variable (en nuestro caso $AllMovies) que está entre los símbolos "<<" y ">>".
  Otra forma de referenciar todo tipo de expresión 4D valida (variable, campo, array, "expresión…") es poner dos puntos ":" antes de la expresión:

```
C_LONGINT($AllMovies)
$AllMovies:=0
Begin SQL
    SELECT COUNT(*)
    FROM MOVIES
    INTO :$AllMovies
End SQL
ALERT("La videoteca contiene "+String($AllMovies)+" películas")
```

  Se debe prestar atención especial a las variables interproceso, donde la notación es un poco diferente: debe poner una variable interproceso entre corchetes "[" y "]":

```
C_LONGINT(<>AllMovies)
<>AllMovies:=0
Begin SQL
    SELECT COUNT(*)
```

```
    FROM MOVIES
    INTO <<[<>AllMovies]>>
End SQL
ALERT("La videoteca contiene "+String(<>AllMovies)+" películas")
```

- La segunda forma de interactuar con el motor SQL es utilizar los comandos SQL genéricos integrados (compatibles ODBC). La búsqueda simple se convierte en:

```
C_LONGINT($AllMovies)
$AllMovies:=0
 ` Inicializa una conexión con el motor SQL interno
SQL LOGIN(SQL_INTERNAL;"";"")
 ` Ejecuta la búsqueda y devuelve el resultado en la variable $AllMovies
SQL EXECUTE("SELECT COUNT(*) FROM MOVIES";$AllMovies)
 ` Recupera todos los registros encontrados
SQL LOAD RECORD(SQL all records)
 ` Cierra la conexión
SQL LOGOUT
ALERT("La videoteca contiene "+String($AllMovies)+" películas")
```

Para mayor información sobre los comandos SQL genéricos, consulte el capítulo "**SQL** del manual Lenguaje 4D.

- La tercera forma de interactuar con el motor SQL es utilizar el comando 4D **QUERY BY SQL**. En este caso, la búsqueda simple se convierte en:

```
C_LONGINT($AllMovies)
$AllMovies:=0
QUERY BY SQL([MOVIES];"ID <> 0")
$AllMovies:=Records in selection([MOVIES])
ALERT("La videoteca contiene "+String($AllMovies)+" películas")
```

El comando **QUERY BY SQL** ejecuta una búsqueda de tipo *SELECT* que puede escribirse de esta forma:

```
SELECT *
FROM myTable
WHERE <SQL_Formula>
```

myTable es el nombre de la tabla pasada en el primer parámetro y **SQL_Formula** es el texto de la búsqueda pasado como segundo parámetro:

```
QUERY BY SQL(myTable;SQL_Formula)
```

En nuestro caso no hay cláusula **WHERE**, entonces forzamos una: "ID <> 0". El equivalente de la búsqueda en código SQL es:

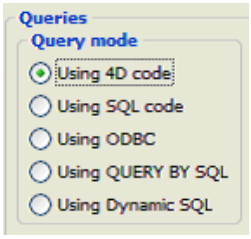
```
SELECT *
FROM MOVIES
WHERE ID <> 0
```

- La cuarta forma de interactuar con el motor SQL es utilizar el comando SQL dinámico *EXECUTE IMMEDIATE*. El código se convierte en:

```
C_LONGINT(AllMovies)
AllMovies:=0
C_TEXT(tQueryTxt)
tQueryTxt:="SELECT COUNT(*) FROM MOVIES INTO :AllMovies"
Begin SQL
    EXECUTE IMMEDIATE :tQueryTxt;
End SQL
ALERT("La videoteca contiene "+String(AllMovies)+" películas")
```

Atención: puede ver que en este último ejemplo, utilizamos variables proceso. Esto es necesario si quiere utilizar la base en modo compilado. En este contexto, en efecto, no es posible utilizar variables locales con el comando **EXECUTE IMMEDIATE**.

Para probar todos estos ejemplos, lance la base "4D SQL Code Samples" y muestre la caja de diálogo principal. A la izquierda de la ventana, puede elegir el modo de interrogación del motor de 4D:



Luego presione el botón **SQL query results in variables**.

Utilizar la cláusula WHERE

Si ahora queremos saber cuántas películas realizadas desde 1960 están en la videoteca.
El código 4D sería:

```
C_LONGINT($NoMovies)

$NoMovies:=0
REDUCE SELECTION([MOVIES];0)
QUERY([MOVIES];[MOVIES]Year_of_Movie>=1960)
$NoMovies:=Records in selection([MOVIES])
ALERT("La videoteca contiene "+String($NoMovies)+" películas realizadas desde 1960")
```

- Utilizando el código SQL:

```
C_LONGINT($NoMovies)

$NoMovies:=0
REDUCE SELECTION([MOVIES];0)
Begin SQL
    SELECT COUNT(*)
    FROM MOVIES
    WHERE Year_of_Movie >= 1960
    INTO :$NoMovies;
End SQL
ALERT("La videoteca contiene "+String($NoMovies)+" películas realizadas desde 1960")
```

- Utilizando los comandos SQL genéricos:

```
C_LONGINT($NoMovies)
$NoMovies:=0
REDUCE SELECTION([MOVIES];0)

SQL LOGIN(SQL_INTERNAL;"";"")
SQL EXECUTE("SELECT COUNT(*) FROM MOVIES WHERE Year_of_Movie >= 1960";$NoMovies)
SQL LOAD RECORD(SQL all records)
SQL LOGOUT
ALERT("La videoteca contiene "+String($NoMovies)+" películas realizadas desde 1960")
```

- Utilizando el comando **QUERY BY SQL**:

```
C_LONGINT($NoMovies)

$NoMovies:=0
REDUCE SELECTION([MOVIES];0)
```

```
QUERY BY SQL([MOVIES];"Year_of_Movie >= 1960")
$NoMovies:=Records in selection([MOVIES])
ALERT("La videoteca contiene "+String($NoMovies)+" películas realizadas desde 1960")
```

- Utilizando el comando SQL *EXECUTE IMMEDIATE*:

```
C_LONGINT($NoMovies)
C_TEXT($tQueryTxt)

$NoMovies:=0
REDUCE SELECTION([MOVIES];0)
$tQueryTxt:="SELECT COUNT(*) FROM MOVIES WHERE Year_of_Movie >= 1960 INTO :$NoMovies;"
Begin SQL
    EXECUTE IMMEDIATE :$tQueryTxt;
End SQL
ALERT("La videoteca contiene "+String($NoMovies)+" películas realizadas desde 1960")
```

Como en la sección anterior, para probar todos los ejemplos, simplemente lance la base "4D SQL Code Samples" y vaya a la ventana principal. Elija el modo de interrogación del motor de 4D y haga clic en el botón **WHERE clause**.

Recibir el resultado de una petición SQL en un array

Ahora queremos pasar una variable que contiene el año a la búsqueda SQL (y no el año) y recuperar la lista de todas las películas estrenadas en 1960 o más recientemente. Además, para cada película encontrada, también queremos información como el año, título, director, medios utilizados y boletos vendidos. La solución consiste en recibir esta información en arrays o en un list box.

- La búsqueda inicial en el código 4D sería:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)
C_LONGINT($MovieYear)

REDUCE SELECTION([MOVIES];0)
$MovieYear:=1960
QUERY([MOVIES];[MOVIES]Year_of_Movie>=$MovieYear)
SELECTION TO ARRAY([MOVIES]Year_of_Movie;aMovieYear;[MOVIES]Title;aTitles;
[MOVIES]Director;aDirectors;
 [MOVIES]Media;aMedias;[MOVIES]Sold_Tickets;aSoldTickets)
  ` Inicializa el resto de las columnas del list box con el fin de visualizar la información
ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

- Utilizando código SQL:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)
C_LONGINT($MovieYear)

$MovieYear:=1960
Begin SQL
    SELECT Year_of_Movie, Title, Director, Media, Sold_Tickets
    FROM MOVIES
    WHERE Year_of_Movie >= :$MovieYear
    INTO :aMovieYear, :aTitles, :aDirectors, :aMedias, :aSoldTickets;
End SQL
  ` Inicializa el resto de las columnas del list box con el fin de visualizar la información
ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

Como puede ver:

- Podemos pasar una variable ($MovieYear) a la búsqueda SQL usando la misma notación que para la recepción de parámetros.
- El resultado de la búsqueda SQL se guarda en los arrays aMovieYear, aTitles, aDirectories, aMedias y aSoldTickets. Los resultados se muestran en la ventana principal de dos maneras:
  - Utilizando arrays agrupados:

  - Utilizando un list box con columnas con los mismos nombres:

- Utilizando comandos SQL genéricos:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)
C_LONGINT($MovieYear)
C_TEXT($tQueryTxt)

REDUCE SELECTION([MOVIES];0)
$MovieYear:=1960
SQL LOGIN(SQL_INTERNAL;"";"")
$tQueryTxt:=""
$tQueryTxt:=$tQueryTxt+"SELECT Year_of_Movie, Title, Director, Media, Sold_Tickets"
$tQueryTxt:=$tQueryTxt+" FROM MOVIES"
$tQueryTxt:=$tQueryTxt+" WHERE Year_of_Movie >= :$MovieYear"
SQL EXECUTE($tQueryTxt;aMovieYear;aTitles;aDirectors;aMedias;aSoldTickets)
SQL LOAD RECORD(SQL all records)
SQL LOGOUT
  ` Inicializa el resto de las columnas del list box con el fin de visualizar la información
ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

- Utilizando el comando **QUERY BY SQL**:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)
C_LONGINT($MovieYear)

REDUCE SELECTION([MOVIES];0)
$MovieYear:=1960
QUERY BY SQL([MOVIES];"Year_of_Movie >= :$MovieYear")
SELECTION TO ARRAY([MOVIES]Year_of_Movie;aMovieYear;[MOVIES]Title;aTitles;
[MOVIES]Director;aDirectors;
 [MOVIES]Media;aMedias;[MOVIES]Sold_Tickets;aSoldTickets)
  ` Inicializa el resto de las columnas del list box con el fin de visualizar la información
ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

- Utilizando el comando SQL *EXECUTE IMMEDIATE*:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)
C_LONGINT($MovieYear)
C_TEXT($tQueryTxt)

REDUCE SELECTION([MOVIES];0)
$MovieYear:=1960
$tQueryTxt:=""
$tQueryTxt:=$tQueryTxt+"SELECT Year_of_Movie, Title, Director, Media, Sold_Tickets"
$tQueryTxt:=$tQueryTxt+" FROM MOVIES"
$tQueryTxt:=$tQueryTxt+" WHERE Year_of_Movie >= :$MovieYear"
$tQueryTxt:=$tQueryTxt+" INTO :aMovieYear, :aTitles, :aDirectors, :aMedias, :aSoldTickets;"
Begin SQL
   EXECUTE IMMEDIATE :$tQueryTxt;
End SQL
 ` Inicializa el resto de las columnas del list box con el fin de visualizar la información
ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

Para probar todos los ejemplos anteriores, lance la base "4D SQL Code Samples" y vaya a la ventana principal. Elija el modo de consulta y presione el botón **SQL query results in arrays**.

Utilizar la función CAST

El lenguaje SQL tiene reglas bastante restrictivas sobre la combinación de datos de diferentes tipos en las expresiones. Por lo general, el SMBD (DBMS) se encarga de la conversión automática. Sin embargo, el estándar SQL requiere que el SMDB genere un error si se intenta comparar números con cadenas de caracteres. En este contexto, la expresión *CAST* es muy importante, especialmente cuando utiliza SQL dentro de un lenguaje de programación cuyos tipos de datos no coinciden con los tipos soportados por el estándar SQL. Encontrará a continuación la búsqueda de la sección modificada ligeramente para poder utilizar la expresión *CAST*.

- El código 4D inicial sería:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)

REDUCE SELECTION([MOVIES];0)
$MovieYear:=Num("1960")
QUERY([MOVIES];[MOVIES]Year_of_Movie>=$MovieYear)
SELECTION TO ARRAY([MOVIES]Year_of_Movie;aMovieYear;[MOVIES]Title;aTitles;
[MOVIES]Director;aDirectors;
 [MOVIES]Media;aMedias;[MOVIES]Sold_Tickets;aSoldTickets)
  ` Inicializa el resto de las columnas del list box para mostrar la información
ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

- Utilizando código SQL:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)

Begin SQL
    SELECT Year_of_Movie, Title, Director, Media, Sold_Tickets
    FROM MOVIES
    WHERE Year_of_Movie >= CAST('1960' AS INT)
    INTO :aMovieYear, :aTitles, :aDirectors, :aMedias, :aSoldTickets;
End SQL
  `Inicializa el resto de las columnas del list box para mostrar la información
ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

- Utilizando comandos SQL genéricos:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)
C_TEXT($tQueryTxt)

REDUCE SELECTION([MOVIES];0)
SQL LOGIN(SQL_INTERNAL;"";"")
$tQueryTxt:=""
$tQueryTxt:=$tQueryTxt+"SELECT Year_of_Movie, Title, Director, Media, Sold_Tickets"
$tQueryTxt:=$tQueryTxt+" FROM MOVIES"
$tQueryTxt:=$tQueryTxt+" WHERE Year_of_Movie >= CAST('1960' AS INT)"
SQL EXECUTE($tQueryTxt;aMovieYear;aTitles;aDirectors;aMedias;aSoldTickets)
SQL LOAD RECORD(SQL all records)
SQL LOGOUT
  ` Inicializa el resto de las columnas del list box para mostrar la información
ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

- Utilizando el comando **QUERY BY SQL**:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)

REDUCE SELECTION([MOVIES];0)
QUERY BY SQL([MOVIES];"Year_of_Movie >= CAST('1960' AS INT)")
SELECTION TO ARRAY([MOVIES]Year_of_Movie;aMovieYear;[MOVIES]Title;aTitles;
[MOVIES]Director;aDirectors;
 [MOVIES]Media;aMedias;[MOVIES]Sold_Tickets;aSoldTickets)
  ` Inicializa el resto de las columnas del list box para mostrar la información
ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

- Utilizando el comando SQL **EXECUTE IMMEDIATE**:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)
C_TEXT($tQueryTxt)

REDUCE SELECTION([MOVIES];0)
$tQueryTxt:=""
$tQueryTxt:=$tQueryTxt+"SELECT Year_of_Movie, Title, Director, Media, Sold_Tickets"
$tQueryTxt:=$tQueryTxt+" FROM MOVIES"
$tQueryTxt:=$tQueryTxt+" WHERE Year_of_Movie >= CAST('1960' AS INT)"
$tQueryTxt:=$tQueryTxt+" INTO :aMovieYear, :aTitles, :aDirectors, :aMedias, :aSoldTickets;"
Begin SQL
```

```
    EXECUTE IMMEDIATE :$tQueryTxt;
End SQL
  ` Inicializa el resto de las columnas del list box para mostrar la información
ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

Para probar todos los ejemplos anteriores, lance la base "4D SQL Code Samples" y vaya a la ventana principal. Luego elija el modo de consulta y presione el botón **Using CAST**.

Utilizar la cláusula ORDER BY

Esta vez nos gustaría conseguir todas las películas que se lanzaron en el año 1960 o más recientemente y para cada película obtener información adicional, como el año, título, director, los medios utilizados y boletos vendidos. El resultado debe ordenarse por año.

- El código 4D inicial sería:

```
ARRAY LONGINT(aNrActors;0)
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)
C_LONGINT($MovieYear)

REDUCE SELECTION([MOVIES];0)
$MovieYear:=1960
QUERY([MOVIES];[MOVIES]Year_of_Movie>=$MovieYear)
SELECTION TO ARRAY([MOVIES]Year_of_Movie;aMovieYear;[MOVIES]Title;aTitles;
[MOVIES]Director;aDirectors;
 [MOVIES]Media;aMedias;[MOVIES]Sold_Tickets;aSoldTickets)
SORT ARRAY(aMovieYear;aTitles;aDirectors;aMedias;>)
```

- Utilizando código SQL:

```
ARRAY LONGINT(aNrActors;0)
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)
C_LONGINT($MovieYear)

REDUCE SELECTION([MOVIES];0)
$MovieYear:=1960
Begin SQL
    SELECT Year_of_Movie, Title, Director, Media, Sold_Tickets
    FROM MOVIES
    WHERE Year_of_Movie >= :$MovieYear
    ORDER BY 1
    INTO :aMovieYear, :aTitles, :aDirectors, :aMedias,   :aSoldTickets;
End SQL
```

- Utilizando comandos SQL genéricos:

```
C_TEXT($tQueryTxt)
ARRAY LONGINT(aNrActors;0)
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)
C_LONGINT($MovieYear)

REDUCE SELECTION([MOVIES];0)
$MovieYear:=1960
SQL LOGIN(SQL_INTERNAL;"";"")
$tQueryTxt:=""
$tQueryTxt:=$tQueryTxt+"SELECT Year_of_Movie, Title, Director, Media, Sold_Tickets"
$tQueryTxt:=$tQueryTxt+" FROM MOVIES"
$tQueryTxt:=$tQueryTxt+" WHERE Year_of_Movie >= :$MovieYear"
$tQueryTxt:=$tQueryTxt+" ORDER BY 1"
SQL EXECUTE($tQueryTxt;aMovieYear;aTitles;aDirectors;aMedias;aSoldTickets)
SQL LOAD RECORD(SQL all records)
SQL LOGOUT
```

- Utilizando el comando **QUERY BY SQL**:

```
ARRAY LONGINT(aNrActors;0)
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)
C_LONGINT($MovieYear)

REDUCE SELECTION([MOVIES];0)
$MovieYear:=1960
QUERY BY SQL([MOVIES];"Year_of_Movie >= :$MovieYear")
SELECTION TO ARRAY([MOVIES]Year_of_Movie;aMovieYear;[MOVIES]Title;aTitles;
[MOVIES]Director;aDirectors;
 [MOVIES]Media;aMedias;[MOVIES]Sold_Tickets;aSoldTickets)
SORT ARRAY(aMovieYear;aTitles;aDirectors;aMedias;>)
```

- Utilizando el comando SQL *EXECUTE IMMEDIATE*:

```
ARRAY LONGINT(aNrActors;0)
C_TEXT($tQueryTxt)
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)
C_LONGINT($MovieYear)

REDUCE SELECTION([MOVIES];0)
```

```
$MovieYear:=1960
$tQueryTxt:=""
$tQueryTxt:=$tQueryTxt+"SELECT Year_of_Movie, Title, Director, Media, Sold_Tickets"
$tQueryTxt:=$tQueryTxt+" FROM MOVIES"
$tQueryTxt:=$tQueryTxt+" WHERE Year_of_Movie >= :$MovieYear"
$tQueryTxt:=$tQueryTxt+" ORDER BY 1"
$tQueryTxt:=$tQueryTxt+"   INTO :aMovieYear, :aTitles, :aDirectors, :aMedias, :aSoldTickets;"
Begin SQL
   EXECUTE IMMEDIATE :$tQueryTxt;
End SQL
```

Para probar todos los ejemplos anteriores, lance la base "4D SQL Code Samples" y vaya a la ventana principal. A continuación, puede elegir el modo de búsqueda y presionar el botón **ORDER BY clause**.

Utilizar la cláusula GROUP BY

Queremos obtener información sobre el número anual total de entradas vendidas desde 1979. El resultado se ordenará por año.

- El código 4D inicial sería:

```
  ` Using standard 4D code
ARRAY INTEGER(aMovieYear;0)
ARRAY LONGINT(aSoldTickets;0)
C_LONGINT($MovieYear;$vCrtMovieYear;$i)

REDUCE SELECTION([MOVIES];0)
$MovieYear:=1979
QUERY([MOVIES];[MOVIES]Year_of_Movie>=$MovieYear)
ORDER BY([MOVIES];[MOVIES]Year_of_Movie;>)
$vCrtMovieYear:=0
$vInd:=Size of array(aMovieYear)
For($i;1;Records in selection([MOVIES]))
   If([MOVIES]Year_of_Movie#$vCrtMovieYear)
      $vCrtMovieYear:=[MOVIES]Year_of_Movie
      $vInd:=$vInd+1
      INSERT IN ARRAY(aMovieYear;$vInd;1)
      aMovieYear{$vInd}:=$vCrtMovieYear
      INSERT IN ARRAY(aSoldTickets;$vInd;1)
   End if
   aSoldTickets{$vInd}:=aSoldTickets{$vInd}+[MOVIES]Sold_Tickets
   NEXT RECORD([MOVIES])
End for
  ` Inicializa el resto de las columnas del list box para mostrar la información
ARRAY TEXT(aTitles;Size of array(aMovieYear))
ARRAY TEXT(aDirectors;Size of array(aMovieYear))
ARRAY TEXT(aMedias;Size of array(aMovieYear))
ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

- Utilizando código SQL:

```
  ` Using 4D SQL
ARRAY INTEGER(aMovieYear;0)
ARRAY LONGINT(aSoldTickets;0)
C_LONGINT($MovieYear)

REDUCE SELECTION([MOVIES];0)
$MovieYear:=1979
Begin SQL
   SELECT Year_of_Movie, SUM(Sold_Tickets)
   FROM MOVIES
   WHERE Year_of_Movie >= :$MovieYear
```

```
    GROUP BY Year_of_Movie
    ORDER BY 1
    INTO :aMovieYear,  :aSoldTickets;
End SQL
 ` Inicializa el resto de las columnas del list box para mostrar la información
ARRAY TEXT(aTitles;Size of array(aMovieYear))
ARRAY TEXT(aDirectors;Size of array(aMovieYear))
ARRAY TEXT(aMedias;Size of array(aMovieYear))
ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

- Utilizando comandos SQL genéricos:

```
 ` Using ODBC commands
C_TEXT($tQueryTxt)
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
C_LONGINT($MovieYear)

REDUCE SELECTION([MOVIES];0)
$MovieYear:=1979
SQL LOGIN(SQL_INTERNAL;"";"")
$tQueryTxt:=""
$tQueryTxt:=$tQueryTxt+"SELECT Year_of_Movie, SUM(Sold_Tickets)"
$tQueryTxt:=$tQueryTxt+" FROM MOVIES"
$tQueryTxt:=$tQueryTxt+" WHERE Year_of_Movie >= :$MovieYear"
$tQueryTxt:=$tQueryTxt+" GROUP BY Year_of_Movie"
$tQueryTxt:=$tQueryTxt+" ORDER BY 1"
SQL EXECUTE($tQueryTxt;aMovieYear;aSoldTickets)
SQL LOAD RECORD(SQL all records)
SQL LOGOUT
 ` Inicializa el resto de las columnas del list box para mostrar la información
ARRAY TEXT(aTitles;Size of array(aMovieYear))
ARRAY TEXT(aDirectors;Size of array(aMovieYear))
ARRAY TEXT(aMedias;Size of array(aMovieYear))
ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

- Utilizando el comando **QUERY BY SQL**:

```
 ` Using QUERY BY SQL
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
C_LONGINT($MovieYear)

REDUCE SELECTION([MOVIES];0)
$MovieYear:=1979
QUERY BY SQL([MOVIES];"Year_of_Movie >= :$MovieYear")
ORDER BY([MOVIES];[MOVIES]Year_of_Movie;>)
$vCrtMovieYear:=0
$vInd:=Size of array(aMovieYear)
For($i;1;Records in selection([MOVIES]))
   If([MOVIES]Year_of_Movie#$vCrtMovieYear)
      $vCrtMovieYear:=[MOVIES]Year_of_Movie
```

```
            $vInd:=$vInd+1
            INSERT IN ARRAY(aMovieYear;$vInd;1)
            aMovieYear{$vInd}:=$vCrtMovieYear
            INSERT IN ARRAY(aSoldTickets;$vInd;1)
        End if
        aSoldTickets{$vInd}:=aSoldTickets{$vInd}+[MOVIES]Sold_Tickets
        NEXT RECORD([MOVIES])
    End for
    ` Inicializa el resto de las columnas del list box para mostrar la información
    ARRAY TEXT(aTitles;Size of array(aMovieYear))
    ARRAY TEXT(aDirectors;Size of array(aMovieYear))
    ARRAY TEXT(aMedias;Size of array(aMovieYear))
    ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

- Utilizando el comando SQL *EXECUTE IMMEDIATE*:

```
    ` Using dynamic SQL by EXECUTE IMMEDIATE
    C_TEXT($tQueryTxt)
    ARRAY LONGINT(aSoldTickets;0)
    ARRAY INTEGER(aMovieYear;0)
    C_LONGINT($MovieYear)

    $MovieYear:=1979
    $tQueryTxt:=""
    $tQueryTxt:=$tQueryTxt+"SELECT Year_of_Movie, SUM(Sold_Tickets)"
    $tQueryTxt:=$tQueryTxt+" FROM MOVIES"
    $tQueryTxt:=$tQueryTxt+" WHERE Year_of_Movie >= :$MovieYear"
    $tQueryTxt:=$tQueryTxt+" GROUP BY Year_of_Movie"
    $tQueryTxt:=$tQueryTxt+" ORDER BY 1"
    $tQueryTxt:=$tQueryTxt+" INTO :aMovieYear, :aSoldTickets;"
    Begin SQL
        EXECUTE IMMEDIATE :$tQueryTxt;
    End SQL
    ` Inicializa el resto de las columnas del list box para mostrar la información
    ARRAY TEXT(aTitles;Size of array(aMovieYear))
    ARRAY TEXT(aDirectors;Size of array(aMovieYear))
    ARRAY TEXT(aMedias;Size of array(aMovieYear))
    ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

Para probar todos los ejemplos anteriores, lance la base "4D SQL Code Samples" y vaya a la ventana principal. A continuación, puede elegir el modo de búsqueda y presionar el botón **GROUP BY clause**.

Utilizar funciones estadísticas

Las funciones estadísticas permiten efectuar cálculos en una serie de valores. SQL contiene numerosas funciones de agregación **MIN**, **MAX**, *AVG*, **SUM**, etc. Utilizando las funciones de agregación, queremos obtener información sobre el número de boletos vendidos cada año. El resultado se ordenará por año.
Para hacer esto, debemos efectuar la suma de todos los boletos vendidos para cada película y luego ordenarlos por año.

- El código 4D inicial sería:

```
C_LONGINT($vMin;$vMax;$vSum)
C_REAL($vAverage)
C_TEXT($AlertTxt)

REDUCE SELECTION([MOVIES];0)
$vMin:=0
$vMax:=0
$vAverage:=0
$vSum:=0
ALL RECORDS([MOVIES])
$vMin:=Min([MOVIES]Sold_Tickets)
$vMax:=Max([MOVIES]Sold_Tickets)
$vAverage:=Average([MOVIES]Sold_Tickets)
$vSum:=Sum([MOVIES]Sold_Tickets)
 `$AlertTxt:=""
 `$AlertTxt:=$AlertTxt+"Minimum tickets sold: "+Chaine($vMin)+Caractere(13)
 `$AlertTxt:=$AlertTxt+"Maximum tickets sold: "+Chaine($vMax)+Caractere(13)
 `$AlertTxt:=$AlertTxt+"Average tickets sold: "+Chaine($vAverage)+Caractere(13)
 `$AlertTxt:=$AlertTxt+"Total tickets sold: "+Chaine($vSum)+Caractere(13)
```

- Utilizando código SQL:

```
C_LONGINT($vMin;$vMax;$vSum)
C_REAL($vAverage)
C_TEXT($AlertTxt)

$vMin:=0
$vMax:=0
$vAverage:=0
$vSum:=0
Begin SQL
   SELECT MIN(Sold_Tickets),
   MAX(Sold_Tickets),
   AVG(Sold_Tickets),
   SUM(Sold_Tickets)
   FROM MOVIES
   INTO :$vMin, :$vMax, :$vAverage, :$vSum;
```

```
End SQL
  `$AlertTxt:=""
  `$AlertTxt:=$AlertTxt+"Minimum tickets sold: "+Chaine($vMin)+Caractere(13)
  `$AlertTxt:=$AlertTxt+"Maximum tickets sold: "+Chaine($vMax)+Caractere(13)
  `$AlertTxt:=$AlertTxt+"Average tickets sold: "+Chaine($vAverage)+Caractere(13)
  `$AlertTxt:=$AlertTxt+"Total tickets sold: "+Chaine($vSum)+Caractere(13)
  `ALERT($AlertTxt)
```

- Utilizando comandos SQL genéricos:

```
C_LONGINT($vMin;$vMax;$vSum)
C_REAL($vAverage)
C_TEXT($tQueryTxt)
C_TEXT($AlertTxt)

$vMin:=0
$vMax:=0
$vAverage:=0
$vSum:=0
SQL LOGIN(SQL_INTERNAL;"";"")
$tQueryTxt:=""
$tQueryTxt:=$tQueryTxt+"SELECT MIN(Sold_Tickets), MAX(Sold_Tickets), AVG(Sold_Tickets),
SUM(Sold_Tickets)"
$tQueryTxt:=$tQueryTxt+" FROM MOVIES"
SQL EXECUTE($tQueryTxt;$vMin;$vMax;$vAverage;$vSum)
SQL LOAD RECORD(SQL all records)
SQL LOGOUT
$AlertTxt:=""
  `$AlertTxt:=$AlertTxt+"Minimum tickets sold: "+Chaine($vMin)+Caractere(13)
  `$AlertTxt:=$AlertTxt+"Maximum tickets sold: "+Chaine($vMax)+Caractere(13)
  `$AlertTxt:=$AlertTxt+"Average tickets sold: "+Chaine($vAverage)+Caractere(13)
  `$AlertTxt:=$AlertTxt+"Total tickets sold: "+Chaine($vSum)+Caractere(13)
  `ALERT($AlertTxt)
```

- Utilizando el comando *EXECUTE IMMEDIATE*:

```
C_LONGINT($vMin;$vMax;$vSum)
C_REAL($vAverage)
C_TEXT($tQueryTxt)
C_TEXT($AlertTxt)

$vMin:=0
$vMax:=0
$vAverage:=0
$vSum:=0
$tQueryTxt:=""
$tQueryTxt:=$tQueryTxt+"SELECT MIN(Sold_Tickets), MAX(Sold_Tickets), AVG(Sold_Tickets),
SUM(Sold_Tickets)"
$tQueryTxt:=$tQueryTxt+" FROM MOVIES"
$tQueryTxt:=$tQueryTxt+" INTO :$vMin, :$vMax, :$vAverage, :$vSum;"
Begin SQL
    EXECUTE IMMEDIATE :$tQueryTxt;
```

```
End SQL
`$AlertTxt:=""
`$AlertTxt:=$AlertTxt+"Minimum tickets sold: "+Chaine($vMin)+Caractere(13)
`$AlertTxt:=$AlertTxt+"Maximum tickets sold: "+Chaine($vMax)+Caractere(13)
`$AlertTxt:=$AlertTxt+"Average tickets sold: "+Chaine($vAverage)+Caractere(13)
`$AlertTxt:=$AlertTxt+"Total tickets sold: "+Chaine($vSum)+Caractere(13)
`ALERT($AlertTxt)
```

Para probar todos los ejemplos anteriores, lance la base "4D SQL Code Samples" y vaya a la ventana principal. A continuación, puede elegir el modo de búsqueda y presionar el botón **Using Aggregate functions**.

Utilizar la cláusula HAVING

Ahora queremos obtener el número total de entradas vendidas por año a partir de 1979, sin incluir las películas con más de 10,000,000 boletos vendidos. El resultado se ordenará por año.
Para hacer esto, debemos sumar el total de boletos vendidos para cada películas desde 1979, eliminar las entradas cuyo total de boletos vendidos sea mayor a 10,000,000, y luego ordenar el resultado por año.

- El código 4D inicial sería:

```
ARRAY INTEGER(aMovieYear;0)
ARRAY LONGINT(aSoldTickets;0)
C_LONGINT($MovieYear;$vCrtMovieYear;$i;$MinSoldTickets;$vInd)

REDUCE SELECTION([MOVIES];0)
$MovieYear:=1979
$MinSoldTickets:=10000000
QUERY([MOVIES];[MOVIES]Year_of_Movie>=$MovieYear)
ORDER BY([MOVIES];[MOVIES]Year_of_Movie;>)
$vCrtMovieYear:=0
$vInd:=Size of array(aMovieYear)
For($i;1;Records in selection([MOVIES]))
    If([MOVIES]Year_of_Movie#$vCrtMovieYear)
        $vCrtMovieYear:=[MOVIES]Year_of_Movie
        If(aSoldTickets{$vInd}<$MinSoldTickets)
            $vInd:=$vInd+1
            INSERT IN ARRAY(aMovieYear;$vInd;1)
            aMovieYear{$vInd}:=$vCrtMovieYear
            INSERT IN ARRAY(aSoldTickets;$vInd;1)
        Else
            aSoldTickets{$vInd}:=0
        End if
    End if
    aSoldTickets{$vInd}:=aSoldTickets{$vInd}+[MOVIES]Sold_Tickets
    NEXT RECORD([MOVIES])
End for
If(aSoldTickets{$vInd}>=$MinSoldTickets)
    DELETE FROM ARRAY(aSoldTickets;$vInd;1)
    DELETE FROM ARRAY(aMovieYear;$vInd;1)
End if
  ` Inicializa el resto de las columnas para mostrar la información
ARRAY TEXT(aTitles;Size of array(aMovieYear))
ARRAY TEXT(aDirectors;Size of array(aMovieYear))
ARRAY TEXT(aMedias;Size of array(aMovieYear))
ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

- Utilizando código SQL:

```
ARRAY INTEGER(aMovieYear;0)
ARRAY LONGINT(aSoldTickets;0)
C_LONGINT($MovieYear;$MinSoldTickets)

$MovieYear:=1979
$MinSoldTickets:=10000000
Begin SQL
    SELECT Year_of_Movie, SUM(Sold_Tickets)
    FROM MOVIES
    WHERE Year_of_Movie >= :$MovieYear
    GROUP BY Year_of_Movie
    HAVING SUM(Sold_Tickets) < :$MinSoldTickets
    ORDER BY 1
    INTO :aMovieYear,   :aSoldTickets;
End SQL
  ` Inicializa el resto de las columnas para mostrar la información
ARRAY TEXT(aTitles;Size of array(aMovieYear))
ARRAY TEXT(aDirectors;Size of array(aMovieYear))
ARRAY TEXT(aMedias;Size of array(aMovieYear))
ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

- Utilizando comandos SQL genéricos:

```
C_TEXT($tQueryTxt)
ARRAY INTEGER(aMovieYear;0)
ARRAY LONGINT(aSoldTickets;0)
C_LONGINT($MovieYear;$MinSoldTickets)

$MovieYear:=1979
$MinSoldTickets:=10000000
SQL LOGIN(SQL_INTERNAL;"";"")
$tQueryTxt:=""
$tQueryTxt:=$tQueryTxt+"SELECT Year_of_Movie, SUM(Sold_Tickets)"
$tQueryTxt:=$tQueryTxt+" FROM MOVIES"
$tQueryTxt:=$tQueryTxt+" WHERE Year_of_Movie >= :$MovieYear"
$tQueryTxt:=$tQueryTxt+" GROUP BY Year_of_Movie"
$tQueryTxt:=$tQueryTxt+" HAVING SUM(Sold_Tickets) < :$MinSoldTickets"
$tQueryTxt:=$tQueryTxt+" ORDER BY 1"
SQL EXECUTE($tQueryTxt;aMovieYear;aSoldTickets)
SQL LOAD RECORD(SQL all records)
SQL LOGOUT
  ` Inicializa el resto de las columnas para mostrar la información
ARRAY TEXT(aTitles;Size of array(aMovieYear))
ARRAY TEXT(aDirectors;Size of array(aMovieYear))
ARRAY TEXT(aMedias;Size of array(aMovieYear))
ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

