# Contents

# INTRODUCCIÓN

## Sentencias

- **Commit, Rollback:** Sentencias que permiten guardar los cambios realizados (con sentencias DML) o revertirlos.
- **Comentarios:** Se pueden hacer con "/*comentario*/" o con "-- comentario".
- **Terminador:** Se termina un bloque de código con el término ";".

## Tipos de datos

- **Numéricos:**
  - Enteros:
    - **SMALLINT:** -32000 hasta 32000, aprox.
    - **INT:** -2M hasta 2M, aprox.
    - **BIGINT:** -9 Trillones hasta 9 Trillones, aprox.
  - **FLOAT, DOUBLE:** Decimales (Con errores en grandes cantidades).
  - **NUMERIC y DECIMAL:** Se ponen como NUMERIC(10,4). En los paréntesis "(10,4)" el primer número significa la cantidad de cifras a almacenar en total, el segundo la cantidad de decimales. NUMERIC es estricto, DECIMAL indica "al menos X cifras" para el primer valor del paréntesis.
- **Fecha y Hora:** DATE, TIME, TIMESTAMP
- **Carácter / Cadena:**
  - CHAR: Define una cadena de longitud fija, por ejemplo si hacemos "cast('hola' as char(10))" va a dar de resultado 'hola ' rellenando con espacios hasta llegar a 10 caracteres.
  - VARCHAR: Define una cadena de longitud variable, por ejemplo "cast('hola' as varchar(10))" da de resultado 'hola'.
- **Especiales:** BLOB, ARRAY
- **BOOLEAN**
- **Otros tipos de variables propias de Firebird:**
  - CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP. Estas muestran la fecha actual, el tiempo actual o la fecha y el tiempo actual.
  - También se puede usar NOW, TODAY, TOMORROW y YESTERDAY.

## Conversiones de tipo y NULL

Las conversiones de tipo se hacen con la sentencia "CAST(<dato a convertir> AS <tipo de dato>)"

El valor "NULL" se puede considerar como un *error* en la base de datos, suele ser anomalías, como un propietario sin nombre, o un piso sin precio. Estas anomalías se pueden propagar en las operaciones en las que se usan, si haces por ejemplo un sumatorio de todos los precios de algo y uno es "NULL" entonces el resultado va a ser "NULL". Se suele evitar fácilmente poniendo datos por defecto en vez de dejar las celdas como nulas: Una cadena como '' (vacía), un número como 0, un booleano como false, etc.

Hay varias sentencias para evitar los nulos, entre ellas la más común es "COALESCE(<dato/datos>, <valor al que convertir los nulos>)".

# EXPRESIONES Y PREDICADOS

## Predicados y Operadores lógicos

Predicados:

- (columna) BETWEEN (valor/cadena) AND (valor/cadena)
    - select * from <tabla> where <columna> is between 2 and 5
- (columna) CONTAINING (cadena)
    - select * from <tabla> where <columna> containing 'hola'
- (columna) LIKE (expresión usando caracteres, % y _) [ESCAPE '<caracter>']
    - select * from <tabla> where <columna> like "%algo_"
    - "%" Significa cualquier caracter y cualquier número de caracteres.
    - "_" Significa cualquier carácter, pero solo uno.
- (columna) SIMILAR TO (expresión regular) [ESCAPE '<caracter>']
    - select * from <tabla> where <columna> similar to '[0-9]{3}\-[A-Za-z]{9}' ESCAPE '\'
- ~~IS [NOT] NULL~~, es lo mismo a "= NULL" o "<> NULL"
- ~~IS [NOT]~~, es equivalente a "=" o "<>"
- ~~IS [NOT] DISTINCT~~, es equivalente a "<>" o "="

Operadores lógicos:

- AND, OR y NOT. Estos tipos comprueban si ambas son verdaderas (AND) o si una por lo menos es verdadera (OR). Por orden de operaciones NOT suele ir el primero, OR suele ejecutarse después, y por último AND, pero a las dudas se pueden poner paréntesis.