- Utilizando el comando **QUERY BY SQL**:

```
C_TEXT($tQueryTxt)
ARRAY INTEGER(aMovieYear;0)
```

```
ARRAY LONGINT(aSoldTickets;0)
C_LONGINT($MovieYear;$MinSoldTickets;$vCrtMovieYear;$vInd;$i)

REDUCE SELECTION([MOVIES];0)
$MovieYear:=1979
$MinSoldTickets:=10000000
QUERY BY SQL([MOVIES];"Year_of_Movie >= :$MovieYear")
ORDER BY([MOVIES];[MOVIES]Year_of_Movie;>)
$vCrtMovieYear:=0
$vInd:=Size of array(aMovieYear)
For($i;1;Records in selection([MOVIES]))
   If([MOVIES]Year_of_Movie#$vCrtMovieYear)
      $vCrtMovieYear:=[MOVIES]Year_of_Movie
      If(aSoldTickets{$vInd}<$MinSoldTickets)
         $vInd:=$vInd+1
         INSERT IN ARRAY(aMovieYear;$vInd;1)
         aMovieYear{$vInd}:=$vCrtMovieYear
         INSERT IN ARRAY(aSoldTickets;$vInd;1)
      Else
         aSoldTickets{$vInd}:=0
      End if
   End if
   aSoldTickets{$vInd}:=aSoldTickets{$vInd}+[MOVIES]Sold_Tickets
   NEXT RECORD([MOVIES])
End for
If(aSoldTickets{$vInd}>=$MinSoldTickets)
   DELETE FROM ARRAY(aSoldTickets;$vInd;1)
   DELETE FROM ARRAY(aMovieYear;$vInd;1)
End if
  ` Inicializa el resto de las columnas para mostrar la información
ARRAY TEXT(aTitles;Size of array(aMovieYear))
ARRAY TEXT(aDirectors;Size of array(aMovieYear))
ARRAY TEXT(aMedias;Size of array(aMovieYear))
ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

- Utilizando el comando SQL *EXECUTE IMMEDIATE*:

```
C_TEXT($tQueryTxt)
ARRAY INTEGER(aMovieYear;0)
ARRAY LONGINT(aSoldTickets;0)
C_LONGINT($MovieYear;$MinSoldTickets)

$MovieYear:=1979
$MinSoldTickets:=10000000
$tQueryTxt:=""
$tQueryTxt:=$tQueryTxt+"SELECT Year_of_Movie, SUM(Sold_Tickets)"
$tQueryTxt:=$tQueryTxt+" FROM MOVIES"
$tQueryTxt:=$tQueryTxt+" WHERE Year_of_Movie >= :$MovieYear"
$tQueryTxt:=$tQueryTxt+" GROUP BY Year_of_Movie"
$tQueryTxt:=$tQueryTxt+" HAVING SUM(Sold_Tickets) < :$MinSoldTickets"
$tQueryTxt:=$tQueryTxt+" ORDER BY 1"
$tQueryTxt:=$tQueryTxt+" INTO :aMovieYear, :aSoldTickets;"
Begin SQL
   EXECUTE IMMEDIATE :$tQueryTxt;
```

```
End SQL
  ` Inicializa el resto de las columnas para mostrar la información
ARRAY TEXT(aTitles;Size of array(aMovieYear))
ARRAY TEXT(aDirectors;Size of array(aMovieYear))
ARRAY TEXT(aMedias;Size of array(aMovieYear))
ARRAY LONGINT(aNrActors;Size of array(aMovieYear))
```

Para probar todos los ejemplos anteriores, lance la base "4D SQL Code Samples" y vaya a la ventana principal. A continuación, puede elegir el modo de búsqueda y presionar el botón **HAVING clause**.

Llamar los métodos 4D desde el código SQL

Ahora queremos efectuar búsquedas relativas a los actores de cada película: más especificamente, estamos interesados en encontrar todas las películas con al menos 7 actores. El resultado se ordenará por año.
Para hacer esto, utilizamos una función 4D (Find_Nr_Of_Actors) que recibe el ID de la película y devuelve el número de actores de la película:

```
  `(F) Find_Nr_Of_Actors
C_LONGINT($0;$1;$vMovie_ID)
$vMovie_ID:=$1

QUERY([MOVIE_ACTOR];[MOVIE_ACTOR]Movie_ID=$vMovie_ID)
$0:=Records in selection([MOVIE_ACTOR])
```

- El código 4D inicial sería:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY LONGINT(aNrActors;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)
C_LONGINT($NrOfActors;$i;$vInd)

$vInd:=0
$NrOfActors:=7
ALL RECORDS([MOVIES])
For($i;1;Records in selection([MOVIES]))
   $vCrtActors:=Find_Nr_Of_Actors([MOVIES]ID)
   If($vCrtActors>=$NrOfActors)
      $vInd:=$vInd+1
      INSERT IN ARRAY(aMovieYear;$vInd;1)
      aMovieYear{$vInd}:=[MOVIES]Year_of_Movie
      INSERT IN ARRAY(aTitles;$vInd;1)
      aTitles{$vInd}:=[MOVIES]Title
      INSERT IN ARRAY(aDirectors;$vInd;1)
      aDirectors{$vInd}:=[MOVIES]Director
      INSERT IN ARRAY(aMedias;$vInd;1)
      aMedias{$vInd}:=[MOVIES]Media
      INSERT IN ARRAY(aSoldTickets;$vInd;1)
      aSoldTickets{$vInd}:=[MOVIES]Sold_Tickets
      INSERT IN ARRAY(aNrActors;$vInd;1)
      aNrActors{$vInd}:=$vCrtActors
   End if
   NEXT RECORD([MOVIES])
End for
SORT ARRAY(aMovieYear;aTitles;aDirectors;aMedias;aSoldTickets;aNrActors;>)
```

- Utilizando código SQL:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY LONGINT(aNrActors;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)
C_LONGINT($NrOfActors;$i;$vInd)

$vInd:=0
$NrOfActors:=7
Begin SQL
    SELECT Year_of_Movie, Title, Director, Media, Sold_Tickets, {fn Find_Nr_Of_Actors(ID) AS
NUMERIC}
    FROM MOVIES
    WHERE {fn Find_Nr_Of_Actors(ID) AS NUMERIC} >= :$NrOfActors
    ORDER BY 1
    INTO :aMovieYear, :aTitles, :aDirectors, :aMedias, :aSoldTickets, :aNrActors;
End SQL
```

Puede ver que podemos llamar a un función 4D dentro del código SQL utilizando la sintaxis:

```
{fn 4DFunctionName AS 4DFunctionResultType}
```

- Utilizando comandos SQL genéricos:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY LONGINT(aNrActors;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)
C_LONGINT($NrOfActors;$i;$vInd)
C_TEXT($tQueryTxt)

$vInd:=0
$NrOfActors:=7
SQL LOGIN(SQL_INTERNAL;"";"")
$tQueryTxt:=""
$tQueryTxt:=$tQueryTxt+"SELECT Year_of_Movie, Title, Director, Media, Sold_Tickets, {fn
Find_Nr_Of_Actors(ID) AS NUMERIC}"
$tQueryTxt:=$tQueryTxt+" FROM MOVIES"
$tQueryTxt:=$tQueryTxt+" WHERE {fn Find_Nr_Of_Actors(ID) AS NUMERIC} >= :$NrOfActors"
$tQueryTxt:=$tQueryTxt+" ORDER BY 1"
SQL EXECUTE($tQueryTxt;aMovieYear;aTitles;aDirectors;aMedias;aSoldTickets;aNrActors)
SQL LOAD RECORD(SQL all records)
SQL LOGOUT
```

- Utilizando el comando **QUERY BY SQL**:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY LONGINT(aNrActors;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)
C_LONGINT($NrOfActors;$i;$vInd)

$vInd:=0
$NrOfActors:=7
QUERY BY SQL([MOVIES];"{fn Find_Nr_Of_Actors(ID) AS NUMERIC} >= :$NrOfActors")
For($i;1;Records in selection([MOVIES]))
   $vInd:=$vInd+1
   INSERT IN ARRAY(aMovieYear;$vInd;1)
   aMovieYear{$vInd}:=[MOVIES]Year_of_Movie
   INSERT IN ARRAY(aTitles;$vInd;1)
   aTitles{$vInd}:=[MOVIES]Title
   INSERT IN ARRAY(aDirectors;$vInd;1)
   aDirectors{$vInd}:=[MOVIES]Director
   INSERT IN ARRAY(aMedias;$vInd;1)
   aMedias{$vInd}:=[MOVIES]Media
   INSERT IN ARRAY(aSoldTickets;$vInd;1)
   aSoldTickets{$vInd}:=[MOVIES]Sold_Tickets
   INSERT IN ARRAY(aNrActors;$vInd;1)
   aNrActors{$vInd}:=Find_Nr_Of_Actors([MOVIES]ID)
   NEXT RECORD([MOVIES])
End for
SORT ARRAY(aMovieYear;aTitles;aDirectors;aMedias;aSoldTickets;aNrActors;>)
```

- Utilizando el comando SQL *EXECUTE IMMEDIATE*:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
ARRAY LONGINT(aNrActors;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
ARRAY TEXT(aMedias;0)
C_LONGINT($NrOfActors;$i;$vInd)
C_TEXT($tQueryTxt)

$vInd:=0
$NrOfActors:=7
$tQueryTxt:=""
$tQueryTxt:=$tQueryTxt+"SELECT Year_of_Movie, Title, Director, Media, Sold_Tickets, {fn
Find_Nr_Of_Actors(ID) AS NUMERIC}"
$tQueryTxt:=$tQueryTxt+" FROM MOVIES"
$tQueryTxt:=$tQueryTxt+" WHERE {fn Find_Nr_Of_Actors(ID) AS NUMERIC} >= :$NrOfActors"
$tQueryTxt:=$tQueryTxt+" ORDER BY 1"
$tQueryTxt:=$tQueryTxt+" INTO :aMovieYear, :aTitles, :aDirectors, :aMedias, :aSoldTickets,"+"
:aNrActors;"
Begin SQL
```

```
    EXECUTE IMMEDIATE :$tQueryTxt;
End SQL
```

Para probar todos los ejemplos anteriores, lance la base "4D SQL Code Samples" y vaya a la ventana principal. A continuación, puede elegir el modo de búsqueda y presionar el botón **Calling 4D methods**.

Joins

Ahora queremos conocer la ciudad de nacimiento de cada actor. La lista de actores está en la tabla ACTORS y la lista de ciudades en la tabla CITIES. Para ejecutar esta búsqueda necesitamos unir las dos tablas: ACTORS y CITIES.

- El código 4D inicial sería:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
C_LONGINT($i;$vlnd)

$vlnd:=0
ALL RECORDS([ACTORS])
For($i;1;Records in selection([ACTORS]))
    $vlnd:=$vlnd+1
    INSERT IN ARRAY(aTitles;$vlnd;1)
    aTitles{$vlnd}:=[ACTORS]FirstName+" "+[ACTORS]LastName
    RELATE ONE([ACTORS]Birth_City_ID)
    INSERT IN ARRAY(aDirectors;$vlnd;1)
    aDirectors{$vlnd}:=[CITIES]City_Name
    NEXT RECORD([ACTORS])
End for
  ` Inicializa el resto de las columnas del list box para mostrar la información
ARRAY INTEGER(aMovieYear;Size of array(aTitles))
ARRAY TEXT(aMedias;Size of array(aTitles))
ARRAY LONGINT(aSoldTickets;Size of array(aTitles))
ARRAY LONGINT(aNrActors;Size of array(aTitles))
MULTI SORT ARRAY(aDirectors;>;aTitles;>;aMovieYear;aMedias;aSoldTickets;aNrActors)
```

- Utilizando código SQL:

```
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)

Begin SQL
    SELECT CONCAT(CONCAT(ACTORS.FirstName,' '),ACTORS.LastName), CITIES.City_Name
    FROM ACTORS, CITIES
    WHERE ACTORS.Birth_City_ID=CITIES.City_ID
    ORDER BY 2,1
    INTO :aTitles, :aDirectors;
End SQL
  ` Inicializa el resto de las columnas del list box para mostrar la información
ARRAY INTEGER(aMovieYear;Size of array(aTitles))
ARRAY TEXT(aMedias;Size of array(aTitles))
ARRAY LONGINT(aSoldTickets;Size of array(aTitles))
ARRAY LONGINT(aNrActors;Size of array(aTitles))
```

- Utilizando comandos SQL genéricos:

```
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
C_TEXT($tQueryTxt)

SQL LOGIN(SQL_INTERNAL;"";"")
$tQueryTxt:=""
$tQueryTxt:=$tQueryTxt+"SELECT CONCAT(CONCAT(ACTORS.FirstName,' '),ACTORS.LastName),
CITIES.City_Name"
$tQueryTxt:=$tQueryTxt+" FROM ACTORS, CITIES"
$tQueryTxt:=$tQueryTxt+" WHERE ACTORS.Birth_City_ID=CITIES.City_ID"
$tQueryTxt:=$tQueryTxt+" ORDER BY 2,1"
SQL EXECUTE($tQueryTxt;aTitles;aDirectors)
SQL LOAD RECORD(SQL all records)
SQL LOGOUT
  ` Inicializa el resto de las columnas del list box para mostrar la información
ARRAY INTEGER(aMovieYear;Size of array(aTitles))
ARRAY TEXT(aMedias;Size of array(aTitles))
ARRAY LONGINT(aSoldTickets;Size of array(aTitles))
ARRAY LONGINT(aNrActors;Size of array(aTitles))
```

- Utilizando el comando **QUERY BY SQL**, no podemos efectuar la búsqueda anterior porque no es posible pasar más de una tabla como primer parámetro.
- Utilizando el comando SQL *EXECUTE IMMEDIATE*:

```
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
C_TEXT($tQueryTxt)

$tQueryTxt:=""
$tQueryTxt:=$tQueryTxt+"SELECT CONCAT(CONCAT(ACTORS.FirstName,' '),ACTORS.LastName),
CITIES.City_Name"
$tQueryTxt:=$tQueryTxt+" FROM ACTORS, CITIES"
$tQueryTxt:=$tQueryTxt+" WHERE ACTORS.Birth_City_ID=CITIES.City_ID"
$tQueryTxt:=$tQueryTxt+" ORDER BY 2,1"
$tQueryTxt:=$tQueryTxt+" INTO :aTitles, :aDirectors"
Begin SQL
   EXECUTE IMMEDIATE :$tQueryTxt;
End SQL
  ` Inicializa el resto de las columnas del list box para mostrar la información
ARRAY INTEGER(aMovieYear;Size of array(aTitles))
ARRAY TEXT(aMedias;Size of array(aTitles))
ARRAY LONGINT(aSoldTickets;Size of array(aTitles))
ARRAY LONGINT(aNrActors;Size of array(aTitles))
```

Para probar todos los ejemplos anteriores, lance la base "4D SQL Code Samples" y vaya a la ventana principal. A continuación, puede elegir el modo de búsqueda y presionar el botón **Joins**.

Utilizar Alias

Si una búsqueda SQL es muy compleja y contiene nombres muy largos que dificultan la lectura, es posible utilizar alias para mejorar su legibilidad.
Este es el ejemplo anterior utilizando dos alias: Act para la tabla ACTORS y Cit para la tabla CITIES.

- El código 4D inicial sería:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
C_LONGINT($i;$vlnd)

$vlnd:=0
ALL RECORDS([ACTORS])
For($i;1;Records in selection([ACTORS]))
    $vlnd:=$vlnd+1
    INSERT IN ARRAY(aTitles;$vlnd;1)
    aTitles{$vlnd}:=[ACTORS]FirstName+" "+[ACTORS]LastName
    RELATE ONE([ACTORS]Birth_City_ID)
    INSERT IN ARRAY(aDirectors;$vlnd;1)
    aDirectors{$vlnd}:=[CITIES]City_Name
    NEXT RECORD([ACTORS])
End for
  ` Inicializa el resto de las columnas del list box para mostrar la información
ARRAY INTEGER(aMovieYear;Size of array(aTitles))
ARRAY TEXT(aMedias;Size of array(aTitles))
ARRAY LONGINT(aSoldTickets;Size of array(aTitles))
ARRAY LONGINT(aNrActors;Size of array(aTitles))
MULTI SORT ARRAY(aDirectors;>;aTitles;>;aMovieYear;aMedias;aSoldTickets;aNrActors)
```

- Utilizando código SQL:

```
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)

Begin SQL
    SELECT CONCAT(CONCAT(ACTORS.FirstName,' '),ACTORS.LastName), CITIES.City_Name
    FROM ACTORS AS 'Act', CITIES AS 'Cit'
    WHERE Act.Birth_City_ID=Cit.City_ID
    ORDER BY 2,1
    INTO :aTitles, :aDirectors;
End SQL
  ` Inicializa el resto de las columnas del list box para mostrar la información
ARRAY INTEGER(aMovieYear;Size of array(aTitles))
ARRAY TEXT(aMedias;Size of array(aTitles))
```

```
ARRAY LONGINT(aSoldTickets;Size of array(aTitles))
ARRAY LONGINT(aNrActors;Size of array(aTitles))
```

- Utilizando comandos SQL genéricos:

```
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
C_TEXT($tQueryTxt)

SQL LOGIN(SQL_INTERNAL;"";"")
$tQueryTxt:=""
$tQueryTxt:=$tQueryTxt+"SELECT CONCAT(CONCAT(ACTORS.FirstName,' '),ACTORS.LastName),
CITIES.City_Name"
$tQueryTxt:=$tQueryTxt+" FROM ACTORS AS 'Act', CITIES AS 'Cit'"
$tQueryTxt:=$tQueryTxt+" WHERE Act.Birth_City_ID=Cit.City_ID"
$tQueryTxt:=$tQueryTxt+" ORDER BY 2,1"
SQL EXECUTE($tQueryTxt;aTitles;aDirectors)
SQL LOAD RECORD(SQL all records)
SQL LOGOUT
  ` Inicializa el resto de las columnas del list box para mostrar la información
ARRAY INTEGER(aMovieYear;Size of array(aTitles))
ARRAY TEXT(aMedias;Size of array(aTitles))
ARRAY LONGINT(aSoldTickets;Size of array(aTitles))
ARRAY LONGINT(aNrActors;Size of array(aTitles))
```

- Utilizando el comando **QUERY BY SQL**, no podemos efectuar la búsqueda anterior porque no es posible pasar másde una tabla como primer parámetro.
- Utilizando el comando SQL *EXECUTE IMMEDIATE*:

```
ARRAY TEXT(aTitles;0)
ARRAY TEXT(aDirectors;0)
C_TEXT($tQueryTxt)
$tQueryTxt:=""
$tQueryTxt:=$tQueryTxt+"SELECT CONCAT(CONCAT(ACTORS.FirstName,' '),ACTORS.LastName),
CITIES.City_Name"
$tQueryTxt:=$tQueryTxt+" FROM ACTORS AS 'Act', CITIES AS 'Cit'"
$tQueryTxt:=$tQueryTxt+" WHERE Act.Birth_City_ID=Cit.City_ID"
$tQueryTxt:=$tQueryTxt+" ORDER BY 2,1"
$tQueryTxt:=$tQueryTxt+" INTO :aTitles, :aDirectors"
Begin SQL
    EXECUTE IMMEDIATE :$tQueryTxt;
End SQL
ARRAY INTEGER(aMovieYear;Size of array(aTitles))
ARRAY TEXT(aMedias;Size of array(aTitles))
ARRAY LONGINT(aSoldTickets;Size of array(aTitles))
ARRAY LONGINT(aNrActors;Size of array(aTitles))
```

Para probar todos los ejemplos anteriores, lance la base "4D SQL Code Samples" y vaya a la ventana principal. Elija el modo de consulta y presione el botón **Using Aliases**.

Subconsultas

Ahora queremos obtener una información estadística sobre los boletos vendidos: cuáles son las películas cuyos boletos vendidos son superiores al promedio de boletos vendidos para todas las películas. Para ejecutar esta búsqueda en SQL, utilizaremos una búsqueda en una búsqueda, en otras palabras una subconsulta.

- El código 4D inicial sería:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY TEXT(aTitles;0)
C_LONGINT($i;$vInd;$vAvgSoldTickets)

$vInd:=0
ALL RECORDS([MOVIES])
$vAvgSoldTickets:=Average([MOVIES]Sold_Tickets)
For($i;1;Records in selection([MOVIES]))
    If([MOVIES]Sold_Tickets>$vAvgSoldTickets)
        $vInd:=$vInd+1
        INSERT IN ARRAY(aTitles;$vInd;1)
        aTitles{$vInd}:=[MOVIES]Title
        INSERT IN ARRAY(aSoldTickets;$vInd;1)
        aSoldTickets{$vInd}:=[MOVIES]Sold_Tickets
    End if
    NEXT RECORD([MOVIES])
End for
 ` Inicializa el resto de las columnas del list box con el fin de visualizar la información
ARRAY INTEGER(aMovieYear;Size of array(aTitles))
ARRAY TEXT(aDirectors;Size of array(aTitles))
ARRAY TEXT(aMedias;Size of array(aTitles))
ARRAY LONGINT(aNrActors;Size of array(aTitles))
SORT ARRAY(aTitles;aDirectors;aMovieYear;aMedias;aSoldTickets;aNrActors;>)
```

- Utilizando código SQL:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY TEXT(aTitles;0)
Begin SQL
    SELECT Title, Sold_Tickets
    FROM MOVIES
    WHERE Sold_Tickets > (SELECT AVG(Sold_Tickets) FROM MOVIES)
    ORDER BY 1
    INTO :aTitles, :aSoldTickets;
End SQL
 ` Inicializa el resto de las columnas del list box con el fin de visualizar la información
ARRAY INTEGER(aMovieYear;Size of array(aTitles))
ARRAY TEXT(aDirectors;Size of array(aTitles))
ARRAY TEXT(aMedias;Size of array(aTitles))
```

```
ARRAY LONGINT(aNrActors;Size of array(aTitles))
SORT ARRAY(aTitles;aDirectors;aMovieYear;aMedias;aSoldTickets;aNrActors;>)
```

- Utilizando comandos SQL genéricos:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY TEXT(aTitles;0)
C_TEXT($tQueryTxt)

SQL LOGIN(SQL_INTERNAL;"";"")
$tQueryTxt:=""
$tQueryTxt:=$tQueryTxt+"SELECT Title, Sold_Tickets"
$tQueryTxt:=$tQueryTxt+" FROM MOVIES"
$tQueryTxt:=$tQueryTxt+" WHERE Sold_Tickets > (SELECT AVG(Sold_Tickets) FROM MOVIES)"
$tQueryTxt:=$tQueryTxt+" ORDER BY 1"
SQL EXECUTE($tQueryTxt;aTitles;aSoldTickets)
SQL LOAD RECORD(SQL all records)
SQL LOGOUT
  ` Inicializa el resto de las columnas del list box con el fin de visualizar la información
ARRAY INTEGER(aMovieYear;Size of array(aTitles))
ARRAY TEXT(aDirectors;Size of array(aTitles))
ARRAY TEXT(aMedias;Size of array(aTitles))
ARRAY LONGINT(aNrActors;Size of array(aTitles))
SORT ARRAY(aTitles;aDirectors;aMovieYear;aMedias;aSoldTickets;aNrActors;>)
```

- Utilizando el comando **QUERY BY SQL**:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY TEXT(aTitles;0)

QUERY BY SQL([MOVIES];"Sold_Tickets > (SELECT AVG(Sold_Tickets) FROM MOVIES)")
ORDER BY([MOVIES];[MOVIES]Title;>)
SELECTION TO ARRAY([MOVIES]Title;aTitles;[MOVIES]Sold_Tickets;aSoldTickets)
  ` Inicializa el resto de las columnas del list box con el fin de visualizar la información
ARRAY INTEGER(aMovieYear;Size of array(aTitles))
ARRAY TEXT(aDirectors;Size of array(aTitles))
ARRAY TEXT(aMedias;Size of array(aTitles))
ARRAY LONGINT(aNrActors;Size of array(aTitles))
SORT ARRAY(aTitles;aDirectors;aMovieYear;aMedias;aSoldTickets;aNrActors;>)
```

- Utilizando el comando SQL *EXECUTE IMMEDIATE*:

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY TEXT(aTitles;0)
C_TEXT($tQueryTxt)

$tQueryTxt:=""
$tQueryTxt:=$tQueryTxt+"SELECT Title, Sold_Tickets"
$tQueryTxt:=$tQueryTxt+" FROM MOVIES"
$tQueryTxt:=$tQueryTxt+" WHERE Sold_Tickets > (SELECT AVG(Sold_Tickets) FROM MOVIES)"
$tQueryTxt:=$tQueryTxt+" ORDER BY 1"
```

```
$tQueryTxt:=$tQueryTxt+" INTO :aTitles, :aSoldTickets"
Begin SQL
   EXECUTE IMMEDIATE :$tQueryTxt;
End SQL
  ` Inicializa el resto de las columnas del list box con el fin de visualizar la información
ARRAY INTEGER(aMovieYear;Size of array(aTitles))
ARRAY TEXT(aDirectors;Size of array(aTitles))
ARRAY TEXT(aMedias;Size of array(aTitles))
ARRAY LONGINT(aNrActors;Size of array(aTitles))
```

Para probar todos los ejemplos anteriores, lance la base "4D SQL Code Samples" y vaya a la ventana principal. Elija el modo de consulta y presione el botón **Subqueries**.

Seguimiento y depuración del código SQL

En 4D, hay dos posibilidades principales para la localización y corrección de su código: o bien utilizando el para localizar y corregir cualquier error o llamando al comando **ON ERR CALL** para capturar el error y tomar las medidas adecuadas. Podemos utilizar estas dos técnicas para resolver los problemas encontrados con el código SQL.
Este es un ejemplo donde falta un paréntesis intecionalmente: en lugar de **HAVING SUM(Sold_Tickets <:$MinSoldTickets)**, tenemos **HAVING SUM(Sold_Tickets <:$MinSoldTickets**.

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
C_LONGINT($MovieYear;$MinSoldTickets)
$MovieYear:=1979
$MinSoldTickets:=10000000

Begin SQL
    SELECT Year_of_Movie, SUM(Sold_Tickets)
    FROM MOVIES
    WHERE Year_of_Movie >= :$MovieYear
    GROUP BY Year_of_Movie
    HAVING SUM(Sold_Tickets < :$MinSoldTickets
    ORDER BY 1
    INTO :aMovieYear,   :aSoldTickets;
End SQL
```

Como puede ver en la ventana inferior, la aplicación detecta el error y abre la que ofrece información más detallada sobre el error y el lugar donde ocurrió. Es entonces fácil de corregir con sólo presionar el botón **Editar**.

Si el error es más complejo, la aplicación ofrece más información incluyendo el contenido de la pila, que se puede ver presionando el botón **Detalles**.
Para probar el ejemplo anterior, en la ventana principal de la base "4D SQL Code Samples", presione el botón **Depurar el código SQL**.
La segunda posibilidad principal para el seguimiento de errores SQL utiliza el comando **ON ERR CALL**.
Este es un ejemplo que instala el método SQL_Error_Handler para detectar errores encontrados en el código SQL.

```
ARRAY LONGINT(aSoldTickets;0)
ARRAY INTEGER(aMovieYear;0)
C_LONGINT($MovieYear;$MinSoldTickets;MySQL_Error)
$MovieYear:=1979
$MinSoldTickets:=10000000
MySQL_Error:=0

  ` Instalación del método SQL_Error_Handler para interceptar los errores
ON ERR CALL("SQL_Error_Handler")
```

```
Begin SQL
    SELECT Year_of_Movie, SUM(Sold_Tickets)
    FROM MOVIES
    WHERE Year_of_Movie >= :$MovieYear
    GROUP BY Year_of_Movie
    HAVING SUM(Sold_Tickets < :$MinSoldTickets
    ORDER BY 1
    INTO :aMovieYear,   :aSoldTickets;
End SQL
  ` Desinstalación del método SQL_Error_Handler
ON ERR CALL("")
```

El método SQL_Error_Handler es el siguiente:

```
If(MySQL_Error#0)
    ALERT("SQL Error number: "+String(MySQL_Error))
End if
```

Para probar el ejemplo anterior, en la ventana principal de de la base "4D SQL Code Samples" presione el botón **Using ON ERR CALL**.

Data Definition Language

Usando el SQL Data Definition Language (DDL), puede definir y gestionar la estructura de la base. Con los comandos DDL, puede crear o modificar tablas y campos, así como agregar y/o eliminar datos.

Este es un ejemplo simple que crea una tabla, añade algunos campos y luego llena los campos con algunos datos.

```
Begin SQL
    DROP TABLE IF EXISTS ACTOR_FANS;

    CREATE TABLE ACTOR_FANS

    (ID INT32,
    Name VARCHAR);

    INSERT INTO ACTOR_FANS
    (ID, Name)
    VALUES(1, 'Francis');

    ALTER TABLE ACTOR_FANS
    ADD Phone_Number VARCHAR;

    INSERT INTO ACTOR_FANS
    (ID, Name, Phone_Number)
    VALUES (2, 'Florence', '01446677888');

End SQL
```

Para probar el ejemplo anterior, en la ventana principal de la base "4D SQL Code Samples", presione el botón DDL.

Nota: este ejemplo sólo funcionará una vez porque si se presiona el botón "DDL" una segunda vez, obtendrá un mensaje de error diciendo que la tabla ya existe.

Conexiones externas

4D permite el uso de bases de datos externas, es decir ejecutar consultas SQL en otras bases distintas de la base local. Para ello, puede conectarse a cualquier fuente de datos externa vía ODBC o directamente a otras bases 4D.
Estos son los comandos que permiten las conexiones con bases de datos externas:

- **Get current data source** indica la fuente de datos ODBC utilizada por la aplicación.
- **GET DATA SOURCE LIST** devuelve la lista de fuentes de datos ODBC instaladas en la máquina.
- **SQL LOGIN** le permite conectarse a una base externa directamente o vía una fuente de datos ODBC instalada en la máquina.
- **SQL LOGOUT** se puede utilizar para cerrar cualquier conexión externa y volver a conectarse a la base 4D local.
- **USE DATABASE** (comando SQL) se puede utilizar para abrir una base de datos 4D externa con el motor SQL de 4D.

El siguiente ejemplo muestra cómo conectarse a una fuente de datos externa (ORACLE), cómo obtener datos de la base ORACLE, y luego cómo desconectarse de la base ORACLE y volver a la base local.
Supongamos que existe una fuente de datos válida llamada "Test_ORACLE_10g" instalada en el sistema.

```
   ` Get the data sources of the User type defined in the ODBC manager
GET DATA SOURCE LIST(User Data Source ;aDSN;aDS_Driver)
$My_ORACLE_DSN:="Test_Oracle_10g"
If (Find in array(aDSN;$My_ORACLE_DSN)>0)
      ` Establish a connection between 4D and the data source $My_ORACLE_DSN="Test_Oracle_10g"

   SQL LOGIN($My_ORACLE_DSN;"scott";"tiger";*)

      ` The current DSN is the ORACLE one
   $Crt_DSN:=Get current data source
   ALERT("The current DSN is "+$Crt_DSN)
   ARRAY TEXT(aTitles;0)
   ARRAY LONGINT(aNrActors;0)
   ARRAY LONGINT(aSoldTickets;0)
   ARRAY INTEGER(aMovieYear;0)
   ARRAY TEXT(aTitles;0)
   ARRAY TEXT(aDirectors;0)
   ARRAY TEXT(aMedias;0)

      ` Do something on the external (ORACLE) database
   Begin SQL
      SELECT ENAME FROM EMP INTO :aTitles
   End SQL

      ` Close the external connection opened with the SQL LOGIN command
   SQL LOGOUT
      ` The current DSN becomes the local one
   $Crt_DSN:=Get current data source
   ALERT("The current DSN is "+$Crt_DSN)
Else
   ALERT("ORACLE DSN not installed")
End if
```

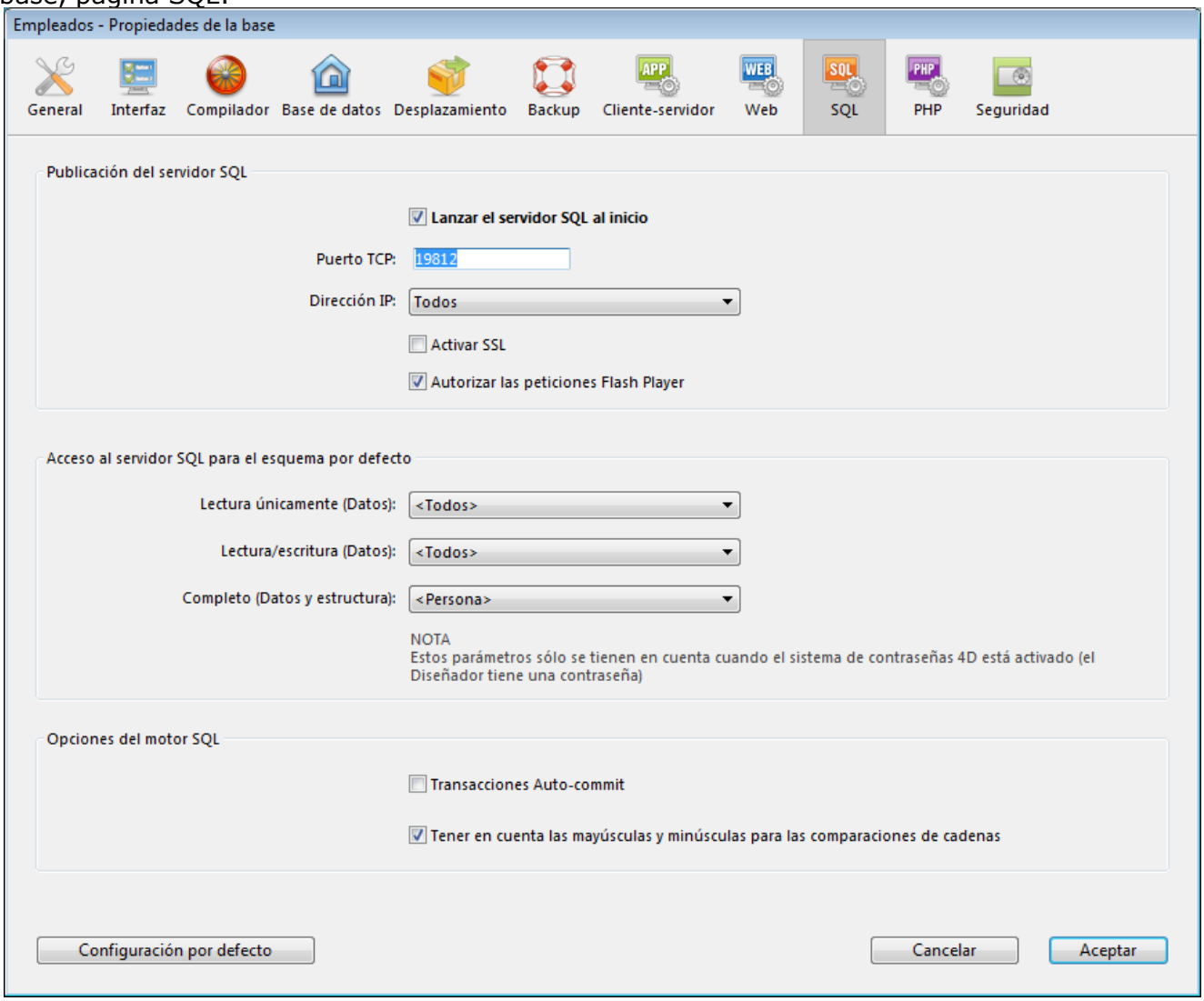
Para probar el ejemplo anterior, en la ventana principal de la base "4D SQL Code Samples", presione el botón **Conectar a ORACLE**.

Conexión al motor SQL 4D vía el driver ODBC

Puede conectarse al motor SQL de 4D desde cualquier base de datos externa vía el driver ODBC para 4D.

Nota: esta configuración se utiliza como un ejemplo. Es posible conectar directamente las aplicaciones 4D entre ellas vía SQL. Para mayor información, consulte la descripción del comando **SQL LOGIN**.

1. Duplique la base de ejemplo que viene con este tutorial
2. Renombre las dos carpetas que contienen las bases, por ejemplo "Cliente" y "Servidor".
3. Lance la base de ejemplo dentro de la carpeta Server y active el lanzamiento del servidor SQL al inicio seleccionando la opción "Lanzar el servidor SQL al inicio" en las Propiedades de la base, página SQL:



4. Cierre y reinicie la base de ejemplo de la carpeta Server para activar el servidor SQL.
5. Instale el driver 4D ODBC Driver for 4D 4D, a continuación, compruebe si aparece en el Administrador de fuentes de datos ODBC:

6. Cree una nueva fuente de datos llamada "Test_ODBC_Driver_v11"

   y pruébela presionando el botón Connection test

7. Lance la base de ejemplo dentro de la carpeta Client, vaya a la ventana principal y haga clic en el botón "Conexión a 4D". El código de este botón es el siguiente:

Como puede ver, en la primera parte del método se hace una consulta en la base local. Luego, en la segunda parte, nos conectamos a la otra base 4D vía el driver ODBC y hacemos la misma consulta. El resultado debe ser el mismo, por supuesto.

Utilizar SQL en 4D

Esta sección ofrece una visión general de las posibilidades de uso de SQL en 4D. Esta sección describe los diferentes modos para acceder al motor SQL integrado, así como las diferentes formas de ejecutar búsquedas y recuperar datos. También detalla la configuración del servidor SQL de 4D y presenta los principios de implementación del SQL a nivel del motor de 4D.

Acceder al motor SQL de 4D

Envío de peticiones al motor SQL de 4D

El motor SQL integrado de 4D puede llamarse de tres maneras diferentes:

- Vía el comando **QUERY BY SQL**. Pase la cláusula **WHERE** de un comando SQL *SELECT* como un parámetro *búsqueda*. Ejemplo:

```
QUERY BY SQL([OFFICES];"SALES > 100")
```

- Vía los comandos SQL integrados de 4D, ubicados en el tema "SQL" (**SQL SET PARAMETER**, **SQL EXECUTE**, etc.). Estos comandos pueden trabajar con una fuente de datos ODBC o el motor SQL de 4D de la base de datos actual.
- Vía el editor de métodos estándar de 4D. Las instrucciones SQL pueden escribirse directamente en el editor de métodos estándar de 4D. Simplemente debe insertar la consulta SQL entre las etiquetas: **Begin SQL** y **End SQL**. El código entre estas etiquetas no será analizado por el intérprete de 4D y será ejecutado por el motor SQL (o por otro motor, si se define por el comando **SQL LOGIN**).

Pasar datos entre 4D y el motor SQL

Referenciar las expresiones 4D

Es posible hacer referencia a todo tipo de expresión 4D válida (variable, campo, array, expresión ...) dentro de las cláusulas **WHERE** e **INTO** de las expresiones SQL. Para indicar una referencia 4D, puede utilizar cualquiera de las siguientes notaciones:

- Poner la referencia entre símbolos dobles menor que y mayor que como se muestra aquí "<<" y ">>"
- Poner dos puntos ":" delante de la referencia.

Ejemplos:

```
C_TEXT(vName)
vName:=Request("Name:")
SQL EXECUTE("SELECT age FROM PEOPLE WHERE name=<<vName>>")
```

o:

```
C_TEXT(vName)
vName:=Request("Name:")
Begin SQL
    SELECT age FROM PEOPLE WHERE name= :vName
End SQL
```

Nota:

- El uso de corchetes [] es necesario cuando trabaja con variables interprocesos (por ejemplo, <<[<>mivar]>> o: [<>mivar]).

Uso de variables en modo compilado

En modo compilado, puede utilizar referencias de variables locales (comenzando por el carácter $) en instrucciones SQL bajo ciertas condiciones:

- Puede utilizar variables locales dentro de una secuencia **Begin SQL** / **End SQL**, excepto con el comando **EXECUTE IMMEDIATE**;
- Puede utilizar variables locales con el comando **SQL EXECUTE** cuando estas variables se utilizan directamente en el parámetro de petición SQL y no vía las referencias.
  Por ejemplo, el siguiente código funciona en modo compilado:

```
SQL EXECUTE("select * from t1 into :$myvar")  // funciona en modo compilado
```

  El siguiente código generará un error en modo compilado:

```
C_TEXT(tRequest)
tRequest:="select * from t1 into :$myvar"
SQL EXECUTE(tRequest)  // error en modo compilado
```

Recuperar los datos de las peticiones SQL en 4D

La recuperación de datos en una instrucción *SELECT* se gestionará dentro de las etiquetas **Begin SQL**/**End SQL** vía la cláusula **INTO** del comando *SELECT* o vía los comandos 4D del tema "SQL".

- En el caso de las etiquetas **Begin SQL**/**End SQL**, puede utilizar la cláusula **INTO** de la consulta SQL y hacer referencia a cualquier expresión válida 4D (campo, variable, array) para obtener el valor:

```
Begin SQL
    SELECT ename FROM emp INTO <<[Employees]Name>>
End SQL
```

- Con el comando **SQL EXECUTE**, también puede utilizar los parámetros adicionales:

```
SQL EXECUTE("SELECT ename FROM emp";[Employees]Name)
```

La principal diferencia entre estas dos maneras de recuperar los datos de una petición SQL (etiquetas **Begin SQL**/**End SQL** y comandos SQL) es que en el primer caso toda la información se devuelve a 4D en un solo paso, mientras que en el segundo caso, los registros deberán cargarse de forma explícita utilizando el comando **SQL LOAD RECORD**.

Por ejemplo, suponiendo que en la tabla PERSONAS hay 100 registros:

- Usando los comandos SQL genéricos de 4D:

```
ARRAY INTEGER(aBirthYear;0)
C_TEXT(vName)
vName:="Smith"
$SQLStm:="SELECT Birth_Year FROM PERSONS WHERE ename= <<vName>>"
SQL EXECUTE($SQLStm;aBirthYear)
While(Not(SQL End selection))
    SQL LOAD RECORD(10)
End while
```

Aquí tenemos que efectuar 10 bucles para recuperar los 100 registros. Si desea cargar todos los registros en un solo paso debemos utilizar:

```
SQL LOAD RECORD(SQL all records)
```

- Usando las etiquetas **Begin SQL**/**End SQL**:

```
ARRAY INTEGER(aBirthYear;0)
C_TEXT(vName)
vName:="Smith"
Begin SQL
    SELECT Birth_Year FROM PERSONS WHERE ename= <<vName>> INTO <<aBirthYear>>
End SQL
```

En esta situación, después de la ejecución de la instrucción *SELECT*, el array añoNacimiento contiene 100 elementos y cada elemento almacena un año de nacimiento de todos los 100 registros.
Si en lugar de una array, queremos almacenar los datos recuperados en una columna (un campo 4D), entonces 4D creará automáticamente tantos registros como sea necesario para guardar todos los datos. En nuestro ejemplo anterior, suponiendo que en la tabla PERSONAS hay 100 registros:

- Usando los comandos SQL genéricos de 4D:

```
C_TEXT(vName)
vName:="Smith"
$SQLStm:="SELECT Birth_Year FROM PERSONS WHERE ename= <<vName>>"
SQL EXECUTE($SQLStm;[MYTABLE]Birth_Year)
While(Not(SQL End selection))
    SQL LOAD RECORD(10)
End while
```

Aquí tenemos que efectuar 10 bucles para recuperar todos los 100 registros. Cada pasada por el bucle crea 10 registros en la tabla [MITABLA] y cada valor Año_Nacimiento recuperado de la tabla PERSONAS se guardará en el campo Año_Nacimiento.

- Usando las etiquetas **Begin SQL**/**End SQL**:

```
C_TEXT(vName)
vName:="Smith"
Begin SQL
    SELECT Birth_Year FROM PERSONS WHERE ename= <<vName>> INTO
<<[MYTABLE]Birth_Year>>
End SQL
```

En este caso, durante la ejecución de la instrucción *SELECT*, se crearán 100 registros en la tabla [MITABLA] y cada campo Año_Nacimiento contendrá los datos correspondientes de la tabla PERSONAS, columna Birth_Year..

Uso de un listbox

4D incluye un funcionamiento automático específico (palabra clave **LISTBOX**) que permite poner los datos de las consultas *SELECT* en un list box. Para mayor información, consulte el Manual de Diseño.

Optimización de las peticiones

Por razones de optimización, es preferible utilizar expresiones 4D en lugar de funciones SQL en las consultas. Las expresiones 4D se calcularán una vez antes de la ejecución de la consulta mientras que las funciones SQL se evalúan para cada registro encontrado.

Por ejemplo, con la siguiente instrucción:

```
SQL EXECUTE("SELECT nombreCompleto FROM PEOPLE WHERE nombreCompleto=
<<vApellido+vNombre>>")
```

... la expresión vApellido+vNombre se calcula una vez, antes de la ejecución de la consulta. Con la siguiente instrucción:

```
SQL EXECUTE("SELECT nombreCompleto FROM PEOPLE WHERE
nombreCompleto=CONCAT(<<vApellido>>,<<vNombre>>)")
```

... la función **CONCAT(<<vApellido>>,<<vNombre>>)** se llama para cada registro de la tabla, es decir, la expresión se evalúa para cada registro.

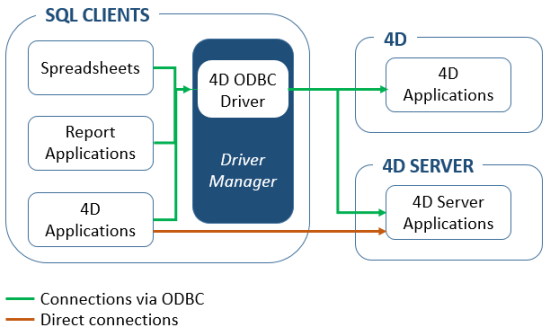
Configuración del servidor SQL de 4D

El servidor SQL de 4D permite el acceso externo a los datos almacenados en la base 4D. Para las aplicaciones de terceras partes y las aplicaciones 4D, este acceso se realiza mediante un driver ODBC 4D. También es posible realizar conexiones directas entre una aplicación cliente 4D y 4D Server. Todas las conexiones se hacen usando el protocolo TCP/IP.

El servidor SQL de una aplicación 4D puede detenerse o iniciarse en cualquier momento. Además, por motivos de rendimiento y seguridad, puede especificar el puerto TCP y la dirección IP de escucha y restringir las posibilidades de acceso a la base de datos 4D.

Acceso externo al servidor SQL

El acceso externo al servidor SQL de 4D puede efectuarse vía ODBC (todas las configuraciones), o directamente (aplicación cliente 4D conectado a 4D Server). Esto se resume en el siguiente diagrama:



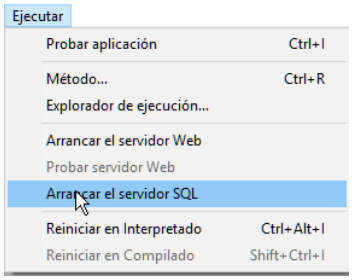
A nivel de búsquedas, la apertura de una conexión externa o vía ODBC se lleva a cabo utilizando el comando **SQL LOGIN**. Para mayor información, consulte la descripción de este comando.

- **Conexiones vía ODBC:** 4D ofrece un driver ODBC que permite a toda aplicación de terceros (hoja de cálculo de tipo Excel®, otros DBMS, etc.) o cualquier otra aplicación 4D conectarse al servidor SQL de 4D. El driver ODBC 4D debe instalarse en la máquina SQL Client. La instalación y configuración del driver ODBC 4D se detalla en otro manual.
- **Conexiones directas:** sólo una aplicación 4D Server puede responder a las consultas SQL directas procedentes de otros aplicaciones 4D. Del mismo modo, sólo las aplicaciones 4D de gama "Profesional" pueden abrir una conexión directa a otra aplicación 4D. Durante una conexión directa, el intercambio de datos se efectúa automáticamente en modo sincrónico, que elimina las cuestiones relacionadas con la sincronización y la integridad de los datos. Sólo una conexión está autorizado por proceso. Si desea establecer varias conexiones simultáneas, debe crear tantos procesos como sea necesario. Las conexiones directas, puede asegurarse seleccionando la opción **Activar TLS** del lado objetivo de la conexión (4D Server) en la pestaña "SQL" de las Propiedades de la base. Las conexiones directas sólo están autorizadas por 4D Server si el servidor SQL está activo. La principal ventaja de las conexiones directas es que se aceleran los intercambios de datos.

Iniciar y detener el servidor SQL

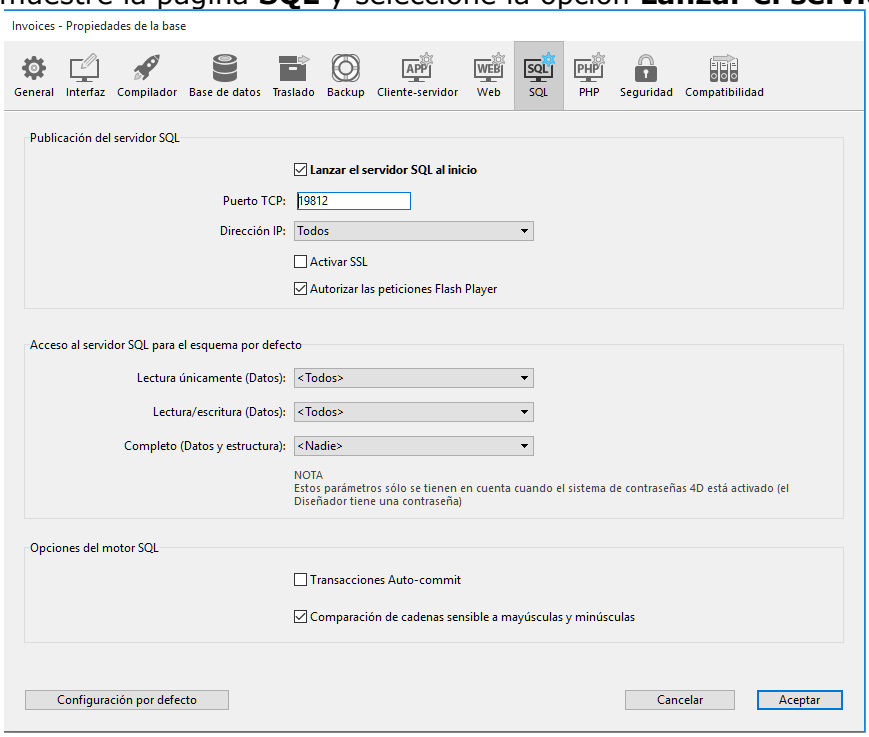
El servidor SQL de 4D puede iniciarse o detenerse de tres formas:

- Manualmente, utilizando los comandos **Arrancar el servidor SQL** en el menú **Ejecutar** de la aplicación 4D:



Nota: con 4D Server, se puede acceder a este comando como botón en la **Página Servidor SQL**.
Cuando se lanza el servidor, esta opción de menú cambia a **Detener el servidor SQL**.

- Automáticamente al inicio de la aplicación, vía las Propiedades de la base. Para hacer esto, muestre la página **SQL** y seleccione la opción **Lanzar el servidor SQL al inicio**:



- Por programación, usando los comandos **START SQL SERVER** y **STOP SQL SERVER** (tema "SQL").
Cuando el servidor SQL está detenido (o cuando no se ha iniciado), 4D no responderá a ninguna consulta SQL externa.
Nota: detener el servidor SQL no afecta el funcionamiento interno del motor SQL de 4D. El motor SQL siempre está disponible para las consultas internas.

Preferencias de publicación del servidor SQL

Es posible configurar los parámetros de publicación del servidor SQL integrado de 4D. Estos parámetros se encuentran en la página **SQL** de las Propiedades de la base:

- La opción **Lanzar el servidor SQL al inicio** permite iniciar el servidor SQL al inicio de la aplicación.
- **Puerto TCP**: por defecto, el servidor SQL de 4D responde a las peticiones en el puerto TCP 19812. Si este puerto ya está siendo utilizado por otro servicio o si sus parámetros de conexión requieren de otra configuración, puede cambiar el puerto TCP utilizado por el servidor SQL de 4D.
Notas:
- si pasa 0, 4D utilizará el número de puerto TCP por defecto, es decir 19812.
- Puede definir este valor por programación utilizando el selector SQL Server Port ID del comando **SET DATABASE PARAMETER**.
- **Dirección IP**: permite definir la dirección IP de la máquina en la que el servidor SQL debe procesar las consultas SQL. Por defecto, el servidor responderá a todas las direcciones IP (opción **Todas**).
La lista desplegable "Dirección IP" contiene automáticamente todas las direcciones IP

presentes en la máquina. Cuando se selecciona una dirección en particular, el servidor sólo responderá a las consultas enviadas a esta dirección.
Esta funcionalidad está dirigida a las aplicaciones 4D alojadas en máquinas con varias direcciones TCP/IP.
Notas:
  - Del lado del cliente, la dirección IP y el puerto TCP del servidor SQL para que la aplicación se conecte deben estar correctamente configurados en la definición de la fuente de datos ODBC.
  - A partir de 4D v14, el servidor SQL soporta la notación de direcciones IPv6. El servidor acepta indiferentemente las conexiones IPv6 o IPv4 cuando la configuración "Dirección IP" de escucha del servidor es Todas. Para obtener más información, consulte la sección **Soporte de IP v6**.
- **Activar TLS**: esta opción indica si el servidor SQL debe activar el protocolo seguro TLS para el procesamiento de las conexiones SQL.
Note que cuando este protocolo está activo, la palabra clave ":ssl" debe añadirse al final de la dirección IP del servidor SQL cuando abra una conexión vía el comando **SQL LOGIN**.
Por defecto, el servidor SQL utiliza los archivos internos para la llave y certificado TLS.
Sin embargo puede utilizar los elementos personalizados: para hacer esto, sólo copie sus propios archivos *key.pem* y *cert.pem* en la siguiente ubicación: MiBase/Preferencias/SQL (donde "MiBase" representa la carpeta/paquete de la base).
Notas:
- A partir de 4D v16 R5, el protocolo por defecto es TLS 1.2.
- Por programación puede modificar este valor utilizando el selector Min TLS version con el comando **SET DATABASE PARAMETER**.
- **Autorizar las peticiones Flash Player**: esta opción permite activar el mecanismo de soporte a las solicitudes Flash Player por el servidor SQL de 4D. Este mecanismo se basa en la presencia de un archivo, llamado "socketpolicy.xml," en la carpeta preferencias de la base (Preferencias/SQL/Flash/). Este archivo es requerido por Flash Player para permitir conexiones entre dominios o conexiones por tomas de aplicaciones Flex (Web 2.0).

  En la versión anterior de 4D, este archivo se tenía que agregar manualmente. A partir de ahora, la activación se realiza utilizando la opción **Autorizar las peticiones Flash Player**: al activar esta opción, las peticiones Flash Player son aceptadas y un archivo "socketpolicy.xml" genérico se crea para la base si es necesario.

  When you deselect this option, the "socketpolicy.xml" file is disabled (renamed). Any Flash Player queries received subsequently by the SQL server are then rejected.
  On opening of the database, the option is checked or not checked depending on the presence of an active "socketpolicy.xml" file in the preferences folder of the database.

  Cuando desactiva esta opción, el archivo "socketpolicy.xml" es deshabilitado (renombrado). Todas las consultas de Flash Player recibidas posteriormente por el servidor SQL se rechazan. Al abrir la base de datos, la opción está marcada o no seleccionada dependiendo de la presencia de un archivo "socketpolicy.xml" activo en la carpeta de preferencias de la base.

  Nota: es posible definir la codificación utilizada por el servidor SQL para el procesamiento de solicitudes externas utilizando el comando 4D **SQL SET OPTION**.

Control del acceso SQL para el esquema por defecto

Por razones de seguridad, es posible controlar las acciones que las consultas externas enviadas al servidor SQL pueden realizar en la base de datos 4D. Este control se efectúa en dos niveles:

- A nivel del tipo de acción autorizada,
- A nivel del usuario que efectúa la consulta.
  Estos ajustes se pueden hacer en la página **SQL** de las Propiedades de la base.

Nota: puede utilizar el **Método de base On SQL Authentication** para controlar de manera personalizada las peticiones externas al motor SQL de 4D.

Los parámetros definidos en esta caja de diálogo se aplican al esquema por defecto. El control de los accesos externos a la base se basan en el concepto de esquemas SQL (ver la sección **Implementaciones del motor SQL de 4D**). Si no crea esquemas personalizados, el esquema por defecto incluye todas las tablas de la base. Si crea otros esquemas con los derechos de acceso

específicos y los asocia con las tablas, el esquema por defecto sólo incluirá las tablas que no están incluidas en los esquemas personalizados.

Puede configurar tres tipos distintos de acceso al esquema por defecto vía el servidor SQL:

- **Sólo lectura (datos)**: acceso ilimitado en lectura a todos los datos de las tablas de la base pero no está permitido agregar, modificar o eliminar registros, ni modificar la estructura de la base.
- **Lectura/escritura (datos)**: acceso en lectura y escritura (añadir, modificar y borrar) a todos los datos de las tablas de base, pero no la modificación de la estructura de la base.
- **Completo (datos y estructura)**: acceso en lectura y escritura (añadir, modificar y borrar) a todos los datos de las tablas de la base, así como a la modificación de la estructura de la base (tablas, campos, relaciones, etc.).

Puede designar un conjunto de usuarios para cada tipo de acceso. Hay tres opciones disponibles para este propósito:

- <**Persona**>: si selecciona esta opción, el tipo de acceso será rechazado para todas las consultas, independientemente de su origen. Este parámetro se puede utilizar incluso cuando el sistema de gestión de acceso por contraseñas de 4D no esté activo.
- <**Todos**>: si selecciona esta opción, el tipo de acceso se aceptará para todas las consultas (sin límites).
- Grupo de usuarios: esta opción permite designar un grupo de usuarios autorizados a efectuar el tipo de acceso asociado. Esta opción requiere que la gestión de contraseñas de 4D esté activada. El usuario al origen de las consultas da su nombre y contraseña durante la conexión al servidor SQL.

ADVERTENCIA: cada tipo de acceso está definido de forma independiente de los demás. Más específicamente, si sólo asigna el tipo de acceso **Sólo lectura** a un grupo esto no tendrá ningún efecto ya que este grupo, así como también todos los demás van a seguir beneficiándose del acceso **Lectura/escritura** (asignado a <Todo el mundo> de forma predeterminada). Con el fin de definir un acceso **Sólo lectura**, también es necesario el acceso **Lectura/escritura**.

ADVERTENCIA: este mecanismo se basa en las contraseñas de 4D. Para que el control de acceso al servidor SQL tenga efecto, el sistema de contraseñas de 4D debe estar activo (una contraseña debe asignarse al Diseñador).

Nota: una opción de seguridad adicional puede establecerse a nivel de cada método de proyecto 4D. Para obtener más información, consulte el párrafo "Opción Disponible vía SQL" en la sección **Implementaciones del motor SQL de 4D**

Implementaciones del motor SQL de 4D

Básicamente, el motor SQL de 4D es compatible con SQL-92. Esto significa que para una descripción detallada de los comandos, funciones, operadores y sintaxis a utilizar, puede referirse a la documentación del SQL-92. Múltiples recursos sobre este tema están disponibles en Internet.

Sin embargo, el motor SQL de 4D no soporta el 100% de las funciones del SQL-92 y ofrece otras funciones adicionales específicas.

Esta sección cubre las principales implementaciones y limitaciones del motor SQL de 4D.

Limitaciones generales

Puesto que el motor SQL de 4D se ha integrado en el corazón de la base de datos de 4D, todas las limitaciones relativas al número máximo de tablas, columnas (campos) y registros por base, así como las reglas para dar nombres a las tablas y columnas, son las mismas a las del motor estándar de 4D. Se enumeran a continuación.

- Número máximo de tablas: teóricamente dos mil millones, pero por razones de compatibilidad con 4D: 32767.
- Número máximo de columnas (campos) por tabla: teóricamente dos mil millones columnas (campos), pero por razones de compatibilidad con 4D: 32767.
- Número máximo de líneas (registros) por tabla: mil millones.
- Número máximo de llaves de índice: 128 mil millones para los tipos alfa, texto y flotante; 256 mil millones para los otros tipos (escalares).
- Una llave primaria no puede ser un valor NULL y debe ser única. No es necesario indexar las columnas (campos) llaves primarias.
- Número máximo de caracteres permitido para los nombres de tablas y campos: 31 caracteres (limitación 4D).

No está permitido crear varias tablas con el mismo nombre. Aplica el mecanismo de control estándar de 4D.

Tipos de datos

La siguiente tabla indica los tipos de datos soportados en el SQL de 4D así como su tipo correspondiente en 4D:

Tipo 4D SQL	Descripción	4D
Varchar	Texto Alfanumérico	Texto o Alfa
Real	Número de punto flotante en el rango de +/-1.7E308	Real
Numeric	Número entre +/- 2E64	Entero 64 bits
Float	Número de punto flotante (virtualmente infinito)	Float
Smallint	Número entre -32 768 y 32 767	Entero
Int	Número entre -2 147 483 648 y 2 147 483 647	Entero largo, Entero
Int64	Número entre +/- 2E64	Entero 64 bits
UUID	Número de 16 bytes (128 bits) representado por 32 caracteres hexadecimales	Alpha format UUID
Bit	Campo que sólo acepta los valores TRUE/FALSE o 1/0	Booleano
Boolean	Campo que sólo acepta los valores TRUE/FALSE o 1/0	Booleano
Blob	Hasta 2 GB; todo objeto binario tal como una imagen, un documento, una aplicación	Blob
Bit varying	Hasta 2 GB; todo objeto binario tal como una imagen, un documento, una aplicación	Blob
Clob	Hasta 2 GB de texto. Esta columna (campo) no puede indexarse. No se guarda en el registro mismo.	Texto
Text	Hasta 2 GB de texto. Esta columna (campo) no puede indexarse. No se guarda en el registro mismo.	Texto
Timestamp	Fecha en formato 'YYYY/MM/DD' y hora en formato 'HH:MM:SS:ZZ'	Partes Fecha y Hora generados por separado (conversión automática)
Duration	Duración en formato 'HH:MM:SS:ZZ'	Hora
Interval	Duración en formato 'HH:MM:SS:ZZ'	Hora
Picture	Imagen PICT hasta de 2 GB	Imagen

La conversión entre los tipos de datos numéricos es automática.
Las cadenas que representan un número no se convierten en un número correspondiente. Hay funciones *CAST* especiales que convertirán los valores de un tipo a otro.
Los siguientes tipos de datos SQL no se implementan:

- NCHAR
- NCHAR VARYING.

Valores NULL en 4D

Los valores NULL se implementan en el lenguaje SQL de 4D, así como en el motor de base de datos de 4D. Sin embargo, no son soportados en el lenguaje 4D. Sin embargo, es posible leer y escribir valores NULL en un campo 4D utilizando los comandos **Is field value Null** y **SET FIELD VALUE NULL**.

Compatibilidad de los procesos y opción Traducir los NULL en valores vacíos

Por razones de compatibilidad en 4D, los valores NULL almacenados en las tablas de la base de datos 4D se convierten automáticamente en valores por defecto cuando se manipulan vía el lenguaje 4D. Por ejemplo, en el caso de la siguiente instrucción:

```
mivarAlf:=[mitabla]MiCampoAlfa
```

... si el campo MiCampoAlfa contiene un valor NULL, la variable mivarAlfa contendrá "" (cadena vacía).

Los valores por defecto dependen del tipo de dato:

- Para los tipos Alfa y Texto: ""
- Para los tipos Real, Entero y Entero largo: 0
- Para el tipo Fecha: "00/00/00"
- Para el tipo Hora: "00:00:00"
- Para el tipo Booleano: Falso
- Para el tipo Imagen: Imagen vacía
- Para el tipo BLOB: BLOB vacío

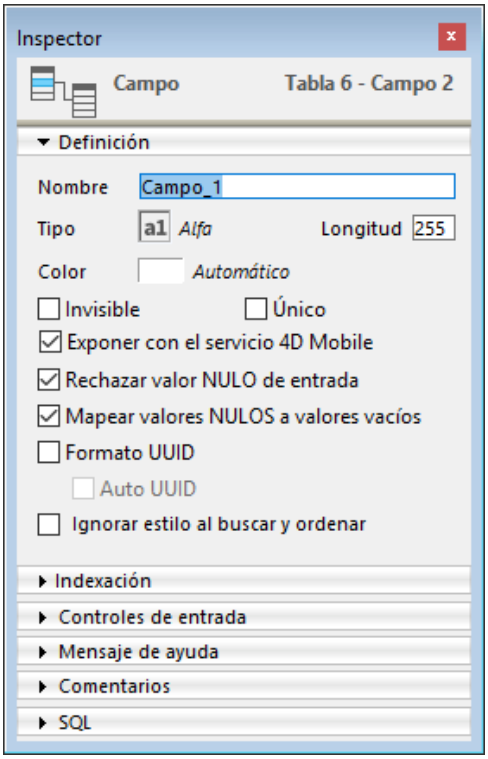
Por otra parte, este mecanismo, en principio, no se aplica a los tratamientos efectuados a nivel del motor de la base de datos 4D, tales como las consultas. De hecho, la búsqueda de un valor "vacío" (por ejemplo mivalor = 0) no encuentra registros que almacenen el valor NULL y viceversa. Cuando los dos tipos de valores (valores por defecto y NULL) están presentes en los registros para un mismo campo, algunos procesos pueden alterarse o necesitar código adicional.

Para evitar estos inconvenientes, una opción permite estandarizar todos los procedimientos el lenguaje 4D: **Mapear valores NULOS a valores vacíos**. Esta opción, que se encuentra en la ventana Inspector de campos del editor de estructura, permite extender el principio de utilizar los valores por defecto en todos los tratamientos. Los campos que contengan valores NULL se consideran sistemáticamente que contienen valores por defecto. Esta opción está seleccionada por defecto.

La propiedad **Mapear valores NULOS a valores vacíos** se tiene en cuenta a un nivel muy bajo del motor de la base de datos. Actúa más en particular en el comando **Is field value Null**.

Rechazar valor NULO de entrada

La propiedad de campo **Rechazar valor NULO de entrada** permite evitar el almacenamiento de valores NULL:



Cuando este atributo está seleccionado para un campo, no será posible almacenar el valor NULL en este campo. Esta propiedad de bajo nivel corresponde exactamente al atributo NOT NULL de SQL. Generalmente, si quiere poder utilizar los valores NULL en su base de datos 4D, se recomienda utilizar exclusivamente el lenguaje SQL de 4D.

Nota: en 4D, los campos también puede tener el atributo "Obligatorio". Los dos conceptos son similares, pero su alcance es distinto: el atributo "obligatorio" es un control de entrada, mientras que el atributo "Rechazar valor NULO de entrada" trabaja a nivel del motor de la base de datos. Si un campo con este atributo recibe un valor NULL, se genera un error.

Expresiones fecha y hora

Constantes fecha y hora

El servidor SQL integrado de 4D soporta las constantes fecha y hora de acuerdo al API ODBC. Esta es la sintaxis para las secuencias de constantes fecha y hora ODBC:

```
{constant_type 'value'}
```

tipo_constante	valor	Descripción
d	aaaa-mm-dd	Fecha únicamente
t	hh:mm:ss[.fff]	Hora únicamente
ts	aaaa-mm-dd hh:mm:ss[.fff]	Fecha y hora (timestamp)

Note: *fff* indica milisegundos.

Por ejemplo, puede utilizar las siguientes constantes:

```
{ d '2013-10-02' }
{ t '13:33:41' }
{ ts '1998-05-02 01:23:56.123' }
```

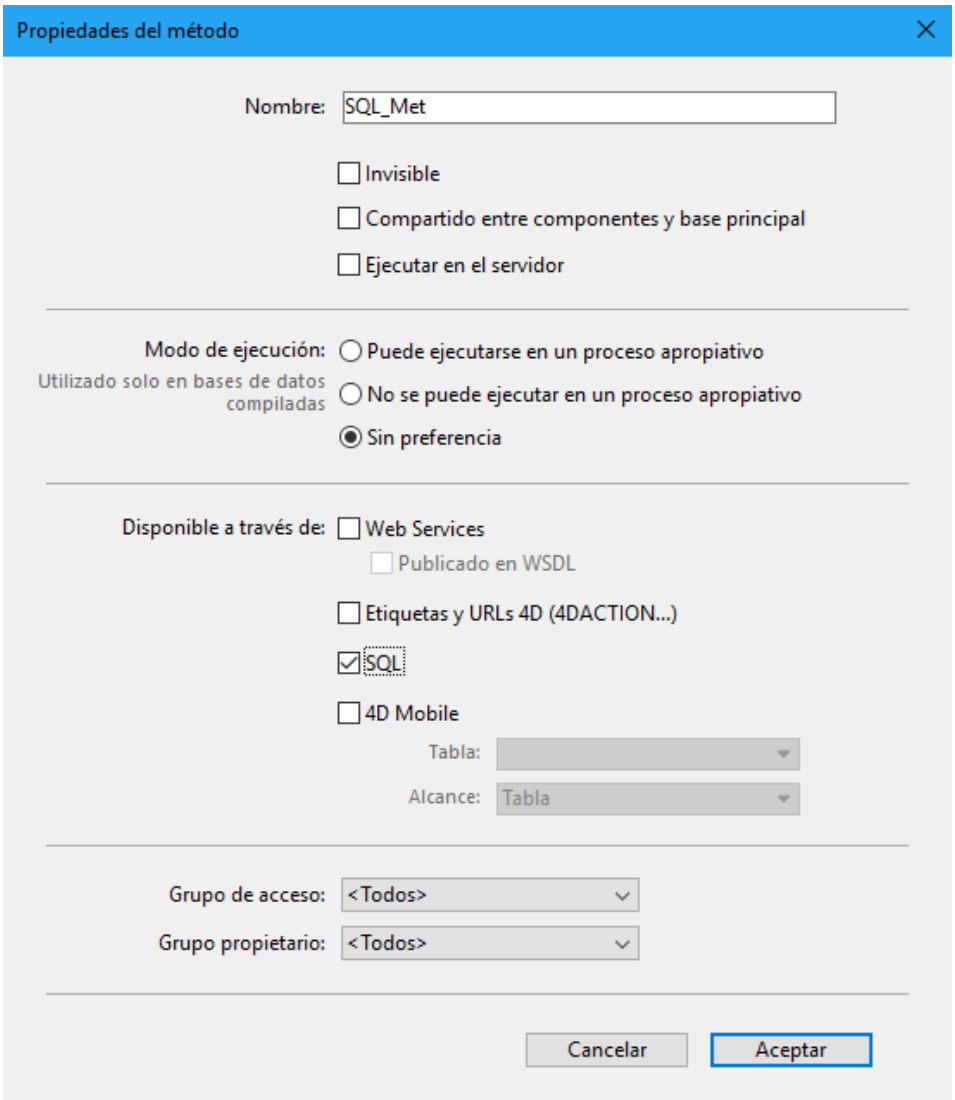
Búsquedas en fechas vacías

El analizador SQL de fecha rechaza toda expresión fecha que especifique "0" como el día o el mes. Las expresiones como {d'0000-00-00'} o CAST('0000-00-00' AS TIMESTAMP) generan un error. Para realizar en SQL búsquedas en fechas vacías (no confundir con fechas nulas), debe usar una expresión 4D intermedia. Por ejemplo:

```
C_LONGINT($count)
$nullDate:=!00-00-00!
Begin SQL
        SELECT COUNT(*) FROM Table_1
        WHERE myDate = :$nullDate
        INTO :$count;
End SQL
```

Opción “Disponible vía SQL”

Una propiedad de seguridad se ha añadido para los métodos proyecto 4D: **Disponible vía SQL**:



Cuando está seleccionada, esta opción permite la ejecución del método de proyecto por el motor SQL de 4D. No está seleccionada por defecto, lo que significa que los métodos proyecto 4D están protegidos y no pueden ser llamados por el motor SQL de 4D a menos que haya sido expresamente autorizado al seleccionar esta opción.

Esta propiedad se aplica a todas las consultas SQL internas y externas, si se ejecuta vía el driver ODBC, el código SQL insertado entre las etiquetas **Begin SQL**/**End SQL** o el comando **QUERY BY SQL**.