Predicados existenciales:

- ALL
    - select * from <tabla> where <columna> = all (1,2,3)
    - select * from tabla1 where id <> all (select id from tabla2)
- ANY
    - select * from <tabla> where <columna> = any (1,2,3)
    - select * from tabla1 where id <> any (select id from tabla2)
- [NOT] EXISTS
    - select * from tabla1 t1 where exists (select * from tabla2 t2 where t1.id = t2.id)
- (columna) IN (lista/subconsulta)
    - select id from tabla1 where id in (select id from tabla2)
    - select id from tabla where id not in (1,2,3)

## Expresiones regulares

- "|" se usa para encontrar un valor u otro. Por ejemplo, 'ab|cd' encuentra a**b**d o a**c**d.
- "_" un carácter cualquiera.
- "%" cualquier carácter y cualquier longitud.
- "(…)" agrupar caracteres. Por ejemplo: '(ab)|(cd)' encuentra ab o cd.
- "[…]" cualquier carácter que esté dentro. Por ejemplo: [aAVE], [a-z], [0-9A-Z]
- "[^…]" cualquier carácter **excepto** los especificados.

- Repetición:
    - "*" 0 o más veces. '[12]*'
    - "?" 0 o 1 vez. 'a?'
    - "+" 1 o más veces. 'abc+'
    - "{n}" *n* veces. '[0-9]{3}'
    - "{n,}" *n* veces o más. '[0-9]{2,}'
    - "{n,m}" *n* a *m* veces. '[0-9]{2,4}'

## Funciones

### CASE / DECODE, COALESCE, NULLIF, IIF

Las funciones de "CASE" y "DECODE" son idénticas, es recomendable por Enrique usar "DECODE". En este ejemplo, para id_inquilino, cuando sea 1 se muestra "10", cuando sea 2 se muestra "20"… y para los que no coinciden con ninguno de los especificados se pone el default, en este caso "0".

```
select DECODE(id_inquilino, 1, 10, 2, 20 …, 0) from inquilinos;
```

"IIF" es como un if de toda la vida.

```
select iif(1>0,1,0) from rdb$database;
```

"NULLIF" devuelve nulo si los dos valores especificados son iguales.

```
select NULLIF(1, 1) from rdb$database;
```

"COALESCE" es usado para controlar los nulos. En este ejemplo se muestra el precio si no es nulo, y si sí lo es se muestra "0".

```
select COALESCE(precio, 0) from habitaciones;
```

### Numéricas

Con estos datos se pueden realizar operaciones relacionales (=,<,<=,>,>=,<>) y aritméticas (+,-,/,*) entre otras funciones:

- ABS/CEIL/FLOOR (<numero/columna>) Valor absoluto, redondeado al mayor o al menor.
- ROUND/TRUNC(<numero/columna> [, escala]) Redondea o trunca un número con tantos decimales como se le indique en la escala (por defecto 0 decimales).
- MAXVALUE/MINVALUE(num1, num3, …) Valor máximo o mínimo de la lista expecificada.
- MOD(num1,num2) Resto de la división entera de num1 entre num2.
- PI() Devuelve el número PI.
- POWER(num1, num2) La potencia de num1 elevado a num2.
- RAND() Devuelve un número aleatorio entre 0 y 1.
- SIGN(<numero>) Devuelve 1 (número positivo), 0 (número es 0) o -1 (número negativo).
- SQRT(<numero>) Raíz cuadrada del número.

## Cadenas

- Concatenación de caracteres, se lleva a cabo escribiendo "||" entre los dos textos, funciones o expresiones a concadenar, por ejemplo ('texto 1 ' || 'texto 2').
- Se pueden comprobar si dos cadenas son iguales o diferentes con "=, <>".