Notas:

- Incluso cuando un método tiene el atributo "Disponible vía SQL", los derechos de acceso definidos a nivel de las Propiedades de la base y las propiedades del método se tienen en cuenta durante su ejecución.
- La función ODBC SQLProcedure devuelve únicamente los métodos proyecto que tienen el atributo "Disponible vía SQL".

Opciones motor SQL

- **Transacciones Auto-commit**: esta opción permite activar el mecanismo de auto-commit en el motor SQL. El propósito del modo auto-commit es preservar la integridad referencial de los datos. Cuando esta opción está seleccionada, toda búsqueda **SELECT**, **INSERT**, **UPDATE** y **DELETE** (SIUD) no efectuada dentro de una transacción se incluye automáticamente en una operación ad hoc. Esto garantiza que las consultas se ejecutarán en su totalidad o en caso de error, se cancelarán por completo.
  Las consultas incluidas en una transacción (gestión personalizada de la integridad referencial) no se verán afectadas por esta opción.
  Cuando esta opción no está seleccionada, no se genera transacción automática (excepto para las consultas **SELECT... FOR UPDATE**, consulte el comando **SELECT**). Por defecto, esta opción no está seleccionada.

También puede administrar esta opción por programación utilizando el comando **SET DATABASE PARAMETER**.
Nota: sólo las bases locales consultadas por el motor SQL de 4D se ven afectadas por este parámetro. En el caso de las conexiones externas a otras bases de datos SQL, el mecanismo de auto-commit es manejado por los motores SQL remotos.

- **Tener en cuenta las mayúsculas y minúsculas en las comparaciones de cadenas**: esta opción permite modificar la sensibilidad a las mayúsculas y minúsculas de los caracteres en las consultas SQL. Está seleccionada por defecto, lo que significa que el motor SQL diferencia entre mayúsculas y minúsculas, así como también entre caracteres acentuados cuando se comparan cadenas (ordenaciones y búsquedas). Por ejemplo, "ABC" = "ABC", pero "ABC" # Abc "."
  En algunos casos, por ejemplo para alinear el funcionamiento del motor SQL con el del motor 4D, es posible que desee que las comparaciones de cadenas no sean sensibles a las mayúsculas ("ABC" = "Abc"). Para ello, sólo tendrá que deseleccionar esta opción.
  También puede administrar esta opción por programación utilizando el comando **SET DATABASE PARAMETER**.

Esquemas

4D implementa el concepto de esquemas. Un esquema es un objeto virtual que contiene las tablas de la base. En el SQL, el propósito de los esquemas es asignar derechos de acceso específicos a los diferentes conjuntos de objetos de la base. Los esquemas dividen la base en entidades independientes que en conjunto forman toda la base. En otras palabras, una tabla siempre pertenece a un sólo esquema.

- Para crear un esquema, debe utilizar el comando *CREATE SCHEMA*. A continuación, puede utilizar los comandos *GRANT* y *REVOKE* para configurar los tipos de acceso a los esquemas.
- Para asociar una tabla a un esquema, puede llamar a los comandos *CREATE TABLE* o *ALTER TABLE*. También puede usar el menú pop-up "Esquemas" del **Inspector** del editor de estructura de 4D, que lista todos los esquemas definidos en la base.
- El comando *DROP SCHEMA* permite eliminar un esquema.

Nota: el control de acceso vía los esquemas sólo se aplica a las conexiones desde el exterior. El código SQL ejecutado en 4D vía las etiquetas **Begin SQL**/**End SQL**, **SQL EXECUTE**, **QUERY BY SQL**, siempre tiene acceso total.

Conexiones a las fuentes SQL

La arquitectura multi-bases se implementa a nivel del servidor SQL de 4D. Desde 4D es posible:

- conectarse a una base existente utilizando el comando **SQL LOGIN**.
- Pasar de una a otra utilizando los comandos **SQL LOGIN** y **SQL LOGOUT**.
- Para abrir y utilizar otra base 4D en lugar de la base actual utilizando el comando **USE DATABASE**.

Llave primaria

En el lenguaje SQL, una llave primaria permite identificar en una tabla la(s) columna(s) (campos) responsables de designar de manera única los registros (líneas). La definición de una llave primaria es particularmente necesaria para la función de replicación de los registros de una tabla de 4D (vér la sección **Replicación vía SQL**) y para la historialización de las tablas 4D a partir de la v14.
4D le permite administrar la llave primaria de una tabla de varias maneras:

- Vía el lenguaje SQL
- Utilizando el editor de estructura de 4D.

Notas:

- También puede definir llaves primarias utilizando **Gestión de llaves primarias** de 4D en el modo Diseño.

- Para una descripción del uso de las llaves primarias, consulte **Reglas de uso de los campos llaves primarias**.

Definir la llave primaria vía el lenguaje SQL

Puede definir una llave primaria durante la creación de una tabla (vía el comando *CREATE TABLE*) o al agregar o modificar una columna (vía el comando *ALTER TABLE*). La llave primaria se define utilizando la cláusula PRIMARY KEY seguida por el nombre de la columna o de una lista de columnas. Para obtener más información, consulte la sección **definición_llave_primaria**.

Definir la llave primaria vía el editor de estructura

4D le permite crear y eliminar directamente llaves primarias vía el menú contextual del editor de la estructura.

Para mayor información, consulte, **Definir o eliminar una llave primaria** en el manual de Diseño 4D.

Vistas SQL

El motor SQL integrado de 4D soporta **vistas SQL** estándar. Una vista es una tabla virtual con datos que pueden provenir de varias tablas de la bases de datos. Una vez que se define una vista, se puede utilizar en un instrucción **SELECT** como una tabla real.

Los datos se encuentran en una vista se definen mediante una petición de definición basada en el comando **SELECT**. Las tablas reales utilizadas en la consulta de definición son llamadas "tablas fuentes". Una vista SQL contiene columnas y líneas como una tabla estándar, pero en realidad no existe, sino que es sólo una representación resultante del procesamiento y se almacena en la memoria durante la sesión. Sólo la definición de la vista se almacenada temporalmente.

Dos comandos SQL se utilizan para administrar vistas en 4D v14: **Comandos SQL** y **DROP VIEW**.

Tablas sistema

El catálogo SQL de 4D incluye varias tablas sistema, accesibles por todo usuario SQL que tenga acceso de lectura: *_USER_TABLES, _USER_COLUMNS*, *_USER_INDEXES*, *_USER_CONSTRAINTS*, *_USER_IND_COLUMNS*, *_USER _CONS_ COLUMNS*, *_USER_SCHEMAS, _USER_VIEWS* y *_USER_VIEW_COLUMNS*.

Conforme a los usos en el mundo SQL, las tablas sistema describen la estructura de la base de datos. Esta es una descripción de estas tablas y sus campos:

_USER_TABLES		**Describe las tablas usuario de la base**
TABLE_NAME	VARCHAR	Nombre de tabla
TEMPORARY	BOOLEAN	True si la tabla es temporal; de lo contrario, false
TABLE_ID	INT64	Número de tabla
SCHEMA_ID	INT32	Número de esquema
REST_AVAILABLE	BOOLEAN	True si la tabla está expuesta con el servicio REST; de lo contrario, False
LOGGED	BOOLEAN	True si las operaciones de la tabla están incluídas en el archivo de historial; de lo contrario, False

_USER_COLUMNS		**Describe las columnas de las tablas usuarios de la base**
TABLE_NAME	VARCHAR	Nombre de tabla
COLUMN_NAME	VARCHAR	Nombre de columna
DATA_TYPE	INT32	Tipo de columna
DATA_LENGTH	INT32	Largo de la columna
OLD_DATA_TYPE	INT32	Tipo de campo (ver el comando **Type**)
NULLABLE	BOOLEAN	True si la columna acepta valores NULL; de lo contrario, false
TABLE_ID	INT64	Número de tabla
COLUMN_ID	INT64	Número de columna
UNIQUENESS	BOOLEAN	True si la columna se declara única; de lo contrario, False
AUTOGENERATE	BOOLEAN	True si el valor de la columna se genera automáticamente para cada nuevo registro; de lo contrario, False
AUTOINCREMENT	BOOLEAN	True si el valor de la columna se incrementa automáticamente; de lo contrario, False
REST_AVAILABLE	BOOLEAN	True si la columna está expuesta con el servicio REST; de lo contrario, False

_USER_INDEXES		**Describe los índices usuarios de la base**
INDEX_ID	VARCHAR	Número de índice
INDEX_NAME	VARCHAR	Nombre de índice
INDEX_TYPE	INT32	Tipo de índice (1=BTree / Composite, 3=Cluster / Palabras claves, 7=Auto, 8=Auto para campo Objeto)
KEYWORD	BOOLEAN	True si el índice es un índice de palabras claves; de lo contrario, False
TABLE_NAME	VARCHAR	Nombre de la tabla con índice
UNIQUENESS	BOOLEAN	True si el índice impone una restricción de unicidad; de lo contrario, false
TABLE_ID	INT64	Número de tabla con índice

_USER_IND_COLUMNS		**Describe las columnas de índices usuarios de la base**
INDEX_ID	VARCHAR	Número de índice
INDEX_NAME	VARCHAR	Nombre de índice
TABLE_NAME	VARCHAR	Nombre de tabla con índice
COLUMN_NAME	VARCHAR	Nombre de columna con índice
COLUMN_POSITION	INT32	Posición de columna en el índice
TABLE_ID	INT64	Número de tabla con índice
COLUMN_ID	INT64	Número de columna

_USER_CONSTRAINTS		**Describe las restricciones usuarios de la base**
CONSTRAINT_ID	VARCHAR	Número de restricción
CONSTRAINT_NAME	VARCHAR	Nombre de restricción
CONSTRAINT_TYPE	VARCHAR	Tipo de restricción
TABLE_NAME	VARCHAR	Nombre de tabla con restricción
TABLE_ID	INT64	Número de tabla con restricción
DELETE_RULE	VARCHAR	Regla de supresión – CASCADE o RESTRICT
RELATED_TABLE_NAME	VARCHAR	Nombre de tabla relacionada
RELATED_TABLE_ID	INT64	Número de tabla relacionada

_USER_CONS_COLUMNS		**Describe las columnas de restricciones usuarios de la base**
CONSTRAINT_ID	VARCHAR	Número de restricción
CONSTRAINT_NAME	VARCHAR	Nombre de restricción
TABLE_NAME	VARCHAR	Nombre de tabla con restricción
TABLE_ID	INT64	Número de tabla con restricción
COLUMN_NAME	VARCHAR	Nombre de columna con restricción
COLUMN_ID	INT64	Número de columna con restricción
COLUMN_POSITION	INT32	Posición de columna con restricción
RELATED_COLUMN_NAME	VARCHAR	Nombre de columna relacionada en una restricción
RELATED_COLUMN_ID	INT32	Número de columna relacionada en una restricción

_USER_SCHEMAS		**Describe los esquemas de la base**
SCHEMA_ID	INT32	Número del esquema
SCHEMA_NAME	VARCHAR	Nombre del esquema
READ_GROUP_ID	INT32	Número del grupo con acceso sólo lectura
READ_GROUP_NAME	VARCHAR	Nombre del grupo con acceso lectura-escritura
READ_WRITE_GROUP_ID	INT32	Número del grupo con acceso lectura-escritura
READ_WRITE_GROUP_NAME	VARCHAR	Nombre del grupo having read-write access
ALL_GROUP_ID	INT32	Número del grupo con acceso completo
ALL_GROUP_NAME	VARCHAR	Nombre del grupo con acceso completo

_USER_VIEWS		**Describe las vistas de los usuarios de la base**
VIEW_NAME	VARCHAR	Nombre de vista
SCHEMA_ID	INT32	ID del nom_schema al cual pertenece la vista

_USER_VIEW_COLUMNS		**Describe las columnas de las vistas de los usuarios de la base**
VIEW_NAME	VARCHAR	Nombre de vista
COLUMN_NAME	VARCHAR	Nombre de columna
DATA_TYPE	INT32	Tipo de columna
DATA_LENGTH	INT32	Tamaño de columna
NULLABLE	BOOLEAN	True si columna acepta los valores NULL; de lo contrario, False

Nota: las tablas sistema se asignan a un esquema particular llamado **SYSTEM_SCHEMA**. Este esquema no puede modificarse o borrarse. No aparece en la lista de esquemas mostrada en el inspector de tablas. Es accesible en modo lectura únicamente por todos los usuarios.

Replicación vía SQL

4D ofrece un mecanismo que permite replicar o sincronizar los datos de dos o más bases 4D vía SQL. Esta funcionalidad específica se puede utilizar para crear una o más bases espejos, garantizando la disponibilidad permanente de los datos.

El principio es el siguiente: una base de datos objetivo replica en local los datos de una base de datos fuente remota. Las actualizaciones se llevarán a cabo periódicamente por la base local que recupera los datos de la base remota. La replicación se lleva a cabo a nivel de tabla: usted replica los datos de una tabla de la base remota en una tabla de la base local.
Esto es posible gracias a la utilización de marcadores (stamps) y de comandos SQL específicos.

En el editor de la estructura, una propiedad de tabla permite activar el mecanismo de replicación en la base remota y local. Del lado de la base local, el comando SQL **REPLICATE** permite recuperar datos de una tabla de la base remota y luego integrar estos datos en una tabla de la base local. El comando SQL **SYNCHRONIZE**, se utiliza para llevar a cabo la sincronización de las dos tablas.

Campos virtuales

Cada tabla de la base 4D puede ser asignada con tres campos "virtuales": __ROW_ID, __ROW_STAMP y __ROW_ACTION. Estos campos se llaman "virtuales" para diferenciarlos de los "clásicos" porque tienen propiedades específicas: se llenan automáticamente, puede se leídos pero no modificados por los usuarios y no aparecen en las tablas sistema de la base. La siguiente tabla describe estos campos, así como su modo de uso:

Campo virtual	Tipo	Contenido	Uso
__ROW_ID	Int32	ID del registro	En toda instrucción SQL excepto **REPLICATE** o **SYNCHRONIZE**
__ROW_STAMP	Int64	Información de replicación del registro	En toda instrucción SQL
__ROW_ACTION	Int16	Tipo de acción efectuada en el registro: 1 = Adición o modificación, 2 = Supresión	Únicamente con el comando **REPLICATE** o **SYNCHRONIZE**

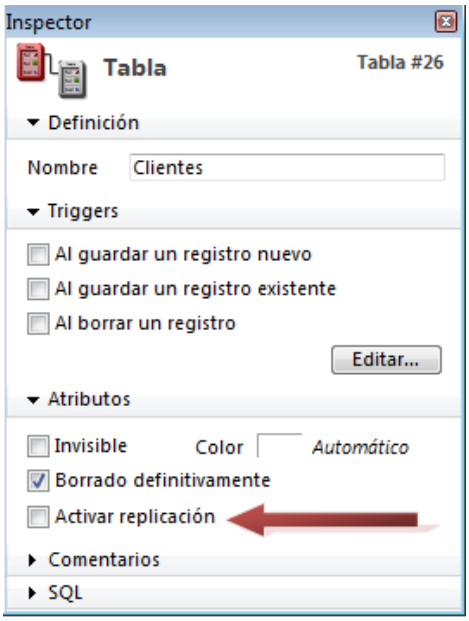
Cuando los mecanismos de replicación están activos, tan pronto como un registro se crea, modifica o elimina, la información correspondiente se actualiza automáticamente en los campos virtuales de este registro.

Activar replicación

Por defecto los mecanismos que permiten la replicación no están activos. Debe activarlos explícitamente tanto en la base remota y en la base local para cada tabla utilizada en la replicación o sincronización.

Tenga en cuenta que la activación del mecanismo no activa la replicación, con el fin de que los datos se repliquen en una base local o sincronizada, debe utilizar los comandos **REPLICATE** o **SYNCHRONIZE**.

Para activar el mecanismo interno de replicación, debe utilizar en cada tabla (en la base remota y local), la propiedad de tabla **Activar replicación** accesible en el Inspector de tablas:



Nota: para que el mecanismo de replicación pueda funcionar, debe especificar una llave primaria para las tablas implicadas en las bases remota y local. Puede crear esta llave vía el editor de estructura o los comandos de SQL. Si no se ha especificado llave primaria, la opción está en gris.

Cuando esta opción está activa, 4D genera la información necesaria para replicar los registros de la tabla (basada en particular en la llave primaria de la tabla). Esta información se almacena en los campos virtuales __ROW_STAMP y __ROW_ACTION.

Nota: es posible activar y desactivar la generación de información de replicación vía los comandos SQL *CREATE TABLE* y *ALTER TABLE*, utilizando las palabras claves ENABLE REPLICATE y DISABLE REPLICATE. Para mayor información, consulte la descripción de estos comandos.

ATENCIÓN: seleccionar esta opción provoca la publicación de la información necesaria por los mecanismos de replicación. Por razones de seguridad, el acceso a esta información debe estar protegido, así como debe estar protegido el acceso a sus datos cuando se publican. Por lo tanto, cuando implemente un sistema de replicación utilizando esta opción, debe asegurarse de:

- si el servidor SQL se lanza, el acceso está protegido utilizando las contraseñas 4D y/o los esquemas SQL (ver **Configuración del servidor SQL de 4D**),
- si el servidor HTTP se lanza, el acceso está protegido utilizando las contraseñas 4D y/o los esquemas SQL (ver **Configuración del servidor SQL de 4D**) y/o del y/o de la definición de una estructura virtual vía los comandos **SET TABLE TITLES** y **SET FIELD TITLES**. Para mayor información, consulte el párrafo "URL 4DSYNC/" en la sección **QR Get drop column**.

Actualización de la base local

Una vez que el mecanismo de replicación está activo en cada tabla de cada base, puede utilizarlo desde la base local vía el comando SQL **REPLICATE**. Para mayor información, consulte la descripción de este comando.

Soporte de combinaciones

El motor SQL de 4D amplía el soporte de las sentencias JOIN.

Las sentencias JOIN puede ser internas o externas, implícitas o explícitas. Las uniones internas (INNER JOIN) implícitas son soportadas por el comando *SELECT*. También puede generar JOINs internas y externas explícitas utilizando la palabra clave SQL **JOIN**.

Nota: la implementación actual de JOINs en el motor SQL de 4D no incluye:

- JOINs naturales.
- el constructor USING en las JOINs internas.
- JOINs cruzados.

Presentación

La sentencia JOIN permite hacer combinaciones entre los registros de dos o más tablas y combinar el resultado en una tabla nueva, llamada JOIN.

Genere combinaciones con la instrucción *SELECT* que especifica las condiciones de la combinación.

A partir de 4D v15 R4, las combinaciones externas donde participan dos tablas y las combinaciones externas donde participan tres o más tablas son diferentes implementaciones y no siguen las mismas reglas. Por favor, consulte a continuación la sección que corresponda a sus necesidades.

Nota: por lo general, en el motor de base de datos, el orden de las tablas está determinado por el orden definido durante la búsqueda. Sin embargo, al usar combinaciones, el orden de las tablas se determina por la lista de tablas. En el ejemplo siguiente:
SELECT * FROM T1 RIGHT OUTER JOIN T2 ON = T2.depID T1.depID;
... el orden de las tablas es T1 y luego T2 (tal como aparecen en la lista de tablas) y no T1 y luego T2 (tal como aparecen en la condición del join).

Base de ejemplo

Para ilustrar cómo funcionan las uniones, vamos a utilizar la siguiente base de datos a lo largo de esta sección:

- Employees

name	depID	cityID
Alan	10	30
Anne	11	39
Bernard	10	33
Fabrice	12	35
Martin	15	30
Philip	NULL	33
Thomas	10	NULL

- Departments

depID	depName
10	Program
11	Engineering
NULL	Marketing
12	Development
13	Quality

- Cities

cityID	cityName
30	Paris
33	New York
NULL	Berlin

Si lo desea, puede generar esta base automáticamente ejecutando el siguiente código:

```
Begin SQL
        DROP TABLE IF EXISTS Employees;
        CREATE TABLE Employees ( depID INT32, name VARCHAR, cityID INT32);
        INSERT INTO Employees (name, depID, cityID) VALUES ('Alan', 10, 30);
        INSERT INTO Employees (name, depID, cityID) VALUES ('Anne', 11, 39);
        INSERT INTO Employees (name, depID, cityID) VALUES ('Bernard', 10, 33);
        INSERT INTO Employees (name, depID, cityID) VALUES ('Fabrice', 12, 35);
        INSERT INTO Employees (name, depID, cityID) VALUES ('Martin', 15, 30);
        INSERT INTO Employees (name, depID, cityID) VALUES ('Philip', NULL, 33);
        INSERT INTO Employees (name, depID, cityID) VALUES ('Thomas', 10, NULL);

        DROP TABLE IF EXISTS Departments;
        CREATE TABLE Departments ( depID INT32, depName VARCHAR );
        INSERT INTO Departments (depID, depName) VALUES (10, 'Program');
        INSERT INTO Departments (depID, depName) VALUES (11, 'Engineering');
        INSERT INTO Departments (depID, depName) VALUES (NULL, 'Marketing');
        INSERT INTO Departments (depID, depName) VALUES (12, 'Development');
        INSERT INTO Departments (depID, depName) VALUES (13, 'Quality');

        DROP TABLE IF EXISTS Cities;
        CREATE TABLE Cities ( cityID INT32, cityName VARCHAR );
        INSERT INTO Cities (cityID, cityName) VALUES (30, 'Paris');
        INSERT INTO Cities (cityID, cityName) VALUES (33, 'New York');
        INSERT INTO Cities (cityID, cityName) VALUES (NULL, 'Berlin');
End SQL
```

Combinaciones internas explícitas

Una combinación interna (inner join) está basada en una comparación de igualdad entre dos columnas.
Este es un ejemplo de join interna explícita:

```
SELECT *
        FROM employees, departments
        WHERE employees.DepID = departments.DepID;
```

En 4D, puede también utilizar la palabra clave JOIN para especificar una join interna explícita:

```
SELECT *
        FROM employees
        INNER JOIN departments
            ON employees.DepID = departments.DepID;
```

Esta búsqueda puede insertarse en el código 4D de la siguiente manera:

```
ARRAY TEXT(aName;0)
ARRAY TEXT(aDepName;0)
ARRAY INTEGER(aEmpDepID;0
ARRAY INTEGER(aDepID;0)
Begin SQL
        SELECT Employees.name, Employees.depID, Departments.depID, Departments.depName
        FROM Employees
                INNER JOIN Departments
                    ON Employees.depID = Departments.depID
                INTO :aName, :aEmpDepID, :aDepID, :aDepName;
End SQL
```

Este el resultado de esta join:

aName	aEmpDepID	aDepID	aDepName
Alan	10	10	Program
Anne	11	11	Engineering
Bernard	10	10	Program
Mark	12	12	Development
Thomas	10	10	Program

Note que ni los empleados Philip o Martin ni los departamentos Marketing o Quality aparecen en la join resultante porque:

- Philip no tiene un departamento asociado con su nombre (valor NULL),
- El ID de departamento de Martin no existe en la tabla Departamentos,
- No hay empleado asociado al departamento de ID 13,
- El departamento de Marketing no tiene un ID asociado (valor NULL).

Combinaciones externas con dos tablas

Puede generar combinaciones externas con 4D (OUTER JOINs). En una combinación externa, no es necesario que haya una correspondencia entre las líneas de las tablas combinadas. La tabla resultante contiene todas las líneas de las tablas (o de al menos una de las tablas combinadas), incluso si no hay líneas correspondientes. Esto significa que toda la información de una tabla puede ser utilizada, aunque las líneas no se llenan por completo entre las diferentes tablas unidas.

Hay tres tipos de combinaciones externas, definidas por las palabras claves LEFT, RIGHT y FULL. LEFT y RIGHT se utilizan para indicar la tabla (ubicada a la izquierda o a la derecha de la palabra clave JOIN) en la que todos los datos deben ser procesados. FULL indica una join externa bilateral.

Nota: sólo las combinaciones externas explícitas son soportadas por 4D.

Con combinaciones externas de dos tablas, las condiciones pueden ser complejas, pero siempre deben basarse en una comparación de igualdad entre las columnas incluidas en la unión. Por ejemplo, no es posible utilizar el operador >= en una condición de combinación explícita. Todo tipo de comparación se puede utilizar en una combinación implícita. Internamente, las comparaciones de igualdad se llevan a cabo directamente por el motor 4D, lo que asegura una ejecución rápida.

Combinaciones externas izquierdas (Left outer joins)

El resultado de una join externa izquierda (o left join) siempre contiene todos los registros de la tabla situada a la izquierda de la palabra clave, incluso si la condición de join no encuentra un registro correspondiente en la tabla a la derecha. Esto significa que para cada línea de la tabla de la izquierda, donde la búsqueda no encuentra ninguna línea correspondiente en la tabla de la derecha, la join contendrá la línea con valores NULL para cada columna de la tabla de la derecha. En otras palabras, una join externa izquierda devuelve todas las líneas de la tabla de la izquierda, además de las de la tabla de la derecha que correspondan a la condición de join (o NULL si ninguna corresponde). Tenga en cuenta que si la tabla de la derecha contiene más de una línea que corresponde con el predicado de la join para una línea de la tabla de la izquierda, los valores de la tabla izquierda se repetirán para cada línea distinta de la tabla derecha.

Este es un ejemplo de código 4D con una join externa izquierda:

```
ARRAY TEXT(aName;0)
ARRAY TEXT(aDepName;0)
ARRAY INTEGER(aEmpDepID;0
ARRAY INTEGER(aDepID;0)
Begin SQL
        SELECT Employees.name, Employees.depID, Departments.depID, Departments.depName
            FROM Employees
            LEFT OUTER JOIN Departments
                ON Employees.DepID = Departments.DepID;
                INTO :aName, :aEmpDepID, :aDepID, :aDepName;
End SQL
```

Este es el resultado de esta join con nuestra base de ejemplo (las líneas adicionales se muestran en rojo):

aName	aEmpDepID	aDepID	aDepName
Alan	10	10	Program
Anne	11	11	Engineering
Bernard	10	10	Program
Mark	12	12	Development
Thomas	10	10	Program
Martin	15	NULL	NULL
Philip	NULL	NULL	NULL

Combinaciones externas derechas (Right outer joins)

Una combinaciónexterna derecha es el opuesto exacto de una join externa izquierda. Su resultado siempre contiene todos los registros de la tabla ubicada a la derecha de la palabra clave JOIN incluso si la condición join no encuentra un registro correspondiente en la tabla izquierda.

Este es un ejemplo de código 4D con una join externa derecha:

```
ARRAY TEXT(aName;0)
ARRAY TEXT(aDepName;0)
ARRAY INTEGER(aEmpDepID;0
ARRAY INTEGER(aDepID;0)
Begin SQL
        SELECT Employees.name, Employees.depID, Departments.depID, Departments.depName
            FROM Employees
            RIGHT OUTER JOIN Departments
                ON Employees.DepID = Departments.DepID;
                INTO :aName, :aEmpDepID, :aDepID, :aDepName;
End SQL
```

Este es el resultado de esta join con nuestra base de ejemplo (las líneas adicionales están en rojo):

aName	aEmpDepID	aDepID	aDepName
Alan	10	10	Program
Anne	11	11	Engineering
Bernard	10	10	Program
Mark	12	12	Development
Thomas	10	10	Program
NULL	NULL	NULL	Marketing
NULL	NULL	13	Quality

Combinaciones externas bilaterales (Full outer joins)

Una join externa bilateral combina los resultados de una join externa izquierda y de una join externa derecha. La tabla join resultante contiene todos los registros de las tablas izquierda y derecha y llena los campos faltantes de cada lado valores NULL.

Este es un ejemplo de código 4D con una join externa bilateral:

```
ARRAY TEXT(aName;0)
ARRAY TEXT(aDepName;0)
ARRAY INTEGER(aEmpDepID;0
ARRAY INTEGER(aDepID;0)
Begin SQL
        SELECT Employees.name, Employees.depID, Departments.depID, Departments.depName
            FROM Employees
            FULL OUTER JOIN Departments
                ON Employees.DepID = Departments.DepID;
                INTO :aName, :aEmpDepID, :aDepID, :aDepName;
End SQL
```

Este es el resultado de esta join con nuestra base de ejemplo (las líneas adicionales se muestran en rojo):

aName	aEmpDepID	aDepID	aDepName
Alan	10	10	Program
Anne	11	11	Engineering
Bernard	10	10	Program
Mark	12	12	Development
Thomas	10	10	Program
Martin	15	NULL	NULL
Philip	NULL	NULL	NULL
NULL	NULL	NULL	Marketing
NULL	NULL	13	Quality

Combinaciones externas con tres o más tablas

A partir de 4D v15 R4, el servidor SQL integrado extiende el soporte de combinaciones externas SQL a búsquedas que involucran tres o más tablas. Esta implementación específica tiene sus propias reglas y limitaciones, que se describen en esta sección.

Como las combinaciones externas de dos tablas, las combinaciones externas con tres o más tablas pueden ser LEFT, RIGHT, o FULL. Para obtener información general sobre las combinaciones externas, consulte el párrafo anterior **Combinaciones externas con dos tablas**.

A diferencia de las combinaciones externas con dos tablas, las combinaciones externas con tres o más tablas soportan varios operadores de comparación, además de los de igualdad (=): <, >, >=, o <=. Estos operadores se pueden combinar dentro de las cláusulas ON.

Reglas básicas

- Cada cláusula ON combinación externa explícita debe referenciar exactamente dos tablas, ni más ni menos. Cada tabla combinada debe ser referenciada al menos una vez en las cláusulas ON.
- Una de las tablas debe provenir de la parte izquierda inmediata de la cláusula JOIN y la otra, de la derecha inmediata.

Por ejemplo, la siguiente búsqueda se ejecutará con éxito:

```
SELECT * FROM T1
LEFT JOIN
(T2 LEFT JOIN T3 ON T2.ID=T3.ID) -- acá T2 está a la izquierda y T3 a la derecha
ON T1.ID=T3.ID -- acá T1 está a la izquierda y T3 a la derecha
```

Con nuestras tres tablas, este ejemplo podría ser:

```
ARRAY TEXT(aName;0)
ARRAY TEXT(aDepName;0)
ARRAY TEXT(aCityName;0)
ARRAY INTEGER(aEmpDepID;0)
ARRAY INTEGER(aEmpCityID;0
ARRAY INTEGER(aDepID;0)
ARRAY INTEGER(aCityID;0)
Begin SQL
        SELECT Employees.name, Employees.depID, Employees.cityID, Departments.depID,
Departments.depName, Cities.cityID, Cities.cityName
            FROM Departments
                LEFT JOIN
                (Employees LEFT JOIN Cities ON Employees.cityID=Cities.cityID)
                ON Departments.depID=Employees.depID
                INTO :aName, :aEmpDepID, :aEmpCityID, :aDepID, :aDepName, :aCityID, :aCityName;
End SQL
```

Estos son los resultados:

aName	aEmpDepID	aEmpCityID	aDepID	aDepName	aCityID	aCityName
Alan	10	30	10	Program	NULL	NULL
Bernard	10	33	10	Program	30	Paris
Anne	11	39	11	Engineering	33	New York
Fabrice	12	35	12	Development	NULL	NULL
Thomas	10	NULL	10	Program	NULL	NULL
NULL	NULL	NULL	NULL	Marketing	NULL	NULL
NULL	NULL	NULL	13	Quality	NULL	NULL

Por otra parte, las siguientes tres búsquedas se rechazarán ya que violan ciertas reglas:

```
SELECT * FROM T1
LEFT JOIN
(T2 LEFT JOIN T3 ON T2.ID=T1.ID) -- acá T2 está a la izquierda pero T1 no está presente en la derecha inmediata
ON T1.ID=T3.ID
```

```
SELECT * FROM
(T1 LEFT JOIN T2 ON T1.ID=T2.ID)
LEFT JOIN
(T3 LEFT JOIN T4 ON T3.ID=T4.ID)
ON T3.Name=T4.Name -- acá T3 y T4 vienen del lado derecho de la cláusula JOIN y ninguna tabla del lado izquierdo
```

```
SELECT * FROM T1
LEFT JOIN
(T2 LEFT JOIN T3 ON T2.ID=T3.ID)
ON T1.ID=T3.ID AND T1.ID=T2.ID -- acá más de dos tablas están siendo utilizadas en la cláusula ON: T1, T2 y T3
```

Soporte de la condición ON

En general, si las tablas (Tx1, Tx2 ..., TXN) a la izquierda de la cláusula JOIN y las tablas (Ty1, Ty2 ..., Tym) a la derecha se combinaron, a continuación la expresión ON debe hacer referencia

exactamente a una tabla de la izquierda Txa y exactamente una tabla Tyb a la derecha.

	No soportado en la cláusula ON	Soportado en la cláusula ON
Operaciones booleanas	OR	AND y NOT
Predicado y funciones	IS NULL, **COALESCE**	Todos los demás predicados y funciones integradas (se pueden usar en cualquier combinación deseada)
Referencias a variables 4D	-	Soportadas sin restricciones
Llamadas a métodos 4D	Cuando la cláusula JOIN actual de ambos lados izquierdo o derecho es una combinación externa explícita	Todos los demás casos (ver el ejemplo a continuación)

El siguiente ejemplo con una llamada al método 4D es soportado porque no hay sub combinaciones no-internas a combinar:

```
SELECT * FROM T1
LEFT JOIN T2
ON T1.ID={FN My4DCall (T2.ID) AS INT32}
```

Por otra parte, este ejemplo de llamada al método 4D no se soporta porque no se están combinando sub combinaciones no internas:

```
SELECT * FROM
(T1 LEFT JOIN T2 ON T1.ID=T2.ID)
LEFT JOIN -- Ambos lados izquierdo y derecho de esta cláusula de combinación contienen combinaciones
LEFT explícitas
(T3 LEFT JOIN T4 ON T3.ID=T4.ID)
ON T1.ID={FN My4DCall (T4.ID) AS INT32} -- no están siendo combinadas sub combinaciones internas
```

Limitaciones generales

- Referencias a **Vistas SQL** no están permitidas en la declaración de combinación explícita
- No se soportan sub búsquedas que utilizan combinaciones externas. Las siguientes serán rechazadas:

```
SELECT T2.ID FROM T2
WHERE T2.ID=(
SELECT COUNT ( * ) FROM
(T1 LEFT JOIN T3 ON T1.ID=T3.ID)
RIGHT JOIN T4 ON T3.ID=T4.ID)
```

Comandos SQL

Comandos SQL
SELECT
INSERT
UPDATE
DELETE
CREATE DATABASE
USE DATABASE
ALTER DATABASE
CREATE TABLE
ALTER TABLE
DROP TABLE
CREATE INDEX
DROP INDEX
LOCK TABLE
UNLOCK TABLE
EXECUTE IMMEDIATE
CREATE SCHEMA
ALTER SCHEMA
DROP SCHEMA
CREATE VIEW
DROP VIEW
GRANT
REVOKE
REPLICATE
SYNCHRONIZE

Comandos SQL

Los comandos SQL generalmente se agrupan en dos categorías:

- Comandos de manipulación de datos, que se utilizan para obtener, agregar, quitar y/o modificar la información de base de datos. Específicamente, esto se refiere a los comandos *SELECT*, **INSERT**, *UPDATE* y *DELETE*.
- Comandos de definición de datos, que se utilizan para crear o eliminar objetos de estructura de la base de datos o las bases de datos. Específicamente, se refiere a los comandos **CREATE DATABASE**, *CREATE TABLE*, *ALTER TABLE*, *DROP INDEX*, *DROP TABLE* o *CREATE SCHEMA*.

En la sintaxis, los nombres y las palabras claves aparecen en negrita y se pasan "tal cual". Otros elementos aparecen en cursiva y se detallan en el capítulo . Las palabras claves y/o las cláusulas que se pasan entre corchetes [] son opcionales. El carácter barra vertical | separa las diferentes alternativas disponibles. Cuando los elementos se pasan entre llaves (), separados por barras verticales, esto indica que sólo un elemento del conjunto debe pasarse.

SELECT

SELECT [**ALL** | **DISTINCT**]

{* | *select_elemento*, ..., *select_elemento*}

FROM *ref_tabla*, ..., *ref_tabla*

[**WHERE** *criterio_búsqueda*]

[**ORDER BY** *lista_orden*]

[**GROUP BY** *lista_orden*]

[**HAVING** *criterio_búsqueda*]

[**LIMIT** {*ref_lenguaje_4d*|*número_entero* | **ALL**}]

[**OFFSET** *ref_lenguaje_4d*|*número_entero*]

[**INTO** {*ref_lenguaje_4d*, ..., *ref_lenguaje_4d*}]

[**FOR UPDATE**]

Descripción

El comando **SELECT** se utiliza para recuperar datos de una o más tablas.
Si pasa *, se devuelven todas las columnas, de lo contrario puede pasar uno o más argumentos de tipo *select_elemento* para especificar individualmente cada columna a recuperar (separados por comas). Si agrega la palabra clave opcional **DISTINCT** a la instrucción **SELECT**, los valores duplicados no se devolverán.
No es posible ejecutar búsquedas que contengan a la vez "*" y campos explícitos. Por ejemplo, la siguiente instrucción:

```
SELECT *, SALES, TARGET FROM OFFICES
```

... no se permite, mientras que:

```
SELECT * FROM OFFICES
```

...se permite.

La cláusula **FROM** se utiliza para especificar uno o más argumentos de tipo *ref_tabla* para la o las tabla(s) de las cuales los datos se van a recuperar. Puede pasar un nombre SQL estándar o una cadena. No es posible pasar una expresión de tipo búsqueda en lugar de un nombre de tabla. También puede pasar la palabra clave opcional **AS** para asignar un alias a la columna. Si se pasa esta palabra clave, debe estar seguida por el nombre del alias que también puede ser un nombre SQL o una cadena.

Nota: este comando no soporta campos 4D de tipo Objeto.

La cláusula opcional **WHERE** establece las condiciones que los datos deben cumplir para ser seleccionados. Esto se hace pasando una *condición_búsqueda* que se aplica a los datos recuperados por la cláusula **FROM**. La expresión *condición_búsqueda* siempre devuelve un valor de tipo booleano.

La cláusula opcional **ORDER BY** se puede utilizar para aplicar un criterio *lista_orden* a los datos seleccionados. También puede agregar la palabra clave **ASC** o **DESC** para especificar si desea ordenar de forma ascendente o descendente. Por defecto, se aplica el orden ascendente.

La cláusula opcional **GROUP BY** se puede utilizar para agrupar datos idénticos en función de los criterios pasados en *lista_orden*. Puede pasar varias columnas de grupo. Esta cláusula se puede utilizar para evitar redundancias o calcular una función de adición (**SUM**, *COUNT*, **MIN** o **MAX**) que se aplicarán a estos grupos. También puede agregar la palabra clave **ASC** o **DESC** como con la cláusula **ORDER BY**.

La cláusula opcional **HAVING** se puede utilizar para aplicar un *criterio_búsqueda* a uno de los grupos. La cláusula **HAVING** se puede pasar sin una cláusula **GROUP BY**.

La cláusula **LIMIT** opcional permite restringir el número de datos devueltos a la cantidad definida por la variable *ref_lenguaje_4d* o el *número_entero*.

La cláusula opcional **OFFSET** permite definir un número (variable *ref_lenguaje_4d* o *número_entero*) de valores a ignorar antes de comenzar a contar para la aplicación de la cláusula **LIMIT**.

La cláusula **INTO** permite indicar las variables *ref_lenguaje_4d* a las cuales los datos se asignarán.

Un comando **SELECT** que especifica una cláusula **FOR UPDATE** intenta bloquear para escritura todos los registros seleccionados. Si al menos un registro no puede bloquearse, todo el comando falla y se devuelve un error. Sin embargo, si todos los registros seleccionados estaban bloqueados, entonces se mantendrán bloqueados hasta que la transacción actual se confirme o se cancele.

Ejemplo 1

Suponga que tiene una base de datos de películas con una tabla que contiene los títulos de las películas, el año en que fueron lanzadas y los boletos vendidos.
Nos gustaría obtener los años a partir de 1979 y la cantidad de entradas vendidas para las películas que vendieron en total menos de 10 millones de boletas. Queremos saltarnos los primeros 5 años y mostrar sólo 10 años, ordenados por año.

```
C_LONGINT($MovieYear;$MinTicketsSold;$StartYear;$EndYear)
ARRAY INTEGER(aMovieYear;0)
ARRAY LONGINT(aTicketsSold;0)
$MovieYear:=1979
$MinTicketsSold:=10000000
$StartYear:=5
$EndYear:=10

Begin SQL
   SELECT Year_of_Movie, SUM(Tickets_Sold)
   FROM MOVIES
   WHERE Year_of_Movie >= :$MovieYear
   GROUP BY Year_of_Movie
   HAVING SUM(Tickets_Sold) < :$MinTicketsSold
   ORDER BY 1
   LIMIT :$EndYear
   OFFSET :$StartYear
   INTO :aMovieYear, :aTicketsSold;
End SQL
```

Ejemplo 2

Este ejemplo utiliza una combinación de criterios de búsqueda:

```
SELECT supplier_id
FROM suppliers
WHERE (name = 'CANON')
OR (name = 'Hewlett Packard' AND city = 'New York')
OR (name = 'Firewall' AND status = 'Closed' AND city = 'Chicago');
```

Ejemplo 3

Dada la tabla VENDEDORES donde QUOTA es la cantidad de ventas esperada para un representante de ventas y VENTAS la cantidad de ventas efectivamente realizadas.

```
ARRAY REAL(arrMin_Values;0)
ARRAY REAL(arrMax_Values;0)
ARRAY REAL(arrTotal_Values;0)
Begin SQL
    SELECT MIN ( ( VENTAS * 100 ) / QUOTA ),
    MAX( (VENTAS * 100 ) / QUOTA ),
    SUM( QUOTA ) - SUM ( VENTAS )
    FROM VENDEDORES
    INTO :arrMin_Values, :arrMax_Values, :arrTotal_Values;
End SQL
```

Ejemplo 4

Este ejemplo busca todos los actores que nacieron en una ciudad determinada:

```
ARRAY TEXT(aActorName;0)
ARRAY TEXT(aCityName;0)
Begin SQL
    SELECT ACTORS.FirstName, CITIES.City_Name
    FROM ACTORS AS 'Act', CITIES AS 'Cit'
    WHERE Act.Birth_City_ID=Cit.City_ID
    ORDER BY 2 ASC
    INTO : aActorName, : aCityName;
End SQL
```

INSERT

INSERT INTO {*nom_sql* | *cadena_sql*}
[(*ref_columna*, ..., *ref_columna*)]
[**VALUES**({[**INFILE**]*expresión_aritmética* |**NULL**}, ..., {[**INFILE**]*expresión_aritmética* |**NULL**};)
|*sub_búsqueda*]

Descripción

El comando **INSERT** se utiliza para añadir datos a una tabla existente. La tabla en donde los datos se añaden se pasa utilizando un argumento de tipo *nom_sql* o *cadena_sql*. Los argumentos opcionales tipo *ref_columna* permiten definir las columnas en las cuales insertar los valores. Si no se pasa *ref_columna*, los valores se insertarán en el mismo orden que en la base (el primer valor pasado se insertará en la primera columna, el segundo valor en la segunda columna, y así sucesivamente).

Nota: este comando no soporta campos 4D de tipo Objeto.

La palabra clave **VALUES** se utiliza para pasar el o los valor(es) a insertar en la(s) columna(s) especificada(s). Puede pasar una *expresión_aritmética* o **NULL**. Por otra parte, una *subconsulta* se puede pasar en la palabra clave **VALUES** con el fin de insertar una selección de datos que se pasan como los valores.

El número de valores pasados vía la palabra clave **VALUES** debe coincidir con el número de columnas especificado por el argumento de tipo *ref_columna* pasado y cada uno de ellos también debe coincidir con el tipo de datos de la columna correspondiente o al menos ser convertible a ese tipo de datos.
La palabra clave **INFILE** le permite utilizar el contenido de un archivo externo para especificar los valores de un nuevo registro. Esta palabra clave sólo se debe utilizar con expresiones de tipo VARCHAR. Cuando se pasa la palabra clave **INFILE**, el valor *expresión_aritmética* se evalúa como una ruta del archivo, y si se encuentra el archivo, el contenido del archivo se inserta en la columna correspondiente. Sólo los campos de tipo texto o BLOB pueden recibir valores de un **INFILE**. El contenido del archivo se transfiere como datos brutos, sin interpretación.
El archivo buscado debe estar en el equipo que aloja el motor SQL, incluso si la consulta viene de un cliente remoto. Del mismo modo, la ruta debe expresarse respetando la sintaxis del sistema operativo del motor de SQL. Puede ser absoluta o relativa.

El comando **INSERT** es soportado en búsquedas mono y multilíneas. Sin embargo, una instrucción **INSERT** multilíneas no permite efectuar operaciones UNION y JOIN.
El motor de 4D permite la inserción de valores multilíneas, que pueden simplificar y optimizar el código, en particular, al introducir grandes cantidades de datos. La sintaxis de las inserciones multilíneas es del tipo:

```
INSERT INTO {sql_name | sql_string}
[(column_ref, ..., column_ref)]
VALUES(arithmetic_expression, ..., arithmetic_expression), ..., (arithmetic_expression, ...,
arithmetic_expression);
```

Esta sintaxis se ilustra en los ejemplos 3 y 4.

Ejemplo 1

Este ejemplo simple permite insertar una selección de la table2 en la table1:

```
INSERT INTO table1 (SELECT * FROM table2)
```

Ejemplo 2

Este ejemplo crea una tabla y luego inserta los valores en ella:

```
CREATE TABLE ACTOR_FANS
(ID INT32, Nom VARCHAR);
INSERT INTO ACTOR_FANS
(ID, Nom)
VALUES (1, 'Francis');
```

Ejemplo 3

La sintaxis multilíneas permite evitar la repetición tediosa de líneas:

```
INSERT INTO MiTabla
(Campo1,Campo2,CampoBol,CampoHora,CampoInfo)
VALUES
(1,1,1,'11/01/01','11:01:01','Primera línea'),
(2,2,0,'12/01/02','12:02:02','Segunda línea'),
(3,3,1,'13/01/03','13:03:03','Tercera línea'),
.....
(7,7,1,'17/01/07','17:07:07','Séptima línea');
```

Ejemplo 4

También puede utilizar las variables o arrays 4D con la sintaxis multilíneas:

```
INSERT INTO MiTabla
(Campo1,Campo2,CampoBool,CampoFecha,CampoHora, CampoInfo)
VALUES
( :vArrId, :vArrIdx, :vArrbool, :vArrdate, :vArrL, :vArrText);
```

Nota: no es posible combinar las variables simples y los arrays en la misma instrucción **INSERT**.

UPDATE

UPDATE {*nom_sql*| *cadena_sql*}
SET *nom_sql* = {*expresión_aritmética* |**NULL**}, ..., *nom_sql* = {*expresión_aritmética* |**NULL**}
[**WHERE** *criterio_búsqueda*]

Descripción

El comando **UPDATE** permite modificar los datos contenidos en una tabla indicada por el argumento *nom_sql* o *cadena_sql*.
La cláusula **SET** se utiliza para asignar nuevos valores (vía una *expresión_aritmética* o el valor NULL).
La cláusula opcional **WHERE** se utiliza para especificar cuales datos (los que cumplen la condición *criterio_búsqueda*) se deben actualizar. Si no se pasa, todos los datos de la tabla se actualizarán con los nuevos valores pasados en la cláusula **SET**.

Nota: este comando no soporta los campos 4D de tipo Objeto.
El comando **UPDATE** es soportado para consultas y subconsultas, sin embargo, una instrucción **UPDATE** posicionada no es compatible.
Las actualizaciones en cascada son implementadas en 4D, pero las reglas de supresión **SET NULL** y **SET DEFAULT** no son soportadas.

Ejemplo

El siguiente ejemplo modifica la tabla MOVIES de manera que los boletos vendidos para la película "Air Force One" sean 3 500 000:

```
UPDATE MOVIES
SET Tickets_Sold = 3500000
WHERE TITLE = 'Air Force One';
```

DELETE

DELETE FROM {*nom_sql* | *cadena_sql*}

[**WHERE** *criterio_búsqueda*]

Descripción

El comando **DELETE** se puede utilizar para suprimir todo o una parte de los datos de la tabla indicada por el argumento *nom_sql* o *cadena_sql* después de la palabra clave **FROM**.

La cláusula **WHERE** opcional se utiliza para indicar que parte de los datos (los que cumplen la condición *criterio_búsqueda*) se van a eliminar. Si no se pasa, todos los datos de la tabla serán suprimidos.

No está soportada la instrucción **DELETE** posicionada. Las supresiones en cascada son implementadas en 4D, pero las reglas de supresión **SET DEFAULT** y **SET NULL** no son soportadas.

Nota: este comando no soporta campos 4D de tipo objeto.

Ejemplo

Este es un ejemplo que elimina todas las películas que estrenaron en el año 2000 o antes de la tabla MOVIES:

```
DELETE FROM MOVIES
WHERE Año_estreno <= 2000;
```

CREATE DATABASE

CREATE DATABASE [**IF NOT EXISTS**] **DATAFILE** *<Ruta de acceso completa>*

Descripción

El comando **CREATE DATABASE** le permite crear una nueva base de datos externa (archivos db .4db y .4dd) en una ubicación específica.

Si se pasa la restricción **IF NOT EXISTS**, la base de datos no se crea y ningún error se genera si una base con el mismo nombre ya existe en la ubicación especificada.
Si no se pasa la restricción **IF NOT EXISTS**, la base de datos no se crea y se muestra el mensaje de error "Esta base de datos ya existe. No se pudo ejecutar el comando CREATE DATABASE." si una base de datos con el mismo nombre ya existe en la ubicación especificada.
La cláusula **DATAFILE** permite especificar el nombre completo (ruta de acceso completa + nombre) de la nueva base de datos externa. Debe pasar el nombre del archivo de estructura. El programa añadirá automáticamente la extensión ".4db" al archivo si no está y crea el archivo de datos. La ruta puede expresarse en sintaxis POSIX o en la sintaxis del sistema, puede ser absoluta o relativa al archivo de estructura de la base 4D principal.

- Sintaxis POSIX (tipo URL): los nombres de las carpetas están separados por una barra oblicua ("/"), independientemente de la plataforma que utilice, por ejemplo:
  .../extdatabases/myDB.4db
  Para una ruta absoluta, pase en la primera posición el nombre del volumen seguido de dos puntos, por ejemplo: "C:/test/extdatabases/myDB.4db"

- Sintaxis sistema: ruta de acceso respetando la sintaxis de la plataforma actual, por ejemplo:
  - (Mac OS) Unidad:Applications:miserv:basesexternas:mibase.4db
  - (Windows) C:ApplicationsmyservextdatabasesmyDB.4db

Después de la ejecución exitosa del comando **CREATE DATABASE**, la nueva base de datos creada no se convierte automáticamente en la base actual. Para hacer esto, debe declararla explícitamente como la base actual con el comando **USE DATABASE**.

Sobre las bases externas

Una base externa es una base 4D independiente de la base 4D principal, pero con la que puede trabajar desde la base 4D principal utilizando el motor SQL de 4D. Usar una base externa significa designar temporalmente esta base como base actual, es decir, como la base de objetivo de las consultas SQL ejecutadas por 4D. Por defecto, la base actual es la base principal.

Puede crear una base de datos externa directamente desde la base principal con el comando **CREATE DATABASE**. Una vez creada, una base externa puede ser designada como la base actual con el comando **USE DATABASE**. A continuación, se puede modificar a través de comandos estándar SQL (**CREATE TABLE, ALTER TABLE**, etc.) y puede almacenar datos en ella. La función **DATABASE_PATH** permite conocer la base de datos actual en cualquier momento.
El interés principal de las bases externas reside en el hecho de que pueden ser creadas y manipuladas vía componentes 4D. Esto permite el desarrollo de componentes que son capaces de crear tablas y campos de acuerdo a sus necesidades.

Nota: una base externa es una base 4D estándar. Se puede abrir y manipular como la base principal mediante una aplicación 4D o 4D Server. Por el contrario, toda base 4D estándar se puede utilizar como base externa. Sin embargo, es imperativo que no active el sistema de gestión

de acceso (mediante la asignación de una contraseña al Diseñador) en una base externa, de lo contrario ya no podrá tener acceso a ella a través del comando **USE DATABASE**.

Ejemplo 1

Creación de archivos de base externa ExternalDB.4DB y ExternalDB.4DD en C:/MiBase/:

```
Begin SQL
   CREATE DATABASE IF NOT EXISTS DATAFILE 'C:/MiBase/ExternalDB';
End SQL
```

Ejemplo 2

Creación de archivos de base externa TestDB.4DB y TestDB.4DD junto al archivo de estructura de la base principal:

```
Begin SQL
   CREATE DATABASE IF NOT EXISTS DATAFILE 'TestDB';
End SQL
```

Ejemplo 3

Creación de los archivos de base externa External.4DB y External.4DD en la ubicación definida por el usuario:

```
C_TEXT($path)
$path:=Select folder("Carpeta de destino de la base externa:")
$path:=$path+"External"
Begin SQL
   CREATE DATABASE DATAFILE <<$path>>;
End SQL
```

USE DATABASE

USE [**LOCAL** | **REMOTE**] **DATABASE**
{**DATAFILE** <*Ruta de acceso completa*> | **SQL_INTERNAL** | **DEFAULT**}
[**AUTO_CLOSE**]

Descripción

El comando **USE DATABASE** se utiliza para designar una base externa como base de datos actual, en otras palabras, la base a la cual se dirigirán las próximas consultas SQL en el proceso actual. Todos los tipos de consultas SQL concernientes: consultas incluidas en la estructura **Begin SQL/End SQL**, comandos **SQL EXECUTE** o **EXECUTE SCRIPT**, etc.

Nota: para mayor información sobre bases externas, consulte la descripción del comando**CREATE DATABASE**.

- Si trabaja en una configuración monopuesto, la base externa debe estar ubicada en la misma máquina que su 4D.
- Si trabaja en modo remoto, la base externa puede estar ubicada en la máquina local o en la máquina 4D Server.

Si utiliza 4D en modo remoto, la palabra clave **REMOTE** permite designar una base externa ubicada en 4D Server.

Por razones de seguridad, este mecanismo sólo funciona con conexiones remotas nativas, es decir, en el contexto de una base 4D remota conectada a 4D Server. Las conexiones vía ODBC o pass-through no están permitidas.
Si no se especifica la palabra clave, la opción **LOCAL** se utiliza por defecto. Si está utilizando 4D en modo local, las palabras claves **REMOTE** y **LOCAL** se ignoran: las conexiones son siempre locales.

Para designar una base externa a utilizar, pase su ruta completa (ruta de acceso + nombre) en la cláusula **DATAFILE**. La ruta puede expresarse en la sintaxis POSIX o en la sintaxis del sistema. Puede ser absoluta o relativa al archivo de estructura de la base 4D principal.

En modo remoto, si se pasa la palabra clave **REMOTE**, este parámetro designa la ruta de acceso de la base a partir de la máquina servidor. Si se omite o si se pasa la palabra clave **LOCAL**, este parámetro designa la ruta de acceso de la base en la máquina 4D local.

Importante: debe designar una base 4D externa válida y en la cual el sistema de control de acceso no esté activado (mediante la asignación de una contraseña al Diseñador). De lo contrario, se genera un error.

Con el fin de restablecer la base principal como la base actual, ejecute el comando al pasar la palabra clave **SQL_INTERNAL** o **DEFAULT**.

Pase **AUTO_CLOSE** si desea cerrar físicamente la base externa después de su uso, es decir, cuando usted cambia la base actual. De hecho, la apertura de una base externa es una operación que requiere un poco de tiempo, por razones de optimización 4D mantiene en la memoria la información relativa a las bases externas abiertas durante la sesión usuario. Esta información se mantiene en memoria hasta que se lance la aplicación 4D. Las aperturas posteriores de la misma base externa, son por lo tanto más rápidas. Sin embargo, esto impide el intercambio de bases externas entre varias aplicaciones 4D porque la base de datos externa sigue estando abierta en lectura/escritura para la primera aplicación que la utilice. Si varias aplicaciones 4D deben utilizar la misma base externa al mismo tiempo, pase la palabra clave **AUTO_CLOSE** para liberar físicamente la base externa después de su uso.

Esta restricción no aplica a los procesos de la misma aplicación: diferentes procesos de una aplicación siempre pueden acceder a la misma base externa en lectura/escritura sin que sea

necesario forzar su cierre.
Tenga en cuenta que cuando varios procesos utilizan la misma base externa, es físicamente liberada sólo hasta que el último proceso que utiliza se cierre, incluso cuando se pasa la opción **AUTO_CLOSE**. Debe tener en cuenta este funcionamiento para las operaciones que implican el intercambio entre aplicaciones o supresión de bases externas.

Ejemplo

Uso de una base externa para una búsqueda luego regresa a la base principal:

```
Begin SQL
    USE DATABASE DATAFILE 'C:/MiBase/Noms'
    SELECT Name FROM emp INTO :tNoms1
    USE DATABASE SQL_INTERNAL
End SQL
```

ALTER DATABASE

ALTER DATABASE {**ENABLE** | **DISABLE**} {**INDEXES** | **CONSTRAINTS | TRIGGERS**}

Descripción

El comando **ALTER DATABASE** activa o deshabilita las opciones SQL de la base actual para la sesión actual. es decir par todos los usuarios y procesos hasta que la base se reinicie.

Este comando está diseñado para permitirle deshabilitar temporalmente las opciones SQL con el fin de acelerar ciertas operaciones que consumen una gran cantidad de recursos. Por ejemplo, desactivar los índices y las restricciones antes de comenzar la importación de una gran cantidad de datos puede reducir significativamente la duración de la importación.

Tenga en cuenta que las restricciones incluyen las llaves primarias y las llaves foráneas, así como los atributos de unicidad y de nulidad.

Si prefiere gestionar los triggers individualmente para cada tabla, debe utilizar **ALTER TABLE**.

Ejemplo

Ejemplo de una importación con desactivación temporal de todas las opciones SQL:

```
Begin SQL
    ALTER DATABASE DISABLE INDEXES;
    ALTER DATABASE DISABLE CONSTRAINTS;
    ALTER DATABASE DISABLE TRIGGERS;
End SQL
SQL EXECUTE SCRIPT("C:\\Exported_data\\Export.sql";SQL On error continue)
Begin SQL
    ALTER DATABASE ENABLE INDEXES;
    ALTER DATABASE ENABLE CONSTRAINTS;
    ALTER DATABASE ENABLE TRIGGERS;
End SQL
```

CREATE TABLE

CREATE TABLE [**IF NOT EXISTS**] [*nom_sql.*]*nom_sql*({*definicion_columna* |*restriccion_tabla*} [**PRIMARY KEY**], ... , {*definicion_columna* |*restriccion_tabla*}[**PRIMARY KEY**]) [{**ENABLE** | **DISABLE**} **REPLICATE**]

Descripción

El comando **CREATE TABLE** se utiliza para crear una tabla llamada *nom_sql* con los campos especificados al pasar una o más *definicion_columna* y/o *restriccion_tabla*. Si se pasa la restricción **IF NOT EXISTS**, la tabla sólo se crea cuando no existe una tabla con el mismo nombre en la base. De lo contrario, no se crea y no se genera ningún error.

El primer parámetro *nom_sql* (opcional) permite designar el esquema SQL al cual quiere asignar la tabla. Si no pasa este parámetro o si pasa el nombre de un esquema que no existe, el esquema por defecto se le asigna automáticamente, llamado "DEFAULT_SCHEMA". Para mayor información sobre los esquemas SQL, consulte la sección **Implementaciones del motor SQL de 4D**.

Nota: también es posible asignar una tabla a un esquema SQL utilizando el menú pop-up "Esquemas" en el Inspector de tablas de 4D. Este menú contiene la lista de esquemas definidos en la base.

Una *definicion_columna* contiene el nombre (*nom_sql*) y el tipo de datos (*tipo_datos_sql*) de una columna y una *restriccion_tabla* restringe los valores que una tabla puede almacenar.

Nota: este comando no permite añadir un campo (columna) de tipo Objeto.

La palabra clave **PRIMARY KEY** se utiliza para especificar la llave primaria cuando se crea la tabla. Para mayor información sobre las llaves primarias, consulte la sección **Implementaciones del motor SQL de 4D**.

Las palabras claves **ENABLE REPLICATE** y **DISABLE REPLICATE** se utilizan para activar o desactivar el mecanismo que permite la replicación de la tabla (ver la sección **Replicación vía SQL**).

Ejemplo 1

Este es un ejemplo simple de creación de una tabla con dos columnas:

```
CREATE TABLE ACTOR_FANS
(ID INT32, Nom VARCHAR);
```

Ejemplo 2

Este ejemplo crea la misma tabla pero sólo si no hay una tabla con el mismo nombre:

```
CREATE TABLE IF NOT EXISTS ACTOR_FANS
(ID INT32, Nom VARCHAR);
```

Ejemplo 3

Este ejemplo crea la tabla "Preferences" y se le asigna el esquema "Control":

```
CREATE TABLE Control.Preferences
(ID INT32, Value VARCHAR);
```

ALTER TABLE

ALTER TABLE *nom_sql*
{**ADD** [**TRAILING**] *definición_columna* [**PRIMARY KEY**] |
[**TRAILING**] |
DROP *nom_sql* |
ADD *definición_llave_primaria* |
DROP PRIMARY KEY |
ADD *definición_llave_foránea* |
DROP CONSTRAINT *nom_sql* |
[{**ENABLE** | **DISABLE**} **REPLICATE**] |
[{**ENABLE** | **DISABLE**} **LOG**] |
[**MODIFY** *sql_name* {**ENABLE** | **DISABLE**} **AUTO_INCREMENT**] |
[**MODIFY** *sql_name* {**ENABLE** | **DISABLE**} **AUTO_GENERATE**] |
[{**ENABLE** | **DISABLE**} **TRIGGERS**] |
SET SCHEMA *nom_sql*}

Descripción

El comando *ALTER TABLE* se utiliza para modificar una tabla existente (*nom_sql*). Puede efectuar una de las siguientes acciones:
Pasar **ADD** *definición_columna* añade una columna a la tabla.
La palabra clave **TRAILING** (debe ubicarse en frente de *definición_columna* si se utiliza) forza la creación de la columna luego de la última columna existente de la tabla en el archivo de estructura. Esta opción es útil cuando las columnas que contienen los datos se han eliminado de la tabla (sin que los datos sean borrados), para evitar que los datos existentes sean reasignados a la nueva columna.

Nota: este comando no permite añadir un campo (columna) de tipo Objeto.

La palabra clave **PRIMARY KEY** se utiliza para definir la llave primaria cuando se añade una columna.
Pasar **DROP** *nom_sql* elimina la columna *nom_sql* de la tabla.
Pasar **ADD** *definición_llave_primaria* añade una llave primaria (**PRIMARY KEY**) a la tabla.
Pasar **DROP PRIMARY KEY** elimina la llave primaria (**PRIMARY KEY**) de la tabla.
Pasar **ADD** *definición_llave_foránea* añade una llave foránea (**FOREIGN KEY**) a la tabla.
Pasar **DROP CONSTRAINT** *nom_sql* elimina la restricción especificada por nom_sql de la tabla.
Pasar **ENABLE REPLICATE** o **DISABLE REPLICATE** activa o desactiva el mecanismo que permite la replicación de la tabla (ver la sección **Replicación vía SQL**).
Pasar **ENABLE LOG** o **DISABLE LOG** activa o desactiva el historial de la tabla.
Pasar **ENABLE AUTO_INCREMENT** o **DISABLE AUTO_INCREMENT** activa o desactiva la opción "Autoincrement" de los campos de tipo entero largo.
Pasar **ENABLE AUTO_GENERATE** o **DISABLE AUTO_GENERATE** activa o desactiva la opción "Auto UUID" para los campos alfa de tipo UUID. En los dos casos, debe pasar la palabra clave **MODIFY** seguida del *nom_sql* de la columna a modificar.
Pasar **ENABLE TRIGGERS** o **DISABLE TRIGGERS** activa o desactiva los triggers para la tabla. Si desea gestionar los triggers globalmente a nivel de la base, debe utilizar **ALTER DATABASE**.
Pasar **SET SCHEMA** *nom_sql* transfiere la tabla al esquema *nom_sql*.

El comando devuelve un error:

- cuando el parámetro opcional **ENABLE LOG** se pasa y ninguna llave primaria válida se ha definido,
- si intenta modificar o eliminar la definición de llave primaria de la tabla sin desactivar el historial vía **DISABLE LOG**.

Ejemplo 1

Este ejemplo crea una tabla, inserta un conjunto de valores en ella, luego añade una columna Num_Tel, inserta otro conjunto de valores y luego elimina la columna ID:

```
CREATE TABLE ACTOR_FANS
(ID INT32, Nom VARCHAR);

INSERT INTO ACTOR_FANS
(ID, Nom)
VALUES(1, 'Francis');

ALTER TABLE ACTOR_FANS
ADD Num_Tel VARCHAR;

INSERT INTO ACTOR_FANS
(ID, Nom, Num_Tel)
VALUES (2, 'Florence', '01446677888');

ALTER TABLE ACTOR_FANS
DROP ID;
```

Ejemplo 2

Ejemplo para activar la opción "Autoincrement" del tipo campo Entero largo [Table_1]id:

```
Begin SQL
    ALTER TABLE Table_1 MODIFY id ENABLE AUTO_INCREMENT;
End SQL
```

Desactivando la opción:

```
Begin SQL
    ALTER TABLE Table_1 MODIFY id DISABLE AUTO_INCREMENT;
End SQL
```

Ejemplo para la activación de "Auto UUID" del campo tipo Alfa [Table_1]uid:

```
Begin SQL
    ALTER TABLE Table_1 MODIFY uid ENABLE AUTO_GENERATE;
End SQL
```

Desactivando la opción:

```
Begin SQL
    ALTER TABLE Table_1 MODIFY uid DISABLE AUTO_GENERATE;
End SQL
```

DROP TABLE

DROP TABLE [**IF EXISTS**] *nom_sql*

Descripción

El comando *DROP TABLE* se utiliza para eliminar la tabla *nom_sql* de una base. Cuando se pasa la restricción **IF EXISTS**, si la tabla a eliminar no existe en la base, el comando no hace nada y no se genera ningún error.

Este comando no sólo elimina la tabla de la estructura, sino también sus datos y los índices, triggers y restricciones asociados. No se puede utilizar este comando con una tabla referenciada por una restricción **FOREIGN KEY**.

Nota: debe asegurarse de que cuando se ejecute el comando *DROP TABLE*, no haya registros de la tabla *nom_sql* que se carguen en memoria en modo de escritura. De lo contrario, se genera el error 1272.

Ejemplo 1

Este ejemplo borra la tabla ACTOR_FANS:

```
DROP TABLE ACTOR_FANS
```

Ejemplo 2

Este ejemplo hace lo mismo que el anterior excepto que en este caso, si la tabla ACTOR_FANS no existe, no se genera ningún error:

```
DROP TABLE IF EXISTS ACTOR_FANS
```

CREATE INDEX

CREATE [**UNIQUE**] **INDEX** *nom_sql* **ON** *nom_sql* (*ref_columna*, ... , *ref_columna*)

Descripción

El comando **CREATE INDEX** permite crear un índice (*nom_sql*) en una o más columnas de una tabla existente (*nom_sql*) designada por el o los parámetros *ref_columna*. Los índices son transparentes para los usuarios y permiten acelerar la ejecución de las peticiones.

También puede pasar la palabra clave opcional **UNIQUE** con el fin de crear un índice que no permite los valores duplicados.

Ejemplo

Este es un ejemplo de creación de índices:

```
CREATE INDEX ID_INDEX ON ACTOR_FANS (ID)
```

DROP INDEX

DROP INDEX *nom_sql*

Descripción

El comando *DROP INDEX* permite eliminar de la base el índice existente designado por *nom_sql*.
No puede utilizarse con los índices creados por las restricciones **PRIMARY KEY** o **UNIQUE**.

Ejemplo

Este es un ejemplo simple de eliminación de un índice:

```
DROP INDEX ID_INDEX
```

LOCK TABLE

LOCK TABLE *nom_sql* **IN** {**EXCLUSIVE** | **SHARE**} **MODE**

Descripción

El comando *LOCK TABLE* permite bloquear la tabla *nom_sql* en el modo **EXCLUSIVE** o **SHARE**.
En el modo **EXCLUSIVE**, los datos de la tabla no pueden leerse o modificarse por otra transacción.
En el modo **SHARE**, los datos de la tabla pueden ser leídos por otras transacciones pero las modificaciones siguen siendo prohibidas.

Ejemplo

Este ejemplo bloquea la tabla MOVIES de manera que pueda ser leída pero no modificada por otras transacciones:

```
LOCK TABLE MOVIES IN SHARE MODE
```

UNLOCK TABLE

UNLOCK TABLE *nom_sql*

Descripción

El comando *UNLOCK TABLE* se utiliza para desbloquear la tabla *nom_sql* previamente bloqueada vía el comando *LOCK TABLE*. Este comando no funciona si se pasa dentro de una transacción o si se utiliza en una tabla que bloqueada por otro proceso.

Ejemplo

Este comando elimina el bloquea de la tabla PELICULAS:

```
UNLOCK TABLE PELICULAS
```

EXECUTE IMMEDIATE

EXECUTE IMMEDIATE <<*nom_sql*>> | <<$*nom_sql*>> | :*nom_sql* | :$nom_*sql*

Descripción

El comando *EXECUTE IMMEDIATE* permite ejecutar una expresión SQL dinámica. El argumento *nom_sql* representa una variable que contiene un conjunto de instrucciones SQL que se ejecutarán como un todo.

Notas:

- Este comando no puede utilizarse en una conexión SQL externa (pass-through) iniciada vía el comando 4D **USE EXTERNAL DATABASE**.
- En modo compilado, no es posible utilizar las variables 4D locales (que comienzan por el carácter $) en la cadena de búsqueda pasada al comando *EXECUTE IMMEDIATE*.

Ejemplo

Este ejemplo recupera el número de películas realizadas desde 1960:

```
C_LONGINT(NumMovies)
C_TEXT(tQueryTxt)
NumMovies:=0

tQueryTxt:="SELECT COUNT(*) FROM MOVIES WHERE Year_of_Movie >= 1960 INTO :NumMovies;"
Begin SQL
   EXECUTE IMMEDIATE :tQueryTxt;
End SQL

ALERT("La videoteca contiene "+String(NumMovies)+" películas realizadas desde 1960")
```

CREATE SCHEMA

CREATE SCHEMA *nom_sql*

Descripción

El comando CREATE SCHEMA permite crear un nuevo esquema SQL llamado *nom_sql* en la base de datos. Puede utilizar todo *nom_sql* excepto por "DEFAULT_SCHEMA" y "SYSTEM_SCHEMA".

Nota: para mayor información sobre los esquemas, consulte la sección .

Cuando se crea un nuevo esquema, por defecto los derechos de acceso asociados son los siguientes:

- Sólo lectura (Datos): <Todos>
- Lectura/Escritura (Datos):<Todos>
- Completo (datos y estructura): <Persona>
  Cada esquema puede atribuirse un tipo de derechos de acceso externos utilizando el comando *GRANT*.

Sólo el Diseñador y el Administrador de la base puede crear, modificar o eliminar esquemas.
Si el sistema de gestión de acceso de 4D no está activo (en otras palabras, si no se le ha asignado contraseña al Diseñador), todos los usuarios pueden crear y modificar esquemas sin ninguna restricción.