Funciones con cadenas:

- ASCII_CHAR(numero) y ASCII_VAL(cadena), devuelven el numero de la posición ascii o el carácter ascii correspondiente.
- LEFT/RIGHT(<cadena>, <numero>), acortan la cadena con tantas letras como le especifiques, por ejemplo: left('hola', 2) = 'ho'; right('hola', 3) = 'ola'
- LOWER/UPPER(<cadena>), devuelven la cadena en mayúsculas o minúsculas.
- LPAD/RPAD(<cadena>, <longitud> [, <cadenaAgregar>]), añade la segunda cadena especificada a la izquierda o derecha de la primera hasta que el tamaño de la cadena sea el que se indica en "longitud".
- OVERLAY(<cadena> PLACING <cadenaSustituir> FROM <inicio> [FOR <longitud>]) Este reemplaza en la cadena original usando la segunda cadena especificada, se escribe por encima empezando en "inicio" y terminando cuando se ponga la cadena entera o cuando llegue al limite de "longitud", sin repetirse si la cadenaSustituir es menor a longitud.
    - Por ejemplo: OVERLAY('hola caracola' PLACING 'vampiro ' FROM 6) da de resultado 'hola vampiro '.
    - Otro ejemplo: OVERLAY('Maria Angeles' PLACING '' FROM 6 FOR 2) daría de resultado "Mariangeles".
- POSITION(<cadenaAEncontrar> IN <cadena>), devuelve la posición de la primera coincidencia de "cadenaAEncontrar" en "cadena".
- REPLACE(<cadena>, <cadenaABuscar>, <cadenaReemplazo>), reemplaza "cadenaABuscar" en "cadena" por "cadenaReemplazo".
- REVERSE(<cadena>), devuelve la misma cadena, pero invertida.
- SUBSTRING(<valor> FROM <posicion> [FOR <longitud>]), devuelve el carácter o cadena de caracteres en cierta posición de la cadena original. Por ejemplo: SUBSTRING('hola' FROM 2 FOR 2) devolverá 'ol'.
- TRIM([[LEADING|TRAILING|BOTH] [<cadenaBorrar>] FROM] <cadena>) Permite borrar una cadena de la parte derecha (trailing) o izquierda (leading), o ambas (both) de otra cadena. Por defecto si no se especifica será "BOTH" y la cadena " ", en otras palabras, por defecto remueve los espacios de la derecha e izquierda de una cadena.
    - Por ejemplo: TRIM('    hola caracola    ') da de resultado "hola caracola"
    - Otro ejemplo: TRIM(LEADING '-' FROM '-adios-') da de resultado "adios-"

## Fechas

- EXTRACT(<elemento> FROM <fecha/hora/timestamp>)
    - Elemento puede ser year, month, day, hour, minute, second, weekday, yearday, week.
- DATEADD(<numero> <parte> TO <fecha/hora/timestamp>), también hay un sintaxis con comas "DATEADD(parte, numero, fecha). "parte" puede ser year, month, day, hour, minute, second o millisecond. "numero" puede ser positivo o negativo. "fecha" es la fecha a la que añadir o restar tiempo.

### Booleanos

- Con estos tipos de datos podemos hacer operaciones lógicas, para ello tenemos dos tipos principales:
    - =, <>: Estos tipos comprueban si ambas son Verdaderas / Falsas, o si son distintas. A efectos prácticos funcionan como un XOR.
    - AND, OR, NOT: Explicados posteriormente, es lo mismo que en Java.

### Funciones de agregación

Estas funciones van de la mano de las sentencias "Group by"

"COUNT([DISTINCT] <columna> | *)" Contar las celdas, si se pone "COUNT(*)" esta cuenta todas las filas, nulos incluidos, si pones "COUNT(<columna>)" cuenta las celdas de la columna, las cuales no tengan nulos.

"SUM(<columna>)" Suma todos los valores de la columna. Cuidado con nulos, usar coalesce.

"AVG(<columna>)" Media aritmética de los valores de la columna. Cuidado con nulos, usar coalesce.

"MAX(<columna>)" Valor máximo de la columna.

"MIN(<columna>)" Valor mínimo de la columna.