Cuando una base de datos se crea o se convierte con 4D v11 SQL (desde la versión 3), se crea un esquema por defecto con el fin de agrupar todas las tablas de la base. Este esquema se llama "DEFAULT_SCHEMA". No puede borrarse ni renombrarse.

Ejemplo

Creación de un esquema llamado "Derechos_Contabilidad":

```
CREATE SCHEMA Derechos_Contabilidad
```

ALTER SCHEMA

ALTER SCHEMA *nom_sql* **RENAME TO** *nom_sql*

Descripción

El comando ALTER SCHEMA permite renombrar el esquema SQL *nom_sql* (primer parámetro) por *nom_sql* (segundo parámetro).

Sólo el Diseñador y el Administrador pueden modificar esquemas.

Nota: no es posible renombrar el esquema por defecto ("DEFAULT_SCHEMA") o el esquema que contiene las tablas sistema ("SYSTEM_SCHEMA") ni utilizar estos nombres en *sql_name* (segundo parámetro).

Ejemplo

Renombrar el esquema MyFirstSchema por MyLastSchema:

```
ALTER SCHEMA MyFirstSchema RENAME TO MyLastSchema
```

DROP SCHEMA

DROP SCHEMA *nom_sql*

Descripción

El comando DROP SCHEMA puede usarse para eliminar el esquema designado por *nom_sql*.

Es posible borrar cualquier esquema, excepto el esquema por defecto (DEFAULT_SCHEMA) y el esquema que contiene las tablas sistema ("SYSTEM_SCHEMA"). Cuando se elimina un esquema, todas las tablas que se le fueron asignadas se transfieren al esquema por defecto. Las tablas transferidas heredan los derechos de acceso del esquema por defecto.

Si intenta eliminar un esquema que no existe o que ya se ha eliminado, se genera un error.

Sólo el Diseñador y el Administrador de la base pueden eliminar los esquemas.

Ejemplo

Usted quiere borrar el esquema MyFirstSchema (al cual están asignadas las tablas Table1 y Table2):

```
DROP SCHEMA MyFirstSchema
```

Después de esta operación, las dos tablas Table1 y Table2, se reasignan al esquema por defecto.

CREATE VIEW

CREATE [**OR REPLACE**] **VIEW** [*nom_schema.*]*nom_vista*[(*lista_columnas*)] **AS** *instruccion_select*[;]

Descripción

El comando **CREATE VIEW** permite crear una vista SQL llamada *nom_vista* (que es un *nom_sql* estándar) que contiene las columnas definidas en el parámetro *lista_columnas*. Es necesario especificar un nombre de columna si esta columna es una función o se deriva de una operación aritmética (escalar). También es necesario especificar un nombre de columna cuando se quiere evitar tener distintas columnas con el mismo nombre (por ejemplo, durante una operación JOIN) o cuando desea utilizar un nombre de columna diferente del que se deriva.

Si se pasa el parámetro *lista_columnas*, debe contener el mismo número de columnas como en la petición de definición *instruccion_select* de la vista. Si se omite *lista_columnas*, las columnas de la vista tendrán los
mismos nombres que los de las columnas de la *instruccion_select* de la vista.

Las vistas y las tablas deben tener nombres únicos.

Si pasa la opción **OR REPLACE**, la vista se recreará automáticamente si ya existe. Esta opción puede ser útil con el fin de cambiar la definición de una vista existente sin tener que borrar/crear/afectar los privilegios
de los objetos que ya están definidos para la vista actual.

Si no se pasa la opción **OR REPLACE**, y si la vista ya existe, se devuelve un error.

nom_schema es también un *nom_sql* estándar y puede utilizarlo para designar el nombre del esquema que contendrá la vista. Si no pasa *nom_schema* o si pasa el nombre de un esquema que no existe, la vista
se asigna automáticamente al esquema por defecto, llamado "DEFAULT_SCHEMA".

Instruccion_select designa la instrucción **SELECT** que es la consulta de definición de la vista. La *Instruccion_select* es la misma que un **SELECT** estándar de 4D, pero con las siguientes restricciones:

- No puede utilizar las cláusulas **INTO**, **LIMIT** u **OFFSET** ya que la limitación, definición o asignación de variables en 4D será realizada por **SELECT** que llama a la vista.
- No puede utilizar la cláusula **GROUP BY**.
- Las vistas son de sólo lectura y no se pueden actualizar.

Una definición de vista es "estática" y no se actualiza si las tablas fuentes se modifican o eliminan. En particular, las columnas añadidas a una tabla no aparecen en la vista basada en esta tabla. Del mismo
modo, si trata de acceder por medio de una vista a las columnas eliminadas, se produce un error.

Sin embargo, una vista que refiera a una vista fuente eliminada seguirá funcionando. De hecho, cuando se crea una vista, convierte cualquier referencia de vistas en referencias a las tablas fuente.

Las vistas tienen un alcance global. Una vez que se crea una vista con **CREATE VIEW**, es accesible para todas las partes de la aplicación (4D remoto vía SQL, bases externas creadas con el comando **CREATE DATABASE**, otras bases utilizan el comando **SQL LOGIN**, etc.) y durante la sesión, hasta que se borra utilizando el comando **DROP VIEW** o se cierre la base.

Ejemplo

Aquí presentamos algunos ejemplos de definiciones de vista basados en la tabla PEOPLE que contiene las siguientes columnas:

ID	INT64
NOMBRE	VARCHAR(30)
APELLIDO	VARCHAR(30)
DEPARTAMENTO	VARCHAR(30)
SALARIO	INT

Una vista sin restricciones:

```
CREATE VIEW FULLVIEW AS
        SELECT * FROM PERSONS;
```

Una vista sin restricciones: "horizontales". Por example, usted quiere mostrar únicamente las personas que trabajan en el departamento de Mercadeo:

```
CREATE VIEW HORIZONTALVIEW (ID, Nombre, Apellido, Salario) AS
        SELECT ID, FIRST_NAME, LAST_NAME, SALARY FROM PERSONS
        WHERE DEPARTMENT = 'Marketing';
```

Una vista agregada:

```
CREATE VIEW AGGREGATEVIEW (Nombre, Apellido AnnualSalary) AS
        SELECT Nombre, Apellido, SALARY*12 FROM PERSONS;
```

Una vista con restricciones "verticales". Por ejemplo, usted no quiere mostrar la columna SALARY:

```
CREATE VIEW VERTICALVIEW (ID, Nombre, Apellido, Departamento) AS
        SELECT ID, FIRST_NAME, LAST_NAME, DEPARTEMENT FROM PERSONS;
```

Una vez definidas las vista, puede utilizarlas como tablas estándar. Por ejemplo, si quiere obtener todas las personas cuyo salario es mayor a 5,000 Euros:

```
SELECT * FROM FULLVIEW
    WHERE SALARY < 5000
    INTO :aID, :aNombre, :aApellido, :aDepartamento, :aSalario;
```

Otro ejemplo: usted quiere obtener todas las personas del departamento de Mercadeo cuyo nombre es "Miguel":

```
SELECT ID, Apellido, Salary FROM HORIZONTALVIEW
    WHERE Nombre='Miguel'
    INTO :aID, :aApellido, :aSalary;
```

DROP VIEW

DROP VIEW [**IF EXISTS**] [*schema_name.*]*view_name*[;]

Descripción

El comando **DROP VIEW** borra de la base la vista llamada *nom_vista*.

Cuando se pasa la restricción **IF EXISTS**, el comando no hace nada y no se genera ningún error si la vista *nom_vista* no existe en la base.

nom_esquema es un *nom_sql* estándar y se puede utilizar para designar el nombre del esquema que va a contener la vista. Si no pasa *nom_esquema* o si pasa un nombre de esquema que no existe, la vista se considera automáticamente como que pertenece al esquema por defecto, llamado "DEFAULT_SCHEMA".

GRANT

GRANT[**READ** | **READ_WRITE** | **ALL**] **ON** *sql_name* **TO** *sql_name*

Descripción

El comando **GRANT** permite definir los derechos de acceso asociados al esquema *nom_sql* (primer parámetro). Estos derechos se asignarán al grupo de usuarios 4D designados por el segundo parámetro *nom_sql*.

Las palabras claves **READ**, **READ_WRITE** y **ALL** permiten definir los tipos de acceso autorizados para la tabla:

- **READ** establece el modo de acceso de sólo lectura (datos). Por defecto: <Everybody>
- **READ_WRITE** establece el modo de acceso lectura/escritura (datos).Por defecto: <Everybody>
- **ALL** establece el modo de acceso completo (datos y estructura). Por defecto: <Nobody>

Note que cada tipo de acceso está definido independientemente de los otros. En particular, si asigna sólo los derechos READ a un grupo, esto no tendrá ningún efecto ya que el grupo como también los otros continuarán beneficiándose del acceso READ_WRITE (asignado a todos los grupos por defecto). Para definir el acceso READ, debe llamar dos veces el comando **GRANT** (ver ejemplo 2).

El control de acceso sólo aplica a las conexiones externas. El código SQL ejecutado al interior de 4D vía las etiquetas **Begin SQL/End SQL** o los comandos tales como **SQL EXECUTE** tiene acceso total.

Nota de compatibilidad: durante la conversión de una base anterior a la versión 11.3 o superior, los derechos de acceso globales (tales como los definidos en la página SQL de las Preferencias de la aplicación) se transfieren al esquema por defecto.

El segundo parámetro *nom_sql* debe contener el nombre del grupo de usuarios 4D al que desea asignar derechos de acceso al esquema. Este grupo debe existir en la base 4D.

Nota: 4D permite definir los nombres de grupos incluyendo espacios y/o caracteres acentuados que no son aceptados por el estándar SQL. En este caso, debe poner el nombre entre los caracteres [y]. Por ejemplo: **GRANT READ ON [mi esquema] TO [los administradores]**

Sólo el Diseñador y el Administrador de la base pueden modificar los esquemas.

Nota sobre la integridad referencial

4D garantiza el principio de integridad referencial independientemente de los derechos de acceso. Por ejemplo, supongamos que tiene dos tablas, Tabla1 y Tabla2, conectadas por una relación de tipo Muchos a Uno (Tabla2 -> Tabla1). Tabla1 pertenece al esquema S1 y Tabla2 al esquema S2. Un usuario que tenga derechos de acceso al esquema S1 pero no al S2 puede eliminar los registros en la Tabla1. En este caso, con el fin de respetar los principios de integridad referencial, todos los registros de Tabla2 que están relacionados con los registros eliminados de la Tabla1 también se eliminarán.

Ejemplo 1

Usted quiere autorizar el acceso en lectura escritura de datos del esquema MiEsquema1 al grupo "Power_Users":

```
GRANT READ_WRITE ON MiEsquema1 TO POWER_USERS
```

Ejemplo 2

Usted quiere autorizar un acceso en lectura únicamente al grupo "Readers". Este caso requiere asignar al menos un grupo con derechos READ_WRITE (aquí "Admins") de manera que ya no se asigna a todos los grupos por defecto:

```
GRANT READ ON MySchema2 TO Readers /*Asignación del acceso en lectura únicamente */
GRANT READ_WRITE ON MySchema2 TO Admins /*Detener el acceso a todos en lectura escritura*/
```

REVOKE

REVOKE [**READ** | **READ_WRITE** | **ALL**] **ON** *nom_sql*

Descripción

El comando *REVOKE* permite borrar los derechos de acceso específicos asociados al esquema definido por el parámetro *nom_sql*.

Cuando ejecuta este comando, asigna el pseudo-grupo de usuarios <Persona> al derecho de acceso definido.

Ejemplo

Usted quiere borrar todo derecho lectura-escritura al esquema MiEsquema1:

```
REVOKE READ_WRITE ON MiEsquema1
```

REPLICATE

REPLICATE *lista_replicada*
FROM *ref_tabla*
[**WHERE** *criterio_búsqueda*]
[**LIMIT** {*número_entero* | *ref_lenguaje_4d*}]
[**OFFSET** {*número_entero* | *ref_lenguaje_4d*}]
FOR REMOTE [**STAMP**] {*número_entero* | *ref_lenguaje_4d*}
[, **LOCAL** [**STAMP**] {*número_entero* | *ref_lenguaje_4d*}]
[{**REMOTE OVER LOCAL** | **LOCAL OVER REMOTE**}]
[**LATEST REMOTE** [**STAMP**] *ref_lenguaje_4d*
[, **LATEST LOCAL** [**STAMP**] *ref_lenguaje_4d*]]
INTO {*lista_objetivo* | *ref_tabla*(*nom_sql_1*;...;*nom_sql_N*)};

Descripción

El comando **REPLICATE** permite replicar los datos de una tabla de una base A en la de una tabla de una base B. Por convención, la base donde se ejecuta el comando se llama "base local" y la base de la cual los datos se replican se llama "base remota".

Este comando sólo puede utilizarse en el marco de un sistema de replicación de base. Para que el sistema funcione, la replicación debe haber sido activada en la base local y en la base remota y cada tabla implicada deberá tener una llave primaria. Para mayor información sobre este sistema, consulte la sección **Replicación vía SQL**.

Nota: si desea implementar un sistema de sincronización completo, consulte la descripción del comando **SYNCHRONIZE**.

Pase una lista de campos (virtuales o estándar), separados por comas en *lista_replicada*. Los campos deben pertenecer a la tabla *ref_tabla* de la base remota.
La cláusula **FROM** debe ir seguida de un argumento del tipo ref_tabla que permite designar la tabla de la base remota desde la cual replicar los datos de los campos *lista_replicada*.

Nota: los campos virtuales de la tabla remota sólo se pueden almacenar en los arrays de la base local.

Lado base remota

La cláusula opcional **WHERE** permite aplicar un filtro preliminar a los registros de la tabla en la base de datos remota; sólo aquellos registros que cumplan los *criterios_de_búsqueda* serán tenidos en cuenta por el comando.

4D recupera los valores de los campos *lista_replicada* para todos los registros designados por la cláusula **FOR REMOTE STAMP**. El valor pasado en esta cláusula puede ser:

- Un valor de tipo **entero largo > 0**: en este caso, se recuperan los registros donde el valor de __ROW_STAMP es mayor o igual a este valor.
- **0**: en este caso, se recuperan todos los registros donde el valor de __ROW_STAMP es diferente de 0. Tenga en cuenta que los registros que existían antes de la activación de la replicación, no serán tenidos en cuenta (el valor de __ROW_STAMP = 0).
- **-1**: en este caso, todos los registros de la tabla remota se recuperan, es decir, todos los registros donde el valor de __ROW_STAMP >= 0. A diferencia del caso anterior, se tendrán en cuenta todos los registros de la tabla, incluidos los que existían antes de la activación de la replicación.

- **-2**: En este caso, se recuperan todos los registros borrados de la tabla remota (después de la activación de la replicación), es decir, todos los registros donde el valor de __ROW_ACTION = 2.

Por último, puede aplicar a la selección obtenida las cláusulas opcionales **OFFSET** y/o **LIMIT**:

- Cuando se pasa, la cláusula **OFFSET** permite ignorar los primeros X registros de la selección (donde X es el valor pasado a la cláusula).
- Cuando se pasa, la cláusula **LIMIT** permite restringir la selección a los Y primeros registros de la selección (donde Y es el valor pasado a la cláusula). Si la cláusula **OFFSET** también se pasa, la cláusula **LIMIT** se aplica a la selección obtenida tras la ejecución de **OFFSET**.

Una vez aplicadas ambas cláusulas, la selección resultante se envía a la base local.

Lado base local

Los valores recuperados se escriben directamente la *lista_objetivo* de la base local o en los campos estándar especificados por *nom_sql* de la tabla *ref_tabla* de la base local.
El argumento *lista_objetivo* puede contener una lista de campos estándar o una lista de arrays del mismo tipo que los campos remotos (pero no una combinación de ambos). Si el destino del comando es una lista de campos, los registros objetivos serán creados, modificados o eliminados automáticamente en función de la acción almacenada en el campo virtual __ROW_ACTION.

Los conflictos para los registros replicados que ya existen en la base objetivo (llaves primarias idénticas) se resuelven utilizando las cláusulas de prioridad (opción **REMOTE OVER LOCAL** y **LOCAL OVER REMOTE**):

- Si pasa la opción **REMOTE OVER LOCAL** u omite la cláusula de prioridad, todos los registros fuente (base remota) designados por la cláusula **FOR REMOTE STAMP** reemplazan los registros objetivo (base local) si ya existen, cambiados o no de un lado o del otro. En este caso, no tiene sentido pasar una cláusula **LOCAL STAMP** porque se ignorará.
- Si pasa la opción **LOCAL OVER REMOTE**, el comando tiene en cuenta el marcador local **LOCAL STAMP**. En este caso, los registros objetivo (base local) cuyo valor de marcador es inferior o igual al pasado en **LOCAL STAMP** no son remplazados por los registros fuente (base remota). Por ejemplo, si pasa 100 en **LOCAL STAMP**, todos los registros de la base local cuyo marcador sea menor o igual a 100 serán remplazados por los registros equivalentes de la base remota. Este principio permite preservar los datos modificados localmente y reducir la selección de los registros a replicar en la tabla local.
- Si pasa las cláusulas **LATEST REMOTE STAMP** y/o **LATEST LOCAL STAMP**, 4D devuelve en las variables *ref_lenguaje_4D* correspondientes los valores de los últimos marcadores de las tablas local y remota. Esta información puede ser útil si desea personalizar la gestión del procedimiento de sincronización. Estos valores corresponden al valor de los marcadores inmediatamente después de terminada la operación de replicación: si los utiliza en una instrucción **REPLICATE** o **SYNCHRONIZE**, no es necesario incrementarlos porque son incrementados automáticamente antes de ser devueltos por el comando **REPLICATE**.

Si la operación de replicación se lleva a cabo correctamente, la variable sistema OK toma el valor 1. Puede controlar este valor desde un método 4D.

Si se producen errores durante la operación de replicación, la operación se detiene cuando ocurra el primer error. La última variable fuente (si se ha especificado) toma el valor con el marcador del registro en el que se produjo el error. La variable sistema OK toma el valor 0. El error generado puede interceptarse por un método de gestión de errores instalado por el comando **ON ERR CALL**.

Nota: las operaciones efectuadas por el comando **REPLICATE** no tienen en cuenta las restricciones de integridad de datos. Esto significa, por ejemplo, que las reglas que rigen las llaves foráneas, unicidad, etc. no son verificadas. Si los datos recibidos pueden afectar la integridad de los datos, debe comprobar los datos después de la operación de replicación. La manera más simple es bloquear, vía el lenguaje 4D o SQL, los registros que se tienen que modificar.

SYNCHRONIZE

SYNCHRONIZE
[**LOCAL**] **TABLE** *ref_tabla* (*ref_columna_1*,...,*ref_columna_N*)
WITH
[**REMOTE**] **TABLE** *ref_tabla* (*ref_columna_1*,...,*ref_columna_N*)
FOR REMOTE [**STAMP**] {*número_entero* | *ref_lenguaje_4d*},
LOCAL [**STAMP**] {*número_entero* | *ref_lenguaje_4d*}
{**REMOTE OVER LOCAL** | **LOCAL OVER REMOTE**}
LATEST REMOTE [**STAMP**] *ref_lenguaje_4d*,
LATEST LOCAL [**STAMP**] *ref_lenguaje_4d*;

Descripción

El comando **SYNCHRONIZE** permite sincronizar dos tablas ubicadas en dos servidores 4D SQL diferentes. Todo cambio realizado en una de las tablas también se efectúa en el otro. El servidor 4D SQL que ejecuta el comando se llama servidor local y el otro servidor se llama servidor remoto.

El comando **SYNCHRONIZE** es una combinación de dos llamadas internas al comando **REPLICATE**. La primera llamada replica los datos desde el servidor remoto al servidor local y la segunda efectúa la operación inversa: replicación de los datos del servidor local al servidor remoto. Las tablas a sincronizar deben estar configuradas para la replicación

- Deben tener una llave primaria,
- La opción "Activar replicación" debe estar seleccionada en la ventana Inspector de cada tabla.

Para mayor información, consulte la descripción del comando **REPLICATE**.

El comando **SYNCHRONIZE** acepta cuatro marcadores (stamps) como "parámetros": dos marcadores en entrada y dos marcadores en salida (última modificación). Los marcadores de entrada se utilizan para indicar el momento de la última sincronización en cada servidor. Los marcadores de salida devuelven el valor de los marcadores de modificación en cada servidor justo después de la última modificación. Gracias a este principio, cuando el comando **SYNCHRONIZE** se llama regularmente, es posible usar los marcadores de salida de la última sincronización como marcadores de entrada para la siguiente.

Nota: los marcadores de entrada y de salida se expresan como valores numéricos (stamps) y no de marcadores de tiempo (timestamps). Para obtener más información sobre estos marcadores, consulte la descripción del comando **REPLICATE**.

En caso de error, el marcador de salida del servidor en cuestión contiene el marcador del registro al origen del error. Si el error se debe a una causa distinta a la sincronización (problemas de red por ejemplo), el marcador contendrá 0.
Hay dos códigos de error diferentes, uno para indicar un error de sincronización en el sitio local y otro para un error de sincronización en el sitio remoto.
Cuando ocurre un error, el estado de los datos dependerá de la de la transacción en el servidor local. En el servidor remoto, la sincronización se realiza siempre dentro de una transacción, de manera que los datos no puedan ser alterados por la operación. Sin embargo, en el servidor local, el proceso de sincronización queda bajo el control del desarrollador. Se efectúa fuera de cualquier transacción si la preferencia **Transacciones Auto-commit** no está activada, (de lo contrario, un contexto de transacción se crea automáticamente). El desarrollador puede optar por iniciar una transacción y es responsabilidad del desarrollador validar o cancelar esta transacción después de la sincronización de los datos.

Puede "forzar" la dirección de sincronización utilizando las cláusulas **REMOTE OVER LOCAL** y **LOCAL OVER REMOTE**, dependiendo de las características de su aplicación. Para mayor información sobre los mecanismos de implementación, consulte la descripción del comando

REPLICATE.

Nota: las operaciones efectuadas por el comando **SYNCHRONIZE** no tienen en cuenta restricciones de integridad de los datos. Esto significa, por ejemplo, que las reglas que rigen las llaves foráneas, unicidad, etc. no son verificadas. Si los datos recibidos pueden afectar la integridad de datos, debe verificar los datos después de la operación de sincronización. La manera más simple es bloquear, vía el lenguaje 4D o SQL, los registros que deben modificarse.
En las variables *4d_language_ref* de las cláusulas **LATEST REMOTE STAMP** y **LATEST LOCAL STAMP**, 4D devuelve los valores de los últimos marcadores de las tablas remota y local. Esta información le permite automatizar la gestión del procedimiento de sincronización. Estos valores corresponden al valor de los marcadores hasta el final de la operación de replicación: si los utiliza en una instrucción **REPLICATE** o **SYNCHRONIZE** posterior, no necesita incrementarlos; se incrementan automáticamente antes de ser devueltos por el comando **REPLICATE**.

Ejemplo

Para entender los mecanismos involucrados en la operación de sincronización, vamos a examinar las diferentes posibilidades relativas a la actualización de un registro existente en las dos bases sincronizadas.
El método de sincronización es de la siguiente forma:

```
C_LONGINT(vRemoteStamp)
C_LONGINT(vLocalStamp)
C_LONGINT(vLatestRemoteStamp)
C_LONGINT(vLatestLocalStamp)

vRemoteStamp:=X...  // ver valores en el array a continuación
vLocalStamp:=X...  // ver valores en el array a continuación
vLatestRemoteStamp:=X...  // valor devuelto en un anterior LATEST REMOTE STAMP
vLatestLocalStamp:=X...  // valor devuelto en un anterior LATEST LOCAL STAMP

Begin SQL
   SYNCHRONIZE
      LOCAL MYTABLE (MyField)
      WITH
      REMOTE MYTABLE (MyField)
      FOR REMOTE STAMP :vRemoteStamp,
      LOCAL STAMP :vLocalStamp
      LOCAL OVER REMOTE   // or REMOTE OVER LOCAL, ver en el array a continuación
      LATEST REMOTE STAMP :vLatestRemoteStamp,
      LATEST LOCAL STAMP :vLatestLocalStamp;
End SQL
```

Los datos iniciales son:

- El marcador del registro en la base LOCAL tiene un valor de 30 y el de la base REMOTA tiene un valor de 4000
- Los valores del campo MiCampo son los siguientes:

LOCAL		**REMOTE**	
Antiguo valor	Nuevo valor	Antiguo valor	Nuevo valor
AAA	BBB	AAA	CCC

- Utilizamos los valores devueltos por las cláusulas anteriores **LATEST LOCAL STAMP** y **LATEST REMOTE STAMP** para sincronizar únicamente los valores que fueron modificados desde la última sincronización.

Estas son las sincronizaciones efectuadas por el comando **SYNCHRONIZE** en función de los valores pasados en los parámetros **LOCAL STAMP** y **REMOTE STAMP** como también la opción de prioridad utilizada: ROL (para **REMOTE OVER LOCAL**) o LOR (para **LOCAL OVER REMOTE**):

LOCAL STAMP	REMOTE STAMP	Prioridad	LOCAL después de sinc	REMOTA después de sinc	Sincronización LOCAL - REMOTA
20	3000	ROL	CCC	CCC	<---->
20	3000	LOR	BBB	BBB	<---->
31	3000	ROL	CCC	CCC	<--
31	3000	LOR	CCC	CCC	<--
20	4001	ROL	BBB	BBB	-->
20	4001	LOR	BBB	BBB	-->
31	4001	ROL	BBB	CCC	Sin sincronización
31	4001	LOR	BBB	CCC	Sin sincronización
40	3000	ROL	CCC	CCC	<--
40	3000	LOR	CCC	CCC	<--
20	5000	ROL	BBB	BBB	-->
20	5000	LOR	BBB	BBB	-->
40	5000	ROL	BBB	CCC	Sin sincronización
40	5000	LOR	BBB	CCC	Sin sincronización

Reglas de sintaxis

Reglas de sintaxis

Esta sección describe el contenido y el uso de diferentes elementos de los predicados utilizados en las instrucciones SQL. Los hemos agrupados de manera que los podamos describir de manera simple y dar indicaciones generales sobre su modo de uso dentro de 4D. Las palabras claves (en negrita) siempre se pasan "tal cual" cuando se utilizan.

llamada_función_4d

{**FN** *nom_sql* ([*expresión_aritmética*, ..., *expresión_aritmética*]) **AS** *tipo_datos_sql*}

Descripción

llamada_función_4d permite ejecutar una función 4D que devuelve un valor.

El argumento *nom_sql* de la función está precedido por la palabra clave **FN** y seguido por uno o varios argumentos de tipo *expresión_aritmética*. El valor devuelto por la función será del tipo definido por *tipo_datos_sql*.

Ejemplo

Este ejemplo utiliza funciones para extraer de la tabla MOVIES el número de actores para cada película que tenga al menos 7 actores:

```
C_LONGINT($NrOfActors)
ARRAY TEXT(aMovieTitles;0)
ARRAY LONGINT(aNrActors;0)

$NrOfActors:=7
Begin SQL
    SELECT Movie_Title, {FN Find_Nr_Of_Actors(ID) AS NUMERIC}
    FROM MOVIES
    WHERE {FN Find_Nr_Of_Actors(ID) AS NUMERIC} >= :$NrOfActors
    ORDER BY 1
    INTO :aMovieTitles; :aNrActors
End SQL
```

ref_lenguaje_4d

<<nom_sql>> |*<<$nom_sql>>* | *<<[nom_sql]nom_sql>>* |
:nom_sql|*:$nom_sql*|*:nom_sql.nom_sql*

Descripción

Un argumento *ref_lenguaje_4d* especifica un nombre de variable o de campo 4D (*nom_sql*) destinado a recibir datos. Este nombre se puede pasar en una de estas formas:

<<nom_sql>>
<<$nom_sql>> (*)
<<[nom_sql]nom_sql>> (corresponde a la sintaxis 4D estándard: [NomTabla]NomCampo)
:nom_sql
:$nom_sql (*)
:nom_sql.nom_sql (corresponde a la sintaxis SQL estándar: NomTabla.NomCampo)

(*) En modo compilado, no puede utilizar referencias a variables locales (que comienzan por el símbolo $).

predicado_all_or_any

expresión_aritmética {< | <= | = | >= | > | <>} {**ANY** | **ALL** | **SOME**} (*sub_búsqueda*)

Descripción

Un *predicado_all_or_any* se utiliza para comparar una *expresión_aritmética* con una *sub_búsqueda*. Puede pasar operadores de comparación tales como <, <=, =, >=, > o <> como también las palabras claves **ANY**, **ALL** y **SOME** para efectuar la comparación con la *sub_búsqueda*.

Ejemplo

El ejemplo efectúa una subconsulta que selecciona las mejores ventas de software. La búsqueda principal selecciona los registros de las tablas VENTAS y CLIENTES donde Valor_Total es superior a los registros seleccionados por la subconsulta:

```
SELECT Valor_Total, CLIENTES.Cliente
FROM VENTAS, CLIENTES
WHERE VENTAS.Cliente_ID = CLIENTES.Cliente_ID
AND Valor_Total > ALL     (SELECT MAX (Valor_Total)
FROM VENTAS
WHERE Tipo_producto = 'Software');
```

expresión_aritmética

val_literal |

ref_columna |

llamada_función |

parámetro_comando |

expresión_condicional |

(*expresión_aritmética*) |

+ expresión_aritmética |

- expresión_aritmética |

expresión_aritmética + expresión_aritmética |

expresión_aritmética - expresión_aritmética |

*expresión_aritmética * expresión_aritmética* |

expresión_aritmética / expresión_aritmética |

Descripción

Una *expresión_aritmética* puede contener un valor literal (*val_literal*), una referencia de columna (*ref_columna*), una llamada de función (*llamada_función*), un parámetro de comando (*parámetro_comando*) o una expresión de tipo *expresión_condicional*. También puede pasar combinaciones de expresiones aritméticas utilizando los operadores +, -, * y /.

predicado_between

expresión_aritmética [**NOT**] **BETWEEN** *expresión_aritmética* **AND** *expresión_aritmética*

Descripción

El *predicado_between* permite buscar los datos cuyos valores están entre dos valores de tipo *expresión_aritmética* (pasados en orden ascendente). También puede pasar la palabra clave opcional **NOT** con el fin de excluir los valores que no están dentro de estos límites.

Ejemplo

Este ejemplo selecciona todos los clientes cuyos nombres comienzan por una letra entre la A y la E:

```
SELECT CLIENT_NOM, CLIENT_APELLIDO
FROM T_CLIENT
WHERE CLIENT_NOMBRE BETWEEN 'A' AND 'E'
```

expresión_condicional

expresión_condicional

Descripción

Una *expresión_condicional* permite aplicar una o varias condiciones al seleccionar una expresión. Puede utilizarse por ejemplo de la siguiente manera:

```
CASE
WHEN criterio_búsqueda THEN expresión_aritmética
...
WHEN criterio_búsqueda THEN expresión_aritmética
[ELSE  expresión_aritmética]
END
```

O:

```
CASE expresión_aritmética
WHEN expresión_aritmética THEN expresión_aritmética
...
WHEN expresión_aritmética THEN expresión_aritmética
[ELSE expresión_aritmética]
END
```

Ejemplo

Este ejemplo selecciona registros de la columna de números de habitación en función del valor de la columna PISO_HAB:

```
SELECT NUM_HAB
CASE PISO_HAB
WHEN 'Planta baja' THEN 0
WHEN 'Primer piso' THEN 1
WHEN 'Segundo piso' THEN 2
END AS PISOS, DORMITORIO
FROM T_HABS
ORDER BY PISOS, DORMITORIO
```

definición_columna

nom_sql tipo_datos_sql [(*número_entero*)][**NOT NULL**] [**UNIQUE**] [**AUTO_INCREMENT**]

Descripción

Una definición de columna (*definición_columna*) contiene el nombre (*nom_sql*) y el tipo de datos (*tipo_datos_sql*) de una columna. Opcionalmente, también puede pasar un *numero_entero* así como las palabras claves **NOT NULL**, **UNIQUE, AUTO_INCREMENT** y/o **AUTO_GENERATE**.

- Pasar **NOT NULL** en la *definición_columna* significa que la columna no acepta valores nulos.
- Pasar **UNIQUE** significa que el mismo valor no podrá insertarse en esta columna dos veces. Tenga en cuenta que sólo las columnas **NOT NULL** puede tener el atributo **UNIQUE**. La palabra clave **UNIQUE** deberá siempre ir precedida por **NOT NULL**.
- Pasar **AUTO_INCREMENT** significa que la columna generará un número único para cada nueva línea. Este atributo sólo se puede usar con las columnas de números.
- Pasar **AUTO_GENERATE** significa que un UUID se generará automáticamente en la columna para cada nueva línea. Este atributo sólo se puede utilizar con las columnas UUID.

Cada columna debe tener un tipo de datos. La columna debe definirse como "nula" o no "nula" y si este valor se deja en blanco, la base de datos asume "nula" como valor predeterminado. El tipo de datos para la columna no restringe los datos que se puede poner en esa columna.

Ejemplo

Este ejemplo crea una tabla con dos columnas, ID y Nombre):

```
CREATE TABLE ACTOR_FANS
(ID INT32, Nom VARCHAR NOT NULL UNIQUE);
```

ref_columna

nom_sql | *nom_sql.nom_sql* | *cadena_sql.cadena_sql*

Descripción

Una referencia de columna (*ref_columna*) consiste de un *nom_sql* o de una *cadena_sql* pasada de una de las siguientes formas:

nom_sql
nom_sql.nom_sql
cadena_sql.cadena_sql

parámetro_comando

? | <<*nom_sql*>> | <<$*nom_sql*>> | <<[*nom_sql*]*nom_sql*>> | :*nom_sql* | :$*nom_sql* | :*nom_sql*.*nom_sql*

Descripción

Un argumento *parámetro_comando* puede ser un signo de interrogación (?) o un *nom_sql* pasado en una de las siguientes formas:

?
<<nom_sql>>
<<$nom_sql>>
<<[nom_sql]nom_sql>>
:nom_sql
:$nom_sql
:nom_sql.nom_sql

predicado_comparación

expresión_aritmética {< |<= |= | >= | > | <> }*expresión_aritmética* |

expresión_aritmética {< |<= |= | >= | > | <> } (*sub_consulta*) |

(*sub_consulta*) {< |<= |= | >= | > | <> } *expresión_aritmética*

Descripción

Un argumento *predicado_comparación* utiliza los operadores <, <=, =, >=, > o <> para comparar dos argumentos de tipo *expresión_aritmética* o para comparar una *expresión_aritmética* con una *sub_consulta* como parte de un *criterio_búsqueda* aplicado a los datos.

predicado_exists

EXISTS (*sub_consulta*)

Descripción

El predicado **EXISTS** permite ejecutar una *sub_consulta* y luego verificar si devuelve un valor. Este resultado se obtiene pasando la palabra clave **EXISTS** seguida de la *sub_consulta*.

Ejemplo

Este ejemplo devuelve el total de las ventas cuando hay una tienda en la región especificada:

```
SELECT SUM (Ventas)
FROM Información_Tienda
WHERE EXISTS
(SELECT * FROM Geografía
WHERE nom_region = 'Occidente')
```

definición_llave_foránea

CONSTRAINT *nom_sql*

FOREIGN KEY (*ref_columna*, ... , *ref_columna*)

REFERENCES *nom_sql* [(*ref_columna*, ... , *ref_columna*)]

[**ON DELETE** {**RESTRICT** |**CASCADE**}]

[**ON UPDATE** {**RESTRICT** |**CASCADE**}]

Descripción

Una definición de llave foránea (*definición_llave_foránea*) se usa para hacer corresponder los campos llaves primarias (*ref_columna*) definidos en otra tabla con el fin de preservar la integridad de los datos. La restricción **FOREIGN KEY** permite pasar las referencias de una o más columnas (*ref_columna*) a definir como llaves foráneas (correspondientes a las claves primarias de otra tabla).

La cláusula **CONSTRAINT** *nom_sql* se utiliza para nombrar la restricción **FOREIGN KEY**.

La cláusula **REFERENCES** permite especificar los campos llaves primarias fuente correspondientes en la otra tabla (designada por *nom_sql*). Puede omitir la lista de campos (argumentos *ref_columna*) si la tabla (*nom_sql*) definida en la cláusula **REFERENCES** tiene una llave primaria que se va a utilizar como la clave correspondiente para la restricción llave foránea.

La cláusula opcional **ON DELETE CASCADE** especifica que cuando se elimina una línea de la tabla padre (que contiene los campos llaves primarias), también se elimina de las líneas asociadas con la línea de la tabla hijo (que contiene los campos llaves foráneas). Pasar la cláusula opcional **ON DELETE RESTRICT** evita la eliminación de datos de una tabla si otras tablas las referencian.

La cláusula opcional **ON UPDATE CASCADE** especifica que siempre que se actualiza una línea en la tabla padre (que contiene los campos de llaves primarias), también se actualiza en las líneas asociadas a esa línea de la tabla hijo (que contiene los campos llaves foráneas). Pasar la cláusula opcional **ON UPDATE RESTRICT** evita la actualización de los datos de una tabla si otras tablas las referencian.
Tenga en cuenta que si se pasan las dos cláusulas **ON DELETE** y **ON UPDATE**, ambas deben ser del mismo tipo (por ejemplo, **ON DELETE CASCADE** y **ON UPDATE CASCADE** o **ON DELETE RESTRICT** y **ON UPDATE RESTRICT**).
Si no se pasan ni la cláusula **ON DELETE** ni la **ON UPDATE**, entonces **CASCADE** se utiliza como la regla por defecto.

Ejemplo

Este ejemplo crea la tabla ORDENES luego define la columna SID_Clientes como llave foránea, asociada a la columna SID de la tabla CLIENTES:

```
CREATE TABLE ORDENES
(ID_Orden INT32,
SID_Clientes INT32,
Cantidad NUMERIC,
```

```
PRIMARY KEY (ID_Comando),
CONSTRAINT fk_1 FOREIGN KEY (SID_Clientes) REFERENCES CLIENTES(SID));
```

llamada_función

llamada_función_sql |
llamada_función_4d

Descripción

Una *llamada_función* puede ser una llamada a las **Funciones SQL** o a una función 4D (*llamada_función_4d)*. Los dos tipos de funciones pueden manipular los datos, devolver los resultados y operar en uno o más argumentos.

Ejemplo

Este ejemplo utiliza la función SQL *COUNT*:

```
C_LONGINT(vNumPersona)
Begin SQL
    SELECT COUNT (*)
    FROM VENDEDORES
    INTO :vNumPersona;
End SQL
```

predicado_in

expresión_aritmética [**NOT**] **IN** (*sub_consulta*) |
expresión_aritmética [**NOT**] **IN** (*expresión_aritmética*, ..., *expresión_aritmética*)

Descripción

El predicado *predicado_in* permite comparar sin una *expresión_aritmética* se incluye (o no se incluye si la palabra clave **NO** también se pasa) en una lista de valores. El conjunto de valores utilizados por la comparación puede ser una secuencia de expresiones aritméticas, que se pasan o el resultado de una *sub_consulta*.

Ejemplo

Este ejemplo selecciona los registos de la tabla ORDENES cuyo valor de la columna id_orden es igual a 10000, 10001, 10003 o 10005:

```
SELECT *
FROM ORDERS
WHERE id_order IN (10000, 10001, 10003, 10005);
```

predicado_is_null

expresión_aritmética **IS** [**NOT**] **NUL**

Descripción

El predicado **IS NULL** permite encontrar una *expresión_aritmética* con un valor **NULL**. También puede pasar la palabra clave **NOT** para buscar las expresiones sin valores **NULL**.

Ejemplo

Este ejemplo selecciona los productos cuyo peso es menor a 15 o cuya columna Color contenga el valor NULL:

```
SELECT Nom, Peso, Color
FROM PRODUCTS
WHERE Peso < 15.00 OR Color IS NULL
```

predicado_like

expresión_aritmética [**NOT**] **LIKE** *expresión_aritmética* [**ESCAPE** *cadena_sql*]

Description

El predicado **LIKE** permite seleccionar los datos correspondientes a la *expresión_aritmética* pasada después de la palabra clave **LIKE**. También puede pasar la palabra clave **NOT** para seleccionar los datos que no correspondan a esta expresión. Puede utilizar la palabra clave **ESCAPE** para evitar que el carácter pasado en *cadena_sql* se interprete como un comodín. Generalmente se utiliza cuando quiere buscar los caracteres '%' o '_'.

Ejemplo 1

Este ejemplo selecciona los proveedores cuyo nombre contenga "bob":

```
SELECT * FROM proveedores
WHERE nom LIKE '%bob%';
```

Ejemplo 2

Este ejemplo selecciona los proveedores cuyo nombre no comienza por la letra T:

```
SELECT * FROM proveedores
WHERE nom NOT LIKE 'T%';
```

Ejemplo 3

Este ejemplo selecciona los proveedores cuyo nombre comience por "Sm" y termine en "th":

```
SELECT * FROM proveedores
WHERE nom LIKE 'Sm_th'
```

literal

número_entero | *número_fraccionario* | *cadena_sql* |número_hexadecimal

Descripción

Un valor *literal* es un dato de tipo *número_entero* (entero), un *número_fraccionario* (fracción), una *cadena_sql* o un *número_hexadecimal*.
La notación hexadecimal (introducida en 4D 12.1) permite expresar todo tipo de datos representados como bytes. Un byte siempre está definido por dos valores hexadecimales. Para indicar el uso de esta notación en un comando SQL, simplemente debe utilizar la sintaxis hexadecimal SQL estándar:

X'<valor hexadecimal>'

Por ejemplo, pra el valor decimal *1*5, puede escribir **X'0f'**. Puede definir un valor vacío (cero byte) escribiendo **X''**.

Nota: los comandos **SQL EXPORT DATABASE** y **SQL EXPORT SELECTION** exportan datos binarios al formato hexadecimal cuando estos datos están incluidos en el archivo principal.

predicado

predicado

Descripción

Un *predicado* sigue la cláusula **WHERE** y permite añadir las condiciones de búsqueda para la selección de los datos. Puede ser de uno de los siguientes tipos:
predicado_comparaison
predicado_between
predicado_like
predicado_is_null
predicado_in
predicado_all_or_any
predicado_exists

definición_llave_primaria

[**CONSTRAINT** *nom_sql*] **PRIMARY KEY** (*nom_sql*, ... , *nom_sql*)

Descripción

n argumento de tipo *definición_llave_primaria* permite pasar la columna o la combinación de columnas (*nom_sql*) que se utilizará como llave primaria (**PRIMARY KEY**, número único) de la tabla. La o las columna(s) pasadas deben no contener duplicados ni valores **NULL**.

Una restricción opcional **CONSTRAINT** también puede preceder la llave **PRIMARY KEY** pasada con el fin de limitar los valores que pueden insertarse en la columna.

Ejemplo

Este ejemplo crea una tabla y define la columna SID como llave primaria:

```
CREATE TABLE Clientes
(SID int32,
Nombre varchar(30),
Apellido varchar(30),
PRIMARY KEY (SID));
```

criterio_búsqueda

predicado |
NOT *criterio_búsqueda* |
(*criterio_búsqueda*) |
criterio_búsqueda **OR** *criterio_búsqueda* |
criterio_búsqueda **AND** *criterio_búsqueda*

Descripción

Un *criterio_búsqueda* especifica una condición a aplicar a los datos seleccionados. También es posible pasar una combinación de condiciones de búsqueda utilizando las palabras claves **AND** y **OR**. También puede preceder un *criterio_búsqueda* con la palabra clave **NOT** para recuperar los datos que no cumplen con la condición definida.

También es posible pasar un *predicado* como *criterio_búsqueda*.

Ejemplo

Este ejemplo utiliza una combinación de criterios de búsqueda en la cláusula **WHERE**:

```
SELECT proveedor_id
FROM proveedor
WHERE (nom = 'CANON')
OR (nom = 'Hewlett Packard' AND city = 'New York')
OR (nom = 'Firewall' AND status = 'Cerrado' AND city = 'Chicago');
```

select_elemento

expresión_aritmética [[**AS**] {*cadena_sql* |*nom_sql*}]

Descripción

Un *select_elemento* especifica uno o varios elementos a incluir en los resultados. Una columna se genera para cada *select_elemento* pasado. Cada *select_elemento* consiste de una *expresión_aritmética*. También puede pasar la palabra clave opcional **AS** con el fin de especificar una *cadena_sql* o un *nom_sql* a pasar a la columna. (Pasar una *cadena_sql* o *nom_sql* sin la palabra clave **AS** tiene el mismo efecto).

Ejemplo

Este ejemplo crea una columna llamada Año_Peli que contiene las películas realizadas a partir del año 2000:

```
ARRAY INTEGER(AñoPeli;0)
Begin SQL
    SELECT Año_de_estreno AS Año_Peli
    FROM MOVIES
    WHERE Año_Peli >= 2000
    ORDER BY 1
    INTO :AñoPeli;
End SQL
```

lista_orden

{*ref_columna* | *número_entero*} [**ASC** | **DESC**], ... , {*ref_columna* | *número_entero*} [**ASC** |**DESC**]

Descripción

Un argumento de tipo *lista_orden* contiene una referencia de columna *ref_columna* o un número *número_entero* que define la columna cuyos valores deben ordenarse. También puede pasar las palabras claves **ASC** o **DESC** con el fin de especificar si la ordenación debe ser ascendente o descendente. Por defecto, la ordenación es ascendente.

tipo_datos_sql

ALPHA_NUMERIC |**VARCHAR** | **TEXT** | **TIMESTAMP** |**INTERVAL** |**DURATION** |**BOOLEAN** |**BIT** | **BYTE** |**INT16** |**SMALLINT** | **INT32** | **INT** |**INT64** |**NUMERIC** | **REAL** | **FLOAT** | **DOUBLE PRECISION** | **BLOB** | **BIT VARYING** | **CLOB** | **PICTURE**

Descripción

Un argumento *tipo_datos_sql* sigue la palabra clave **AS** en una *llamada_función_4d* y puede tener uno de los siguientes valores:

ALPHA_NUMERIC
VARCHAR
TEXT
TIMESTAMP
INTERVAL
DURATION
BOOLEAN
BIT
BYTE
INT16
SMALLINT
INT32
INT
INT64
NUMERIC
REAL
FLOAT
DOUBLE PRECISION
BLOB
BIT VARYING
CLOB
PICTURE

nom_sql

nom_sql

Description

Un *nom_sql* es un nombre SQL estándar que comienza por un carácter del alfabeto Latino y contiene únicamente los caracteres latinos, los números y/o guiones bajos, o una cadena entre corchetes. Pase dos corchetes derechos como caracteres de escape para el corchete derecho.

Ejemplos:

Cadena a pasar	**nom_sql**
MiNomSQL_2	MiNomSQL_2
Mi nom !&^#%!&#% no estándar	[Mi nom !&^#%!&#% no estándar]
[nombre entre corchetes]	[[nombre entre corchetes]]]
nombre con [] corchetes	[nombre con []] corchetes]

cadena_sql

cadena_sql

Descripción

Una *cadena_sql* contiene una cadena de caracteres entre apóstrofes. Los apóstrofes en la cadena deben ser dobles y las cadenas que ya están entre apóstrofes deben ser triplicados.

Ejemplos:

Cadena a pasar	**cadena_sql**
mi cadena	'mi cadena'
cadena con ' dentro	'cadena con ' ' dentro'
'cadena entre apóstrofes'	' ' 'cadena entre apóstrofes' ' '

sub_consulta

SELECT [**ALL | DISTINCT**]
{* | *select_elemento*, ..., *select_elemento*}
FROM *ref_tabla*, ..., *ref_tabla*
[**WHERE** *criterio_búsqueda*]
[**GROUP BY** *lista_orden*]
[**HAVING** *criterio_búsqueda*]
[**LIMIT** {*número_entero* | **ALL**}]
[**OFFSET** *número_entero*]

Descripción

Una *sub_consulta* es como una instrucción *SELECT* independiente entre paréntesis y pasada en el predicado de otra instrucción SQL (*SELECT*, **INSERT**, *UPDATE* o *DELETE*). Actúa como una petición en una búsqueda y con frecuencia se pasa como parte de una cláusula **WHERE** o **HAVING**.

restricción_tabla

{*definición_llave_primaria* | *definición_llave_foránea*}

Descripción

Una *restricción_tabla* restringe los valores que una tabla puede almacenar. Puede pasar una *definición_llave_primaria* o un *definición_llave_foránea*. La *definición_llave_primaria* define la llave primaria para la tabla y la *definición_llave_foránea* se utiliza para definir la llave foránea (que corresponde a la llave primaria de otra tabla).

ref_tabla

{*nom_sql* | *cadena_sql*} [[**AS**] {*nom_sql*|*cadena_sql*}]

Descripción

Una referencia de tabla puede ser un nombre SQL estándar o una cadena. También puede pasar la palabra clave opcional **AS** para asignar un alias (en la forma de un *nom_sql* o *cadena_sql*) a la columna. (Pasar una *cadena_sql* o un *nom_sql* sin la palabra clave **AS** tiene el mismo efecto).

Transacciones

Transacciones
START
COMMIT
ROLLBACK

Transacciones

Descripción

Las transacciones son un conjunto de instrucciones SQL ejecutadas en bloque. O todas las instrucciones tienen éxito o no ninguna de ellas. Las transacciones utilizan bloqueos para preservar la integridad de los datos durante su ejecución. Si la transacción termina correctamente, puede utilizar el comando **COMMIT** para almacenar permanentemente sus modificaciones. De lo contrario, llame el comando **ROLLBACK** para cancelar todas las modificaciones y restaurar la base de datos a su estado anterior.

No hay diferencia entre una transacción 4D y una transacción SQL. Los dos tipos comparten los mismos datos y procesos. Las instrucciones SQL incluidas en una estructura **Begin SQL/End SQL**, el comando **QUERY BY SQL** y los comandos SQL genéricos integrados aplicados a la base local siempre se ejecutan en el mismo contexto que los comandos 4D estándar.

Nota: 4D ofrece la opción "Auto-commit" que se puede utilizar para iniciar y validar las transacciones de forma automática al usar los comandos SIUD (*SELECT*, **INSERT**, **UPDATE** y *DELETE*) con el fin de preservar la integridad de los datos. Para mayor información, consulte la sección .
Los siguientes ejemplos ilustran las diferentes combinaciones de transacciones.

Ni "John", ni "Smith" se añadirán a la tabla emp:

```
SQL LOGIN(SQL_INTERNAL;"";"")  `Inicializar el motor SQL de 4D
START TRANSACTION  `Iniciar una transacción en el proceso actual
Begin SQL
   INSERT INTO emp
   (NAME)
   VALUES ('John');
End SQL SQL EXECUTE("START")  `Otra transacción en el proceso actual
SQL CANCEL LOAD SQL EXECUTE("INSERT INTO emp (NAME) VALUES ('Smith')")  `Esta instrucción se ejecuta en el mismo proceso
SQL CANCEL LOAD SQL EXECUTE("ROLLBACK")  `Cancelar la transacción interna del proceso
CANCEL TRANSACTION  `Cancelar la transacción externa del proceso
SQL LOGOUT
```

Sólo “John” se añadirá a la tabla emp:

```
SQL LOGIN(SQL_INTERNAL;"";"")
START TRANSACTION
Begin SQL INSERT INTO emp(NAME)
   VALUES ('John');
End SQL SQL EXECUTE("START")
SQL CANCEL LOAD
SQL EXECUTE("INSERT INTO emp (NAME) VALUES ('Smith')")
SQL CANCEL LOAD SQL EXECUTE("ROLLBACK")  `Cancelar la transacción interna del proceso
VALIDATE TRANSACTION  `Validar la transacción externa del proceso
SQL LOGOUT
```

Ni “John” ni “Smith” se añadirán a la tabla emp. La transacción externa anula la transacción interna:

```
SQL LOGIN(SQL_INTERNAL;"";"")
START TRANSACTION
Begin SQL
    INSERT INTO emp
    (NAME)
    VALUES ('John');
End SQL
SQL EXECUTE("START")
SQL CANCEL LOAD
SQL EXECUTE("INSERT INTO emp (NAME) VALUES ('Smith')")
SQL CANCEL LOAD
SQL EXECUTE("COMMIT") `Validar la transacción interna del proceso
CANCEL TRANSACTION `Cancelar la transacción externa del proceso
SQL LOGOUT
```

"John" y "Smith" se añadirá a la tabla emp:

```
SQL LOGIN(SQL_INTERNAL;"";"")
START TRANSACTION
Begin SQL
    INSERT INTO emp
    (NAME)
    VALUES ('John');
End SQL
SQL EXECUTE("START")
SQL CANCEL LOAD
SQL EXECUTE("INSERT INTO emp (NAME) VALUES ('Smith')")
SQL CANCEL LOAD
SQL EXECUTE("COMMIT") `Validar la transacción interna del proceso
VALIDATE TRANSACTION `Validar la transacción externa del proceso
SQL LOGOUT
```

START

START [**TRANSACTION**]

Descripción

El comando *START* se utiliza para definir el inicio de una transacción. Si este comando se ejecuta cuando una transacción está en curso, se inicia una sub transacción. La palabra clave **TRANSACTION** es opcional.

Ejemplo

Este ejemplo efectúa una selección al interior de una transacción:

```
START TRANSACTION;
SELECT * FROM suppliers
WHERE supplier_name LIKE '%bob%';
COMMIT TRANSACTION;
```

COMMIT

COMMIT [**TRANSACTION**]

Descripción

El comando *COMMIT* define el final de una transacción ejecutada con éxito. Este comando asegura que todas las modificaciones efectuadas durante la transacción se vuelvan parte permanente de la base. También libera los recursos utilizados por la transacción. Recuerde que no puede utilizar el comando *ROLLBACK* después de un *COMMIT* ya que los cambios son permanentes. La palabra clave **TRANSACTION** es opcional.

Ejemplo

Ver el ejemplo del comando *START*.

ROLLBACK

ROLLBACK [**TRANSACTION**]

Descripción

El comando *ROLLBACK* cancela la transacción en curso y restaura los datos a su estado anterior al comienzo de la transacción. También libera todos los recursos utilizados por la transacción. La palabra clave **TRANSACTION** es opcional.

Ejemplo

Este ejemplo ilustra el empleo del comando *ROLLBACK*:

```
START TRANSACTION
SELECT * FROM suppliers
WHERE supplier_name like '%bob%';
ROLLBACK TRANSACTION;
```

Funciones

Funciones SQL
ABS
ACOS
ASCII
ASIN
ATAN
ATAN2
AVG
BIT_LENGTH
CAST
CEILING
CHAR
CHAR_LENGTH
COALESCE
CONCAT
CONCATENATE
COS
COT
COUNT
CURDATE
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURTIME
DATABASE_PATH
DATE_TO_CHAR
DAY
DAYNAME
DAYOFMONTH
DAYOFWEEK
DAYOFYEAR
DEGREES
EXP
EXTRACT
FLOOR
HOUR
INSERT
LEFT
LENGTH
LOCATE
LOG
LOG10
LOWER
LTRIM
MAX
MILLISECOND
MIN
MINUTE
MOD
MONTH
MONTHNAME
NULLIF

- OCTET_LENGTH
- PI
- POSITION
- POWER
- QUARTER
- RADIANS
- RAND
- REPEAT
- REPLACE
- RIGHT
- ROUND
- RTRIM
- SECOND
- SIGN
- SIN
- SPACE
- SQRT
- SUBSTRING
- SUM
- TAN
- TRANSLATE
- TRIM
- TRUNC
- TRUNCATE
- UPPER
- WEEK
- YEAR

Funciones SQL

Las funciones SQL se aplican a las columnas de datos con el fin de devolver un resultado específico en 4D. Los nombres de las funciones aparecen en negrita y se pasan como están, por lo general seguidos de uno o varios argumentos de tipo *expresión_aritmética*.

ABS

ABS (*expresión_aritmética*)

Descripción

La función **ABS** devuelve el valor absoluto de la *expresión_arirmética*.

Ejemplo

Este ejemplo devuelve el valor absoluto de los precios y los multiplica por una cierta cantidad:

```
ABS(Price) * quantity
```

ACOS

ACOS (expresión_*aritmética*)

Descripción

La función **ACOS** devuelve el arco coseno de la *expresión_aritmética*. Es el inverso de la función **COS**. La *expresión_aritmética* representa el ángulo expresado en radianes.

Ejemplo

Este ejemplo devuelve el arcocoseno del ángulo de -0.73 radianes:

```
SELECT ACOS(-0.73)
FROM TABLES_OF_ANGLES;
```

ASCII

ASCII (*expresión_aritmética*)

Descripción

La función *ASCII* devuelve el caracter más a la izquierda de la *expresión_aritmética* como un entero. Si la *expresión_aritmética* está vacía, la función devuelve el valor **NULL**.

Ejemplo

Este ejemplo devuelve la primera letra de cada apellido como un entero:

```
SELECT ASCII(SUBSTRING(Apellido,1,1))
FROM PEOPLE;
```

ASIN

ASIN (*expresión_aritmética*)

Descripción

La función **ASIN** devuelve el arcoseno de la *expresión_aritmética*. Es el inverso de la función (**SIN**). La *expresión_aritmética* representa el ángulo expresado en radianes.

Ejemplo

Este ejemplo devuelve el arcoseno del ángulo de -0.73 radianes:

```
SELECT ASIN(-0.73)
FROM TABLES_OF_ANGLES;
```

ATAN

ATAN (*expresión_aritmética*)

Descripción

La función **ATAN** devuelve el arcotangente de la *expresión_aritmética*. Es el inverso de la función tangente (**TAN**). La *expresión_aritmética* representa el ángulo expresado en radianes.

Ejemplo

Este ejemplo devuelve el arcotangente del ángulo de -0.73 en radianes:

```
SELECT ATAN(-0.73)
FROM TABLES_OF_ANGLES;
```

ATAN2

ATAN2 (*expresión_aritmética*, *expresión_aritmética*)

Descripción

La función *ATAN2* devuelve el arcotangente de las coordenadas "x" y "y", donde "x" es la primera *expresión_aritmética* pasada y "y" la segunda.

Ejemplo

Este ejemplo devuelve el arcotangente de las coordenadas 0.52 y 0.60:

```
SELECT ATAN2( 0.52, 0.60 );
```

AVG

AVG ([**ALL** | **DISTINCT**] *expresión_aritmética*)

Descripción

La función *AVG* devuelve la media de la *expresión_aritmética*. Las palabras claves opcionales **ALL** y **DISTINCT** permiten respectivamente incluir o eliminar los valores duplicados.

Ejemplo

Este ejemplo devuelve el valor mínimo, máximo, promedio y total de los boletos vendidos para la tabla MOVIES:

```
SELECT MIN(Tickets_Vendidos),
MAX(Tickets_Vendidos),
AVG(Tickets_Vendidos),
SUM(Tickets_Vendidos)
FROM MOVIES
```

BIT_LENGTH

BIT_LENGTH (*expresión_aritmética*)

Descripción

La función *BIT_LENGTH* devuelve la longitud en bits de la *expresión_aritmética*.

Ejemplo

Este ejemplo devuelve 8:

```
SELECT BIT_LENGTH( '01101011' );
```

CAST

CAST (*expresión_aritmética* **AS** *tipo_datos_sql*)

Descripción

La función **CAST** convierte la *expresión_aritmética* en el *tipo_datos_sql* pasado después de la palabra clave **AS**.
Nota: la función **CAST** no es compatible con campos de tipo "Entero 64 bits" en modo compilado.

Ejemplo

Este ejemplo convierte el año de la película en Entero:

```
SELECT Año_Realizacion, Titulo, Director, Media, Tickets_vendidos
FROM MOVIES
WHERE Año_Realizacion >= CAST('1960' AS INT)
```

CEILING

CEILING (*expresión_aritmética*)

Descripción

La función *CEILING* devuelve el entero más pequeño que sea mayor o igual a la *expresión_aritmética*.

Ejemplo

Este ejemplo devuelve el entero más pequeño que sea mayor o igual a -20.9:

```
CEILING (-20.9)
`devuelve -20
```

CHAR

CHAR (*expresión_aritmética*)

Descripción

La función **CHAR** devuelve una cadena de caracteres de longitud fija basada en el tipo de la *expresión_aritmética* pasada.

Ejemplo

Este ejemplo devuelve una cadena de caracteres basada en el código de la primera letra de cada apellido:

```
SELECT CHAR(ASCII(SUBSTRING(Apellido,1,1)))
FROM PEOPLE;
```

CHAR_LENGTH

CHAR_LENGTH (*expresión_aritmética*)

Descripción

La función *CHAR_LENGTH* devuelve el número de caracteres presentes en la *expresión_aritmética*.

Ejemplo

Este ejemplo devuelve el número de caracteres en el nombre de los productos con peso menor a 15 lbs.

```
SELECT CHAR_LENGTH (Nombre)
FROM PRODUCTS
WHERE Weight < 15.00
```

COALESCE

COALESCE (*expresión_aritmética, ..., expresión_aritmética*)

Descripción

La función *COALESCE* devuelve el valor de la primera expresión no null de la lista de argumentos de tipo *expresión_aritmética* pasado. La función devuelve **NULL** si todas las expresiones pasadas son NULL.

Ejemplo

Este ejemplo devuelve todos los números de facturas desde el año 2007 donde el IVA es superior a 0:

```
SELECT NO_FACTURAS
FROM FACTURAS
WHERE EXTRACT(AÑO(FECHA_FACTURA)) = 2007
HAVING (COALESCE(IVA_FACTURA;0) > 0)
```

CONCAT

CONCAT (*expresión_aritmética*, *expresión_aritmética*)

Descripción

La función *CONCAT* devuelve las dos expresiones pasadas como parámetros en forma de una sola cadena concatenada.

Ejemplo

Este ejemplo devuelve el nombre y apellido como una cadena concatenada:

```
SELECT CONCAT(CONCAT(PERSONAS.Nombre,' '), PERSONAS.Apellido) FROM PERSONAS;
```

CONCATENATE

CONCATENATE (*expresión_aritmética*, *expresión_aritmética*)

Descripción

La función *CONCATENATE* devuelve las dos expresiones pasadas en parámteros en forma de una sola cadena concatenada.

Ejemplo

Ver el ejemplo de la función *CONCAT*.

COS

COS (expresión_*aritmética*)

Descripción

La función **COS** devuelve el coseno de la *expresión_aritmética*. La *expresión_aritmética* representa el ángulo expresado en radianes.

Ejemplo

Este ejemplo devuelve el coseno del ángulo expresado en radianes (grados * 180/3,1416):

```
SELECT COS(degrees * 180 / 3,1416)
FROM TABLES_OF_ANGLES;
```

COT

COT (*expresión_aritmética*)

Descripción

La función *COT* devuelve la cotangente de la *expresión_aritmética*. La *expresión_aritmética* define un ángulo expresado en radianes.

Ejemplo

Este ejemplo devuelve la cotangente del ángulo expresado en radianes (3,1416):

```
SELECT COT(3,1416)
FROM TABLES_OF_ANGLES;
```

COUNT

COUNT ({ [[**ALL** |**DISTINCT**] *expresión_aritmética*] |* })

Descripción

La función *COUNT* devuelve el número de valores no null presentes en la *expresión_aritmética*. Las palabras claves opcionales **ALL** e **DISTINCT** permiten respectivamente incluir o excluir los valores duplicados.

Si pasa el carácter *, la función devuelve el número total de registros en la *expresión_aritmética*, incluyendo los valores NULL.

Ejemplo

Este ejemplo devuelve el número total de películas en la tabla PELICULAS:

```
SELECT COUNT(*)
FROM PELICULAS
```

CURDATE

CURDATE ( )

Descripción

La función *CURDATE* devuelve la fecha actual.

Ejemplo

Este ejemplo crea una tabla de facturas e inserta la fecha actual en la columna FECHA_FACT:

```
ARRAY TEXT(30;aDate;0)
Begin SQL
    CREATE TABLE INVOICES
    (DATE_FACT VARCHAR(40));

    INSERT INTO INVOICES
    (DATE_FACT)
    VALUES (CURDATE());

    SELECT *
    FROM FACTURAS
    INTO :aDate;
End SQL
```

`el array aDate devuelve la fecha y la hora de ejecución del comando INSERT.

CURRENT_DATE

CURRENT_DATE ( )

Descripción

La función *CURRENT_DATE* devuelve la fecha local actual.

Ejemplo

Este ejemplo crea una tabla de facturas e inserta la fecha actual en la columna FECHA_FACT:

```
ARRAY TEXT(30;aDate;0)
Begin SQL
   CREATE TABLE INVOICES
   (INV_DATE VARCHAR(40));

   INSERT INTO INVOICES
   (INV_DATE)
   VALUES (CURRENT_DATE());

   SELECT *
   FROM INVOICES
   INTO :aDate;
End SQL
  `el array aDate devuelve la fecha y la hora de ejecuciòn del comando INSERT.
```

CURRENT_TIME

CURRENT_TIME ( )

Descripción

La función *CURRENT_TIME* devuelve la hora local actual.

Ejemplo

Este ejemplo crea una tabla de facturas e inserta la hora actual en la columna HORA_FACT:

```
ARRAY TEXT(30;aDate;0)
Begin SQL
    CREATE TABLE INVOICES
    (INV_DATE VARCHAR(40));

    INSERT INTO INVOICES
    (INV_DATE)
    VALUES (CURRENT_TIME());

    SELECT *
    FROM INVOICES
    INTO :aDate;
End SQL
  `el array aDate devuelve la hora de ejecución del comando INSERT.
```

CURRENT_TIMESTAMP

CURRENT_TIMESTAMP ( )

Descripción

La función *CURRENT_TIMESTAMP* devuelve la fecha y la hora local actual.

Ejemplo

Este ejemplo crea una tabla de facturas e inserta la fecha y la hora actuales en la columna FECHAHORA_FACT:

```
ARRAY TEXT(30;aDate;0)
Begin SQL
    CREATE TABLE INVOICES
    (FECHAHORA_FACT VARCHAR(40));

    INSERT INTO INVOICES
    (FECHAHORA_FACT)
    VALUES (CURRENT_TIMESTAMP());

    SELECT *
    FROM INVOICES
    INTO :aDate;
End SQL
```

`El array aDate devuelve la fecha y la hora de ejecución del comando INSERT.

CURTIME

CURTIME ( )

Descripción

La función *CURTIME* devuelve la hora actual con una precisión de un segundo.

Example

Este ejemplo crea una tabla de facturas e inserta la hora actual en la columna HORA_FACT:

```
ARRAY TEXT(aDate;0)
Begin SQL
    CREATE TABLE INVOICES
    (HORA_FACT VARCHAR(40));

    INSERT INTO INVOICES
    (HORA_FACT)
    VALUES (CURTIME());

    SELECT *
    FROM FACTURAS
    INTO :aDate;
End SQL
  `El array aDate devuelve la hora de ejecución del comando INSERT.
```

DATABASE_PATH

DATABASE_PATH()

Descripción

La función **DATABASE_PATH** devuelve la ruta de acceso completa de la base actual. La base actual puede modificarse utilizando el comando **USE DATABASE**. Por defecto, la base actual es la base 4D principal.

La ruta de acceso devuelta está en formato POSIX.

Ejemplo

Supongamos que la base externa actual se llama TestBase.4DB y está en la carpeta "C:MyDatabases". Después de la ejecución del siguiente código:

```
C_TEXT($vCrtDatabasePath)
Begin SQL
    SELECT DATABASE_PATH()
    FROM _USER_SCHEMAS
    LIMIT 1
    INTO :$vCrtDatabasePath;
End SQL
```

... la variable *$vCrtDatabasePath* contendrá "C:/MyDatabases/TestBase.4DB”.

DATE_TO_CHAR

DATE_TO_CHAR (*expresión_aritmética*, *expresión_aritmética*)

Descripción

La función *DATE_TO_CHAR* devuelve una representación en forma de texto de la fecha pasada en la primera *expresión_aritmética* en función del formato especificado en la segunda *expresión_aritmética*. La primera *expresión_aritmética* debe ser de tipo Timestamp o Duration y la segunda debe ser de tipo de texto.

Los códigos de formato que se pueden utilizar se presentan a continuación. En general, si un código de formato comienza por un carácter en mayúsculas y produce un cero, entonces el número se iniciará con uno o más ceros cuando sea el caso; de lo contrario, no habrá ceros a la izquierda. Por ejemplo, si dd devuelve 7, luego Dd devolverá 07.

El uso de caracteres en mayúsculas y minúsculas en los códigos de formato los días y meses se reproducirá en los resultados devueltos. Por ejemplo, pasar "day" devolverá "lunes", pasar "Day" devolverá "Lunes" y pasar "DAY" devolverá "LUNES".

am - am o pm en función del valor de la hora
pm - am o pm en función del valor de la hora
am - am o pm en función del valor de la hora
pm - am o pm en función del valor de la hora

d - número del día de la semana (1-7)
dd - número del día del mes (1-31)
ddd - número del día del año
day - número del día de la semana
dy - nombre abreviado del día de la semana en 3 letras

hh - hora en números, basada en 12 horas (0-11)
HH12 - hora en números, basada en 12 horas (0-11)
hh24 - hora en números, basada en 24 horas (0-23)

J - día Juliano

mi - minutos (0-59)
mm - mes en números (0-12)

q - trimestre

ss - segundos (0-59)
sss - milisegundos (0-999)

w - semana del mes (1-5)
ww - semana del año (1-53)

yy - año
yyyy - año

[Cualquier texto] - el texto entre corchetes ([ ]) no se interpreta y se inserta como está 'espacio de carácter -.,:;'espacio' 'tab' - se deja tal cual, sin cambios.

Ejemplo

Este ejemplo devuelve la fecha de nacimiento como un número de día de la semana (1-7):

```
SELECT DATE_TO_CHAR (Fecha_Nacimiento,'d')
FROM EMPLEADOS;
```

DAY

DAY (*expresión_aritmética*)

Descripción

La función *DAY* devuelve el día del mes de la fecha pasada en la *expresión_aritmética*.

Ejemplo

Este ejemplo devuelve el día del mes para la fecha "05-07-2007":

```
SELECT DAY('05-07-2007');
`devuelve 7
```

DAYNAME

DAYNAME (*expresión_aritmética*)

Descripción

La función *DAYNAME* devuelve el nombre del día de la semana para la fecha pasada en *expresión_aritmética*.

Ejemplo

Este ejemplo devuelve el nombre del día de la semana para cada fecha de nacimiento pasada:

```
SELECT DAYNAME(Fecha_nacimiento);
```

DAYOFMONTH

DAYOFMONTH (*expresión_aritmética*)

Descripción

La función *DAYOFMONTH* devuelve el número del día del mes (de 1 a 31) de la fecha pasada en la *expresión_aritmética*.

Ejemplo

Supongamos que tenemos la tabla PERSONAS con un campo Fecha_Nacimiento. El siguiente código permite obtener el día de nacimiento de cada persona en la tabla PERSONAS:

```
SELECT DAYOFMONTH(Fecha_Nacimiento)
FROM PERSONAS;
```

DAYOFWEEK

DAYOFWEEK (*expresión_aritmética*)

Descripción

La función *DAYOFWEEK* devuelve un número que representa el día de la semana (de 1 a 7, donde 1 es domingo y 7 es sábado) de la fecha pasada en la *expresión_aritmética*.

Ejemplo

Supongamos que tenemos la tabla PERSONAS con un campo Fecha_Nacimiento. El siguiente código permite obtener el día de la semana de la fecha de nacimiento de cada persona almacenada en la tabla PERSONAS:

```
SELECT DAYOFWEEK(Fecha_Nacimiento)
FROM PERSONAS;
```

DAYOFYEAR

DAYOFYEAR (*expresión_aritmética*)

Descripción

La función *DAYOFYEAR* devuelve el número del día del año (de 1 a 366, donde 1 es primero de enero) de la fecha pasada en la *expresión_aritmética*.