"LIST([DISTINCT] <expresión> [, <delimitador>])" Se usa para listar todos los datos de una lista en una misma celda, por ejemplo: select LIST(nombre, ', ') from propietarios;

## SENTENCIA SELECT

**Resumen de lo que tiene una sentencia Select:**

```
with cte as (select…),
cte2 as (select…)
select [first <num>] [skip <num>] [distinct] (* | <columnas> | <constantes>)
   from <tabla> [<joins>]
   where <condicion>
   group by 1
   having <condicion>
union [distinct|all]
select…
order by <columnas> [asc/desc]
rows <num> to <num>
```

### SELECT

En esta parte se especifica qué mostrar. Se especifica qué mostrar, la cantidad de columnas y cuales (first <num> skip <num>), y si mostrar repetidas o no (distinct). Las columnas a mostrar también se pueden especificar en la última parte de la sentencia (rows <num> to <num>).

## FROM

En el "from" se especifica las tablas de las que seleccionar los datos, CTE incluidas. Se pueden unir varias con las sentencias "join":

- [INNER] JOIN <condicionJoin>: Es la unión más común, se puede poner como "JOIN" o como "INNER JOIN", esta mira todas las celdas que coincidan en la columna de la tabla especificada en el "from" con la especificada en el propio "join" y une los datos de las filas que coincidan.
- CROSS JOIN: Es el producto de las filas de una tabla con las filas de otra. Si en una tabla tenemos la columna "A" con los datos "1, 2" (2 filas), y en otra tenemos columna "B" con los datos "3, 4" (2 filas), haciendo cross join se queda en una tabla con dos columnas (A y B), las filas son: 1,3 - 1,4 - 2,3 - 2,4 (4 filas).
- LEFT/RIGHT/FULL JOIN <condicionJoin>: Este tipo de sentencia es como el INNER JOIN pero con el añadido de que deja las entradas que no coincidan de la tabla que se especifica, la del from o anterior (left), la del propio join (right) o ambas (full). Para estas filas que no puede unir pone a NULL todas las celdas que no tenga datos.
- OUTER JOIN <condicionJoin>: Es como lo contrario al INNER JOIN, deja solamente las filas que no coincidan.

<condicionJoin> → Sentencias "using(<columna>)" y "on <condicion>": Estas sentencias se usan para marcar que columnas usar para hacer las uniones, using es más simple, se usa cuando las dos columnas tienen el mismo nombre, on es más complejo y se usa cuando no tienen el mismo nombre. Ejemplos: "using(id_propietario)", "on t1.nombreFormal = t2.nombre2".

## WHERE

En esta clausula se indican las condiciones a cumplir, comprobaciones lógicas, etc.

Su sintaxis es "WHERE <condición/condiciones de búsqueda>", todas las condiciones las hemos visto anteriormente, véase la parte de "EXPRESIONES Y PREDICADOS" de este resumen. Dato importante, la clausula WHERE afecta a los datos antes de ser agrupados.

## GROUP BY

Los datos modificados por la clausula WHERE se pueden agrupar por diferentes celdas o expresiones relacionadas con las celdas.

Su sintaxis es "GROUP BY <columna/columnas/expresión/numeroDeColumna>" así podemos tener por ejemplo:

- GROUP BY id_inquilino
- GROUP BY id_inquilino, extract(year from fecha_inicio)
- GROUP BY 1, 2 (Siendo 1 y 2 las primeras dos columnas de la cláusula "select")

## HAVING

Después de "GROUP BY" se puede hacer más selecciones con expresiones y funciones. "HAVING" funciona exactamente igual que "WHERE" pero afecta después de agrupar las filas.

Su sintaxis es "HAVING <condición/condiciones de búsqueda>".

### UNION

Una unión de dos (o más) tablas se puede hacer siempre y cuando se tengan las mismas columnas (mismo nombre, misma cantidad y mismo tipo de columnas).

Su sintaxis es "<sentencia select> UNION [ALL] <sentencia select>". Por ejemplo:

```
select nombre from propietarios
UNION
select nombre from inquilinos;
```

UNION por defecto suprime los resultados duplicados, pero si queremos esos resultados también entonces podemos poner "UNION ALL".

### ORDER BY

Esta cláusula se usa para ordenar los resultados finales.