Ejemplo

Supongamos que tenemos la tabla PERSONAS con un campo Fecha_Nacimiento. El siguiente código permite obtener el día del año de la fecha de nacimiento de cada persona en la tabla PERSONAS:

```
SELECT DAYOFYEAR(Fecha_Nacimiento)
FROM PERSONAS;
```

DEGREES

DEGREES (*expresión_aritmética*)

Descripción

La función *DEGREES* devuelve el número de grados de la *expresión_aritmética*. La *expresión_aritmética* representa el ángulo expresado en radianes.

Ejemplo

Este ejemplo crea una tabla e inserta valores basados en los números de grados del valor Pi:

```
CREATE TABLE Degrees_table (PI_value float);
INSERT INTO Degrees_table VALUES
(DEGREES(PI()));
SELECT * FROM Degrees_table
```

EXP

EXP (*expresión_aritmética*)

Descripción

La función **EXP** devuelve el valor exponencial de la *expresión_aritmética*, es decir e elevado a la potencia x, "x" es el valor pasado en la *expresión_aritmética*.

Ejemplo

Este ejemplo devuelve e elevado a la 15:

```
SELECT EXP( 15 ); `devuelve 3269017,3724721107
```

EXTRACT

EXTRACT ({**YEAR** | **MONTH** | **DAY** | **HOUR** | **MINUTE** | **SECOND** | **MILLISECOND**} **FROM** expresión_aritmética)

Descripción

La función *EXTRACT* devuelve de la *expresión_aritmética* la parte especificada por la palabra clave asociada a la función. La *expresión_aritmética* pasada debe ser del tipo Timestamp.

Ejemplo

Este ejemplo devuelve todos los números de factura desde el mes de enero:

```
SELECT NUM_FACTURA
FROM FACTURAS
WHERE EXTRACT(MONTH(FECHA_FACTURA)) = 1;
```

FLOOR

FLOOR (*expresión_aritmética*)

Descripción

La función *FLOOR* devuelve el número entero mayor que sea inferior o igual a la *expresión_aritmética*.

Ejemplo

Este ejemplo devuelve el entero mayor inferior o igual a -20.9:

```
FLOOR (-20.9);
`devuelve -21
```

HOUR

HOUR (*expresión_aritmética)*

Descripción

La función *HOUR* devuelve la parte "hora" de una hora pasada en la *expresión_aritmética*. El valor devuelto va de 0 a 23.

Ejemplo

Imagine que tenemos la tabla FACTURAS con un campo Hora_Entrega. Para mostrar la hora de entrega:

```
SELECT HOUR(Hora_Entrega)
FROM FACTURAS;
```

INSERT

INSERT (*expresión_aritmética*, *expresión_aritmética*, *expresión_aritmética*, *expresión_aritmética*)

Descripción

La función **INSERT** inserta una cadena en otra en una posición dada. La primera *expresión_aritmética* pasada es la cadena de destino. La segunda *expresión_aritmética* es el índice donde la cadena pasada en la cuarta *expresión_aritmética* se insertará y la tercera *expresión_aritmética* indica el número de caracteres a eliminar en un punto de inserción dado.

Ejemplo

Este ejemplo insertará "Querido " en frente de los nombres en la tabla PERSONAS:

```
SELECT INSERT (PERSONAS.Nombre,0,0,'Querido ') FROM PERSONAS;
```

LEFT

LEFT (expresión_aritmética, expresión_aritmética)

Descripción

La función *LEFT* devuelve la parte más a la izquierda de la *expresión_aritmética* pasada. La segunda *expresión_aritmética* indica el número de caracteres a devolver a partir de los extraídos de la primera *expresión_aritmética* indicada.

Ejemplo

Este ejemplo devuelve los nombres y las dos primeras letras de los apellidos de la tabla PERSONAS:

```
SELECT Nombre, LEFT(Apellido, 2)
FROM PERSONAS;
```

LENGTH

LENGTH (*expresión_aritmética*)

Descripción

La función **LENGTH** devuelve el número de caracteres de la *expresión_aritmética*.

Ejemplo

Este ejemplo devuelve el número de caracteres de los nombres de los productos que pesan menos de 15 kg.

```
SELECT LENGTH (Nombre)
FROM PRODUCTOS
WHERE Peso < 15.00
```

LOCATE

LOCATE (*expresión_aritmética, expresión_aritmética, expresión_aritmética*)

LOCATE (*expresión_aritmética, expresión_aritmética*)

Descripción

La función *LOCATE* devuelve la posición inicial de la primera ocurrencia de una *expresión_aritmética* que se encuentra dentro de un segundo *expresión_aritmética*. También puede pasar una tercera *expresión_aritmética* para especificar la posición del carácter donde la búsqueda debe comenzar.

Ejemplo

Este ejemplo devuelve la posición de la primera letra X que se encuentra entre los apellidos de la tabla PERSONAS:

```
SELECT Nombre, LOCATE('X',Apellido)
FROM PERSONAS;
```

LOG

LOG (*expresión_aritmética*)

Descripción

La función **LOG** devuelve el logaritmo neperiano o logaritmo natural de la *expresión_aritmética*.

Ejemplo

Este ejemplo devuelve el logaritmo neperiano de 50:

```
SELECT LOG( 50 );
```

LOG10

LOG10 (*expresión_aritmética*)

Descripción

La función *LOG10* devuelve el logaritmo decimal (logaritmo base 10) de la *expresión_aritmética*.

Ejemplo

Este ejemplo devuelve el logaritmo decimal del valor 50:

```
SELECT LOG10( 50 );
```

LOWER

LOWER (*expresión_aritmética*)

Description

La función *LOWER* devuelve la cadena *expresión_aritmética* con todos los caracteres convertidos en minúsculas.

Ejemplo

Este ejemplo devuelve los nombres de los productos en minúsculas:

```
SELECT LOWER (Nombre)
FROM PRODUCTOS;
```

LTRIM

LTRIM (*expresión_aritmética*[, *expresión_aritmética*])

Descripción

La función *LTRIM* elimina los posibles espacios vacíos ubicados al inicio de la *expresión_aritmética*. La segunda *expresión_aritmética* opcional permite designar los caracteres especificos a borrar al inicio de la primera *expresión_aritmética*.

Ejemplo

Este ejemplo borra todo espacio ubicado al inicio del nombre de los productos:

```
SELECT LTRIM(Nombre)
FROM PRODUCTOS;
```

MAX

MAX (*expresión_aritmética*)

Descripción

La función **MAX** devuelve el valor máximo de la *expresión_aritmética*.

Ejemplo

Ver los ejemplos de las funciones **SUM** y *AVG*.

MILLISECOND

MILLISECOND (*expresión_aritmética*)

Descripción

La función *MILLISECOND* devuelve la parte de milisegundos de la hora pasada en *expresión_aritmética*.

Ejemplo

Supongamos que tenemos la tabla FACTURAS con un campo Hora_Entrega. Para mostrar los milisegundos de la hora de entrega:

```
SELECT MILLISECOND(Hora_Entrega)
FROM FACTURAS;
```

MIN

MIN (*expresión_aritmética*)

Descripción

La función **MIN** devuelve el valor mínimo de la *expresión_aritmética*.

Ejemplo

Ver los ejemplos de las funciones **SUM** y *AVG*.

MINUTE

MINUTE (*expresión_aritmética*)

Descripción

La función *MINUTE* devuelve la parte de minutos de la hora pasada en *expresión_aritmética*. El valor devuelto va de 0 a 59.

Ejemplo

Supongamos que tenemos la tabla FACTURAS con un campo Hora_Entrega. Para mostrar los minutos de la hora de entrega:

```
SELECT MINUTE(Hora_Entrega)
FROM FACTURAS;
```

MOD

MOD (*expresión_aritmética, expresión_aritmética*)

Descripción

La función **MOD** devuelve el resto de la división de la primera *expresión_aritmética* dividida por la segunda *expresión_aritmética*.

Ejemplo

Este ejemplo devuelve el resto de la división de 3 entre 10:

```
MOD(10,3) `devuelve 1
```

MONTH

MONTH (*expresión_aritmética*)

Descripción

La función *MONTH* devuelve el número del mes (de 1 a 12) de la fecha pasada en la *expresión_aritmética*.

Ejemplo

Supongamos que tenemos la tabla PERSONAS con un campo Fecha_Nacimiento. Para obtener el mes de la fecha de nacimiento de cada persona:

```
SELECT MONTH(Fecha_Nacimiento)
FROM PERSONAS;
```

MONTHNAME

MONTHNAME (*expresión_aritmética*)

Descripción

La función *MONTHNAME* devuelve el nombre del mes de la fecha pasada en la *expresión_aritmética*.

Ejemplo

Este ejemplo devuelve el nombre del mes de cada fecha de nacimiento:

```
SELECT MONTHNAME(Fecha_Nacimiento);
```

NULLIF

NULLIF (*expresión_aritmética*, *expresión_aritmética*)

Descripción

La función *NULLIF* devuelve NULL si la primera *expresión_aritmética* es igual a la segunda. De lo contrario, se devuelve el valor de la primera *expresión_aritmética*. Las dos expresiones deben ser comparables.

Ejemplo

Este ejemplo devuelve NULL si el total de la factura es 0:

```
NULLIF(FACTURAS_TOTAL,0);
```

OCTET_LENGTH

OCTET_LENGTH (e*xpresión_aritmética*)

Descripción

La función *OCTET_LENGTH* devuelve el número de bytes de *expresión_aritmética*, incluyendo los posibles espacios.

Ejemplo

Este ejemplo devuelve el número de bytes para una columna de datos binarios:

```
SELECT OCTET_LENGTH (MiCol_binaria)
FROM MiTabla
WHERE MiCol_binaria = '93FB';
` devuelve 2
```

PI

PI ( )

Descripción

La función *PI* devuelve el valor de la constante Pi.

Ejemplo

Ver el ejemplo de la función *DEGREES*.

POSITION

POSITION (*expresión_aritmética* **IN** *expresión_aritmética*)

Descripción

La función **POSITION** devuelve un valor indicando la posición de la primera *expresión_aritmética* dentro de la segunda *expresión_aritmética*. Si no se encuentra la cadena, la función devuelve cero.

Ejemplo

Este ejemplo devuelve la posición inicial de la palabra "York" en todos los apellidos en la tabla PERSONAS:

```
SELECT Nombre, POSITION('York' IN Apellido)
FROM PERSONAS;
```

POWER

POWER (*expresión_aritmética, expresión_aritmética*)

Descripción

La función *POWER* eleva la primera *expresión_aritmética* pasada a la potencia "x", donde "x" es la segunda *expresión_aritmética* pasada.

Ejemplo

Este ejemplo eleva cada valor a la potencia 3:

```
SELECT SourceValues, POWER(SourceValues, 3)
FROM Values
ORDER BY SourceValues
`devuelve 8 para SourceValues = 2
```

QUARTER

QUARTER (*expresión_aritmética*)

Description

La función *QUARTER* devuelve el trimestre del año (de 1 a 4) en el cual aparece la fecha pasada en *expresión_aritmética*.

Ejemplo

Supongamos que tenemos la tabla PERSONAS con un campo Fecha_Nacimiento. Para obtener el trimestre de la fecha de nacimiento de cada persona:

```
SELECT QUARTER(Fecha_Nacimiento)
FROM PERSONAS;
```

RADIANS

RADIANS (*arithmetic_expression*)

Descripción

La función *RADIANS* devuelve el número de radianes de la *expresión_aritmética*. La *expresión_aritmética* representa el ángulo expresado en grados.

Ejemplo

Este ejemplo devuelve el número de radianes de un ángulo de 30 grados:

```
RADIANS ( 30 );
`devuelve el valor 0,5236
```

RAND

RAND ([*expresión_aritmética*])

Descripción

La función **RAND** devuelve un valor de tipo flotante aleatorio entre 0 y 1. La *expresión_aritmética* opcional puede utilizarse para pasar un valor clave.

Ejemplo

Este ejemplo inserta los valores de ID generados por la función **RAND**:

```
CREATE TABLE PERSONAS
(ID INT32,
Nombre VARCHAR);

INSERT INTO PERSONAS
(ID, Nombre)
VALUES(RAND, 'Francis');
```

REPEAT

REPEAT (*expresión_aritmética*, *expresión_aritmética*)

Descripción

La función *REPEAT* devuelve la primera *expresión_aritmética* repetida el número de veces definido por la segunda *expresión_aritmética*).

Ejemplo

Este ejemplo muestra cómo funciona:

```
SELECT REPEAT( 'repetir', 3 )
` devuelve 'repetirrepetirrepetir'
```

REPLACE

REPLACE (*expresión_aritmética*, *expresión_aritmética*, *expresión_aritmética*)

Descripción

La función *REPLACE* busca en la primera *expresión_aritmética* pasada todas las ocurrencias de la segunda *expresión_aritmética* y reemplaza cada ocurrencia encontrada por la tercera *expresión_aritmética* pasada. Si no se encuentra ninguna ocurrencia, la primera *expresión_aritmética* permanece igual.

Ejemplo

Este ejemplo reemplazará la cadena "Francos" por "Euros" en la columna Divisa:

```
SELECT Nom, REPLACE(Divisa, 'Francos', 'Euros')
FROM PRODUCTOS;
```

RIGHT

RIGHT (*expresión_aritmética, expresión_aritmética*)

Descripción

La función *RIGHT* devuelve la parte más a la derecha de la *expresión_aritmética*. La segunda *expresión_aritmética* indica el número de caracteres a devolver a partir de la derecha de la primera *expresión_aritmética*.

Ejemplo

Este ejemplo devuelve los nombres y los dos primeros caracteres de los nombres de la tabla PERSONAS:

```
SELECT Nombre, RIGHT(Apellido, 2)
FROM PERSONAS;
```

ROUND

ROUND (*expresión_aritmética[, expresión_aritmética*])

Descripción

La función **ROUND** redondea el valor pasado en la primera *expresión_aritmética* a "x" decimales (donde "x" es la segunda *expresión_aritmética* opcional pasada). Si no se pasa la segunda *expresión_aritmética*, la *expresión_aritmética* se redondea al número entero más próximo.

Ejemplo

Este ejemplo redondea el número de origen a dos números decimales:

```
ROUND (1234.1966, 2)
`devuelve1234.2000
```

RTRIM

RTRIM (*expresión_aritmética*[, *expresión_aritmética*])

Descripción

La función *RTRIM* borra los posibles espacios vacíos ubicados al final de la *expresión_aritmética*. La segunda *expresión_aritmética* opcional permite designar los caracteres específicos a borrar al final de la primera *expresión_aritmética*.

Ejemplo

Este ejemplo borra todo espacio vacío del final de los nombres de los productos:

```
SELECT RTRIM(Nombre)
FROM PRODUCTOS;
```

SECOND

SECOND (*expresión_aritmética*)

Descripción

La función *SECOND* devuelve la parte segundos (de 0 a 59) del tiempo pasado en la *expresión_aritmética*.

Ejemplo

Supongamos que tenemos la tabla FACTURAS con un campo Hora_Entrega. Para mostrar los segundos de la hora de entrega:

```
SELECT SECOND(Hora_Entrega)
FROM FACTURAS;
```

SIGN

SIGN (*expresión_aritmética*)

Descripción

La función *SIGN* devuelve el signo de la *expresión_aritmética* (1 para un número positivo, -1 para un número negativo o 0).

Ejemplo

Este ejemplo devuelve todas las cantidades negativas que se encuentran en la tabla FACTURAS:

```
SELECT AMOUNT
FROM FACTURAS
WHERE SIGN(CANTIDAD) = -1;
```

SIN

SIN (*expresión_aritmética*)

Descripción

La función **SIN** devuelve el seno de la *expresión_aritmética*. La *expresión_aritmética* representa el ángulo expresado en radianes.

Ejemplo

Este ejemplo devuelve el seno del ángulo en radianes:

```
SELECT SIN(radians)
FROM TABLAS_DE_ANGULOS;
```

SPACE

SPACE (*expresión_aritmética*)

Descripción

La función *SPACE* devuelve una cadena de caracteres del número de espacios definido en la *expresión_aritmética*. Si el valor de la *expresión_aritmética* es menor de cero, se devolverá un valor NULL.

Ejemplo

Este ejemplo añade tres espacios delante de los apellidos de la tabla PERSONAS:

```
SELECT CONCAT(SPACE(3),PERSONAS.Apellido) FROM PERSONAS;
```

SQRT

SQRT (*expresión_aritmética*)

Descripción

La función *SQRT* devuelve la raíz cuadrada de la *expresión_aritmética*.

Ejemplo

Este ejemplo devuelve la raíz cuadrada del flete:

```
SELECT Flete, SQRT(Flete) AS "Raíz cuadrada del flete"
FROM Ordenes
```

SUBSTRING

SUBSTRING (*expresión_aritmética*, *expresión_aritmética*, [*expresión_aritmética*])

Descripción

La función **SUBSTRING** devuelve una subcadena de la primera *expresión_aritmética* pasada. La segunda *expresión_aritmética* indica la posición de inicio de la subcadena y la tercera *expresión_aritmética* opcional indica el número de parámetros a devolver a partir de la posición de inicio. Si la tercera *expresión_aritmética* se omite, la función devolverá todos los caracteres a partir de la posición de indicada.

Ejemplo

Este ejemplo devuelve 4 caracteres del nombre de la tienda a partir del segundo carácter:

```
SELECT SUBSTRING(Nombre_tienda,2,4)
FROM Geography
WHERE Nombre_tienda = 'Paris';
```

SUM

SUM ([**ALL** |**DISTINCT**] *expresión_aritmética*)

Descripción

La función **SUM** devuelve la suma de la *expresión_aritmética*. Las palabras claves opcionales **ALL** y **DISTINCT** permiten respectivamente incluir o eliminar los posibles valores duplicados.

Ejemplo

Este ejemplo devuelve la suma de las ventas esperadas menos la suma de las ventas realizadas, como también la cantidad mínima y máxima de ventas realizadas multiplicada por 100 y dividida por las ventas esperadas en la tabla VENDEDORES:

```
SELECT MIN ( ( SALES * 100 ) / QUOTA ),
MAX ( ( SALES * 100 ) / QUOTA ),
SUM ( QUOTA ) - SUM ( SALES )
FROM VENDEDORES
```

TAN

TAN (*expresión_aritmética*)

Descripción

La función **TAN** devuelve la tangente de la *expresión_aritmética*. La *expresión_aritmética* representa el ángulo expresado en radianes.

Ejemplo

Este ejemplo devuelve la tangente del ángulo expresado en radianes:

```
SELECT TAN(radianes)
FROM TABLAS_DE_ANGULOS;
```

TRANSLATE

TRANSLATE (*expresión_aritmética, expresión_aritmética, expresión_aritmética)*

Descripción

La función **TRANSLATE** devuelve la primera *expresión_aritmética* con todas las ocurrencias de cada uno de los caracteres pasados en la segunda *expresión_aritmética* reemplazados por sus caracteres correspondientes pasados en la tercera *expresión_aritmética*.

El reemplazo se efectúa carácter por carácter (por ejemplo, el primer carácter de la segunda *expresión_aritmética* se reemplaza cada vez que aparece en la primera *expresión_aritmética* por el primer carácter de la tercera *expresión_aritmética* y así sucesivamente).

Si hay menos caracteres en la tercera *expresión_aritmética* que en la segunda, todas las ocurrencias de caracteres de la segunda *expresión_aritmética* que no tengan un carácter correspondiente en la tercera *expresión_aritmética* serán eliminados de la primera *expresión_aritmética* (por ejemplo, si la segunda *expresión_aritmética* tiene cinco caracteres a buscar y la tercera *expresión_aritmética* sólo contiene cuatro caracteres de reemplazo, cada vez que el quinto carácter de la segunda *expresión_aritmética* se encuentre en la primera *expresión_aritmética*, se eliminará del valor devuelto).

Ejemplo

Este ejemplo reemplaza todas las ocurrencias de "a" por "1" y las ocurrencias de "b" por "2":

```
TRANSLATE ('abcd', 'ab', '12')
` devuelve '12cd'
```

TRIM

TRIM ([[**LEADING** |**TRAILING** |**BOTH**] [*expresión_aritmética*] **FROM**]*expresión_aritmética*)

Descripción

La función **TRIM** suprime de las extremidades de la *expresión_aritmética* los espacios vacíos o caracteres especificados en la primera *expresión_aritmética*.

Puede pasar **LEADING** para indicar que los espacios/caracteres deben eliminarse al principio de la *expresión_aritmética* únicamente, **TRAILING** con el fin de indicar los que deben eliminarse al final de la expresión o **BOTH** para combinar las dos operaciones. No pasar ninguna de estas palabras claves, equivale a pasar **BOTH** (los espacios o caracteres se eliminan al principio y al final de la *expresión_aritmética*).

La primera *expresión_aritmética* opcional pasada indica los caracteres específicos a eliminar individualmente de la segunda *expresión_aritmética*. Si se omite, sólo se eliminarán los espacios vacíos.

Ejemplo

Este ejemplo borra los espacios de los nombres de los productos:

```
SELECT TRIM(Nombre)
FROM PRODUCTOS;
```

TRUNC

TRUNC (*expresión_aritmética*[, *expresión_aritmética*])

Descripción

La función **TRUNC** devuelve la primera *expresión_aritmética* truncada en "x" cifras a la derecha del separador decimal, donde "x" es el valor pasado en la segunda *expresión_aritmética* opcional. Si se omite la segunda *expresión_aritmética*, la primera *expresión_aritmética* se trunca eliminando la parte decimal.

Ejemplo

Esta función trunca el número pasado a una cifra después del separador decimal:

```
TRUNC(2.42 , 1)
`devuelve 2.40
```

TRUNCATE

TRUNCATE (*expresión_aritmética*[, *expresión_aritmética*])

Descripción

La función *TRUNCATE* devuelve la primera expresión *expresión_aritmética* truncada en "x" cifras a la derecha del separador decimal, donde "x" es el valor pasado en la segunda *expresión_aritmética* opcional. Si la segunda *expresión_aritmética* no se pasa, la primera *expresión_aritmética* se trunca eliminando los decimales.

Ejemplo

Ver el ejemplo de la función **TRUNC**.

UPPER

UPPER (*expresión_aritmética*)

Descripción

La función *UPPER* devuelve la *expresión_aritmética* pasada como una cadena donde todos los caracteres están en mayúsculas.

Ejemplo

Este ejemplo devuelve los nombres de los productos en mayúsculas:

```
SELECT UPPER (Nombre)
FROM PRODUCTS;
```

WEEK

WEEK (*expresión_aritmética*)

Descripción

La función *WEEK* devuelve el número de la semana del año (valor entre 1 y 54) de la fecha pasada en la *expresión_aritmética*. La semana comienza el domingo y el primero de enero es siempre la primera semana.

Ejemplo

Este ejemplo devuelve un número que indica la semana del año para cada fecha de nacimiento pasada:

```
SELECT WEEK(Fecha_nacimiento);
```

YEAR

YEAR (expresión_*aritmética*)

Descripción

La función *YEAR* devuelve la parte del año de la fecha pasada en la *expresión_aritmética*.

Ejemplo

Supongamos que tenemos la tabla PERSONAS con un campo Fecha_Nacimiento. Para obtener el año de la fecha de nacimiento de cada persona:

```
SELECT YEAR(Fecha_Nacimiento)
FROM PERSONAS;
```

Anexos

Anexo A: Códigos de errores SQL

Anexo A: Códigos de errores SQL

El motor SQL de 4D devuelve los errores específicos, listados a continuación. Estos errores pueden interceptarse utilizando un método de gestión de errores instalado por el comando **ON ERR CALL**.

Errores genéricos

1001	INVALID ARGUMENT
1002	INVALID INTERNAL STATE
1003	SQL SERVER IS NOT RUNNING
1004	Access denied
1005	FAILED TO LOCK SYNCHRONIZATION PRIMITIVE
1006	FAILED TO UNLOCK SYNCHRONIZATION PRIMITIVE
1007	SQL SERVER IS NOT AVAILABLE
1008	COMPONENT BRIDGE IS NOT AVAILABLE
1009	REMOTE SQL SERVER IS NOT AVAILABLE
1010	EXECUTION INTERRUPTED BY USER

Errores semánticos

1101 Table '{key1}' does not exist in the database.
1102 Column '{key1}' does not exist.
1103 Table '{key1}' is not declared in the FROM clause.
1104 Column name reference '{key1}' is ambiguous.
1105 Table alias '{key1}' is the same as table name.
1106 Duplicate table alias - '{key1}'.
1107 Duplicate table in the FROM clause - '{key1}'.
1108 Operation {key1} {key2} {key3} is not type safe.
1109 Invalid ORDER BY index - {key1}.
1110 Function {key1} expects one parameter, not {key2}.
1111 Parameter {key1} of type {key2} in function call {key3} is not implicitly convertible to {key4}.
1112 Unknown function - {key1}.
1113 Division by zero.
1114 Sorting by indexed item in the SELECT list is not allowed - ORDER BY item {key1}.
1115 DISTINCT NOT ALLOWED
1116 Nested aggregate functions are not allowed in the aggregate function {key1}.
1117 Column function is not allowed.
1118 Cannot mix column and scalar operations.
1119 Invalid GROUP BY index - {key1}.
1120 GROUP BY index is not allowed.
1121 GROUP BY is not allowed with 'SELECT * FROM ...'.
1122 HAVING is not an aggregate expression.
1123 Column '{key1}' is not a grouping column and cannot be used in the ORDER BY clause.
1124 Cannot mix {key1} and {key2} types in the IN predicate.
1125 Escape sequence '{key1}' in the LIKE predicate is too long. It must be exactly one character.
1126 Bad escape character - '{key1}'.
1127 Unknown escape sequence - '{key1}'.
1128 Column references from more than one query in aggregate function {key1} are not allowed.
1129 Scalar item in the SELECT list is not allowed when GROUP BY clause is present.
1130 Sub-query produces more than one column.
1131 Subquery must return one row at the most but instead it returns {key1}.
1132 INSERT value count {key1} does not match column count {key2}.
1133 Duplicate column reference in the INSERT list - '{key1}'.
1134 Column '{key1}' does not allow NULL values.
1135 Duplicate column reference in the UPDATE list - '{key1}'.
1136 Table '{key1}' already exists.
1137 Duplicate column '{key1}' in the CREATE TABLE command.
1138 DUPLICATE COLUMN IN COLUMN LIST
1139 More than one primary key is not allowed.
1140 Ambiguous foreign key name - '{key1}'.
1141 Column count {key1} in the child table does not match column count {key2} in the parent table of the foreign key definition.
1142 Column type mismatch in the foreign key definition. Cannot relate {key1} in child table to {key2} in parent table.
1143 Failed to find matching column in parent table for '{key1}' column in child table.
1144 UPDATE and DELETE constraints must be the same.
1145 FOREIGN KEY DOES NOT EXIST
1146 Invalid LIMIT value in SELECT command - {key1}.
1147 Invalid OFFSET value in SELECT command - {key1}.
1148 Primary key does not exist in table '{key1}'.

1149 FAILED TO CREATE FOREIGN KEY
1150 Column '{key1}' is not part of a primary key.
1151 FIELD IS NOT UPDATEABLE
1152 FOUND VIEW COLUMN
1153 Bad data type length '{key1}'.
1154 EXPECTED EXECUTE IMMEDIATE COMMAND
1155 INDEX ALREADY EXISTS
1156 Auto-increment option is not allowed for column '{key1}' of type {key2}.
1157 SCHEMA ALREADY EXISTS
1158 SCHEMA DOES NOT EXIST
1159 Cannot drop system schema
1160 CHARACTER ENCODING NOT ALLOWED

Errores de implementación

1203 The functionality is not implemented.
1204 Failed to create record {key1}.
1205 Failed to update field '{key1}'.
1206 Failed to delete record '{key1}'.
1207 NO MORE JOIN SEEDS POSSIBLE
1208 FAILED TO CREATE TABLE
1209 FAILED TO DROP TABLE
1210 CANT BUILD BTREE FOR ZERO RECORDS
1211 COMMAND COUNT GREATER THAN ALLOWED
1212 FAILED TO CREATE DATABASE
1213 FAILED TO DROP COLUMN
1214 VALUE IS OUT OF BOUNDS
1215 FAILED TO STOP SQL_SERVER
1216 FAILED TO LOCALIZE
1217 Failed to lock table for reading.
1218 FAILED TO LOCK TABLE FOR WRITING
1219 TABLE STRUCTURE STAMP CHANGED
1220 FAILED TO LOAD RECORD
1221 FAILED TO LOCK RECORD FOR WRITING
1222 FAILED TO PUT SQL LOCK ON A TABLE
1223 FAILED TO RETAIN COOPERATIVE TASK
1224 FAILED TO LOAD INFILE

Error de análisis

1301 PARSING FAILED

Errores de acceso al lenguaje runtime

1401 COMMAND NOT SPECIFIED
1402 ALREADY LOGGED IN
1403 SESSION DOES NOT EXIST
1404 UNKNOWN BIND ENTITY
1405 INCOMPATIBLE BIND ENTITIES
1406 REQUEST RESULT NOT AVAILABLE
1407 BINDING LOAD FAILED
1408 COULD NOT RECOVER FROM PREVIOUS ERRORS
1409 NO OPEN STATEMENT
1410 RESULT EOF
1411 BOUND VALUE IS NULL
1412 STATEMENT ALREADY OPENED
1413 FAILED TO GET PARAMETER VALUE
1414 INCOMPATIBLE PARAMETER ENTITIES
1415 Query parameter is either not allowed or was not provided.
1416 COLUMN REFERENCE PARAMETERS FROM DIFFERENT TABLES
1417 EMPTY STATEMENT
1418 FAILED TO UPDATE VARIABLE
1419 FAILED TO GET TABLE REFERENCE
1420 FAILED TO GET TABLE CONTEXT
1421 COLUMNS NOT ALLOWED
1422 INVALID COMMAND COUNT
1423 INTO CLAUSE NOT ALLOWED
1424 EXECUTE IMMEDIATE NOT ALLOWED
1425 ARRAY NOT ALLOWED IN EXECUTE IMMEDIATE
1426 COLUMN NOT ALLOWED IN EXECUTE IMMEDIATE
1427 NESTED BEGIN END SQL NOT ALLOWED
1428 RESULT IS NOT A SELECTION
1429 INTO ITEM IS NOT A VARIABLE
1430 VARIABLE WAS NOT FOUND
1431 PTR OF PTR NOT ALLOWED
1432 POINTER OF UNKNOWN TYPE

Errores de análisis de fecha

1501 SEPARATOR_EXPECTED
1502 FAILED TO PARSE DAY OF MONTH
1503 FAILED TO PARSE MONTH
1504 FAILED TO PARSE YEAR
1505 FAILED TO PARSE HOUR
1506 FAILED TO PARSE MINUTE
1507 FAILED TO PARSE SECOND
1508 FAILED TO PARSE MILLISECOND
1509 INVALID AM PM USAGE
1510 FAILED TO PARSE TIME ZONE
1511 UNEXPECTED CHARACTER
1512 Failed to parse timestamp.
1513 Failed to parse duration.
1551 FAILED TO PARSE DATE FORMAT

Errores lexer

1601 NULL INPUT STRING
1602 NON TERMINATED STRING
1603 NON TERMINATED COMMENT
1604 INVALID NUMBER
1605 UNKNOWN START OF TOKEN
1606 NON TERMINATED NAME
1607 NO VALID TOKENS

Errores de validación - Errores de estado después de errores directos

1701 Failed to validate table '{key1}'.
1702 Failed to validate FROM clause.
1703 Failed to validate GROUP BY clause.
1704 Failed to validate SELECT list.
1705 Failed to validate WHERE clause.
1706 Failed to validate ORDER BY clause.
1707 Failed to validate HAVING clause.
1708 Failed to validate COMPARISON predicate.
1709 Failed to validate BETWEEN predicate.
1710 Failed to validate IN predicate.
1711 Failed to validate LIKE predicate.
1712 Failed to validate ALL ANY predicate.
1713 Failed to validate EXISTS predicate.
1714 Failed to validate IS NULL predicate.
1715 Failed to validate subquery.
1716 Failed to validate SELECT item {key1}.
1717 Failed to validate column '{key1}'.
1718 Failed to validate function '{key1}'.
1719 Failed to validate CASE expression.
1720 Failed to validate command parameter.
1721 Failed to validate function parameter {key1}.
1722 Failed to validate INSERT item {key1}.
1723 Failed to validate UPDATE item {key1}.
1724 Failed to validate column list.
1725 Failed to validate foreign key.
1726 Failed to validate SELECT command.
1727 Failed to validate INSERT command.
1728 Failed to validate DELETE command.
1729 Failed to validate UPDATE command.
1730 Failed to validate CREATE TABLE command.
1731 Failed to validate DROP TABLE command.
1732 Failed to validate ALTER TABLE command.
1733 Failed to validate CREATE INDEX command.
1734 Failed to validate LOCK TABLE command.
1735 Failed to calculate LIKE predicate pattern.

Errores de ejecución - Errores de estado después de errores directos

1801	Failed to execute SELECT command.
1802	Failed to execute INSERT command.
1803	Failed to execute DELETE command.
1804	Failed to execute UPDATE command.
1805	Failed to execute CREATE TABLE command.
1806	Failed to execute DROP TABLE command.
1807	Failed to execute CREATE DATABASE command.
1808	Failed to execute ALTER TABLE command.
1809	Failed to execute CREATE INDEX command.
1810	Failed to execute DROP INDEX command.
1811	Failed to execute LOCK TABLE command.
1812	Failed to execute TRANSACTION command.
1813	Failed to execute WHERE clause.
1814	Failed to execute GROUP BY clause.
1815	Failed to execute HAVING clause.
1816	Failed to aggregate.
1817	Failed to execute DISTINCT.
1818	Failed to execute ORDER BY clause.
1819	Failed to build DB4D query.
1820	Failed to calculate comparison predicate.
1821	Failed to execute subquery.
1822	Failed to calculate BETWEEN predicate.
1823	Failed to calculate IN predicate.
1824	Failed to calculate ALL/ANY predicate.
1825	Failed to calculate LIKE predicate.
1826	Failed to calculate EXISTS predicate.
1827	Failed to calculate NULL predicate.
1828	Failed to perform arithmetic operation.
1829	Failed to calculate function call '{key1}'.
1830	Failed to calculate case expression.
1831	Failed to calculate function parameter '{key1}'.
1832	Failed to calculate 4D function call.
1833	Failed to sort while executing ORDER BY clause.
1834	Failed to calculate record hash.
1835	Failed to compare records.
1836	Failed to calculate INSERT value {key1}.
1837	DB4D QUERY FAILED
1838	FAILED TO EXECUTE ALTER SCHEMA COMMAND
1839	FAILED TO EXECUTE GRANT COMMAND

Errores de caché

2000	CACHEABLE NOT INITIALIZED
2001	VALUE ALREADY CACHED
2002	CACHED VALUE NOT FOUND
2003	CACHE IS FULL
2004	CACHING IS NOT POSSIBLE

Errores de protocolo

3000 HEADER NOT FOUND
3001 UNKNOWN COMMAND
3002 ALREADY LOGGED IN
3003 NOT LOGGED IN
3004 UNKNOWN OUTPUT MODE
3005 INVALID STATEMENT ID
3006 UNKNOWN DATA TYPE
3007 STILL LOGGED IN
3008 SOCKET READ ERROR
3009 SOCKET WRITE ERROR
3010 BASE64 DECODING ERROR
3011 SESSION TIMEOUT
3012 FETCH TIMESTAMP ALREADY EXISTS
3013 BASE64 ENCODING ERROR
3014 INVALID HEADER TERMINATOR
3015 INVALID SESSION TICKET
3016 HEADER TOO LONG
3017 INVALID AGENT SIGNATURE
3018 UNEXPECTED HEADER VALUE