Su sintaxis es "ORDER BY <columna/expresion/grado> [{, <columna/expresion/grado>}]" añadiendo que a cada columna/expresion/grado puedes añadirle un "ASC/DES" para indicar el orden de ordenación y "[NULLS LAST/FIRST]" para indicar donde posicionar los nulos.

Por ejemplo: ORDER BY id_inquilino ASC NULLS LAST, fecha_inicio DES

### CTE

Estas CTE son consultas que vamos a usar a posteriori en la consulta principal.

Su sintaxis es "WITH <nombre consulta> AS (<sentencia select>) [{, (sentencias select)}]". Estas consultas se ponen antes de la principal, y normalmente solamente se suele poder usar estas tablas en la cláusula "FROM" y sus "JOIN" para integrarla.

## SENTENCIAS INSERT, UPDATE, DELETE

Estas se usan junto a "COMMIT;" y "ROLLBACK;" para aceptar o cancelar los cambios.

Para evitar ejecutar estas sentencias sin saber si van a funcionar correctamente o no podemos usar sentencias "select" para previsualizar los cambios que vamos a hacer.

### INSERT

Se usa para insertar una o varias filas a una tabla. Tiene dos tipos de sintaxis válida:

- INSERT INTO <tabla> (<lista de columnas>)
  VALUES (<lista de valores/expresiones>)
- INSERT INTO <tabla> (<lista de columnas>)
  <sentencia select>

Ejemplos:

- INSERT INTO inquilinos (nombre, nif, direccion)
  VALUES ('Fernando', '12345678A', 'C/ Fernando');
- INSERT INTO propietarios (nombre, nif, direccion)
  SELECT nombre, nif, direccion
  FROM inquilinos
  WHERE id_inquilino < 5;

Equivalencia de sentencia SELECT para previsualizar lo que se va a añadir:

- SELECT 'Fernando', '12345678A', 'C/ Fernando' FROM RDB$DATABASE;
- SELECT nombre, nif, direccion
  FROM inquilinos
  WHERE id_inquilino < 5;

## UPDATE

Se usa para "actualizar" los campos de una o varias filas ya existentes de una tabla. Su sintaxis es:

```
UPDATE <tabla> [<alias>]
    SET <columna> = <expresion> [{, <columna> = <expresion>}]
    [WHERE …]
    [ORDER BY …]
    [ROWS <num> [TO <num>]]
```

En el SET indicamos las columnas a cambiar y la expresión a utilizar, esta expresión puede ser simplemente una constante (1, 'hola', false, etc), puede ser una función, o puede ser una sentencia select.

En el WHERE indicamos una función para marcar qué columnas queremos cambiar.

ORDER BY junto a ROWS se usan para marcar cuantas filas se van a modificar, por ejemplo, podemos ordenar por nombre ascendente y modificar solamente las 3 primeras filas.

Un ejemplo y su equivalencia a la sentencia SELECT para previsualizar qué se va a cambiar:

- UPDATE inquilinos i
  SET nombre = 'Ernesto',
  población =
  (select poblacion from inquilinos where id_inquilino = 2)
  WHERE i.id_inquilino = 1;
- SELECT nombre, poblacion, 'Ernesto' nombre_final,
  (select poblacion from inquilinos where id_inquilino = 2) poblacion_final
  FROM inquilinos i
  WHERE i.id_inquilino = 1;

## DELETE

Se usa para borrar una fila o varias filas de una tabla. Su sintaxis es:

DELETE FROM <tabla> [<alias>]
    [WHERE …]
    [ORDER BY …]
    [ROWS <num> [TO <num>]]

En el "WHERE" marcamos las expresiones y funciones para marcar que filas queremos borrar.

ORDER BY y ROWS se puede usar al igual que en la sentencia UPDATE para marcar que filas se quieren borrar.

Un ejemplo y su equivalencia a la sentencia SELECT para previsualizar qué se va a cambiar:

- DELETE FROM reparaciones r
      WHERE extract(year from r.fecha) < 2022;
- SELECT *
      FROM reparaciones r
      WHERE extract(year from r.fecha) < 2022;