ИСПОЛЬЗОВАНИЕ ВЫРАЖЕНИЙ В ЗАПРОСАХ

Выражения могут быть использованы, если информация, помещаемая в таблицы базы данных или выбираемая из базы данных, должна быть подвергнута некоторой обработке.

Операндами в выражениях могут быть имена столбцов, константы, внутренние контекстные переменные и литералы даты, а также другие выражения.

В качестве операций могут использоваться арифметические операции, логические операции, операции сравнения, операция конкатенации, предикаты существования, предикаты отсутствия значения, вызовы функций.

Порядок вычисления выражения определяется приоритетами операций и может быть изменен с помощью круглых скобок.

При построении выражений следует учитывать, что результат вычисления может зависеть от отсутствия значений операндов (`null`-значений). Большинство СУБД, в случае отсутствия значения одного из операндов, полагает результат равным `null`.

Предикаты – это логические выражения. Предикаты позволяют записывать условия, поверка истинности которых влияет на выполнение операторов SQL. Предикат может быть истинным, ложным и неопределенным (в случае, когда хотя бы один из операндов предиката имеет значение `null`). В SQL ложный и неопределенный результат объединяются и трактуются как ложь. Простейшие предикаты строятся с помощью операций сравнения (=, <, >, < >, >=, <=) и логических операций `NOT`, `AND`, `OR`.

Ниже будут рассмотрены только те элементы выражений, допустимые в реализации SQL СУБД Firebird, которые имеют особенности использования по сравнению с традиционными языками программирования.

1.1 Оператор конкатенации

Оператор конкатенации (||) создает новую строку путем соединения двух строк. Операция конкатенации проверяет результирующую строку на допустимую длину. Если длина превышена, никакого усечения строки не происходит и выдается сообщение об ошибке.

Соединяемые стоки могут быть константами или значениями, полученными из таблицы, или значениями, возвращаемыми строковыми функциями.

Пример 1. Сформировать список сотрудников, выдавая номер отдела в формате «dept_no = номер»

```
SELECT  A.FULL_NAME, 'DEPT_NO = '||A.DEPT_NO
FROM EMPLOYEE A
```

Если хотя бы один из операндов не содержит значения (`null`), то и результирующая строка будет `null`.

1.2 Операторы сравнения и предикаты

При использовании операций сравнения на равенство и неравенство следует учитывать, что если хотя бы один из операндов, участвующих в сравнении имеет значение `null`, то операция сравнения вернет значение «не определено». В логических выражениях оператора SQL это значение будет трактоваться как «ложь».

Наряду с традиционными операциями сравнения, в SQL используются дополнительные операции, которые упрощают форму записи сложных условий.

Предикаты `IS NULL` и `IS NOT NULL` проверяют, что объект в левой части операции сравнения не имеет (или имеет) значение. Если значение имеется, то предикат `IS NULL` вернет «ложь», а `IS NOT NULL` вернет «истину». И наоборот.

Пример 2. Выдать информацию об отделах, не имеющих управляющего.

```
SELECT *
FROM DEPARTMENT WHERE MNGR_NO IS NULL;
```

Предикат `IS [NOT] DISTINCT FROM` считает, что два операнда различные (DISTINCT), если они имеют различные значения, или если одно из значений — NULL, а другое нет. Они считаются NOT DISTINCT (равными), если имеют одинаковые значения или оба имеют значение NULL.

Предикат `BETWEEN...AND...` проверяет, что значение попадает в диапазон значений.

Пример 3. Выдать сведения о сотрудниках, принятых на работу в 1992 году.

```
SELECT * FROM EMPLOYEE
    WHERE HIRE_DATE BETWEEN '01.01.1992' AND '31.12.1992';
```

Предикат `CONTAINING` проверяет, содержится ли указанная последовательность символов в строковом представлении операнда (без учета регистра).

Пример 4. Выдать сведения обо всех изменениях оплаты за 93 год.

```
SELECT * FROM SALARY_HISTORY
    WHERE CHANGE_DATE CONTAINING 93;
```

Предикат `STARTING WITH` проверяет, начинается ли строка с указанной последовательности (с учетом регистра).

Пример 5. Найти сотрудников, фамилия которых начинается с «Jo».

```
SELECT LAST_NAME, FIRST_NAME
    FROM EMPLOYEE
    WHERE LAST_NAME STARTING WITH 'Jo'
```

Предикат `LIKE` позволяет создавать чувствительный к регистру шаблон поиска в символьной строке. Он позволяет использовать два шаблонных символа: % - для представления любого количества любых

символов; символ подчеркивания _ для представления одного неопределенного символа.

Пример 6.

```
SELECT FULL_NAME
     FROM EMPLOYEE
     WHERE FIRST_NAME  LIKE '%at%'
     OR FIRST_NAME LIKE  'Mar_'
```

Предикат `IN (…)` проверяет, присутствует ли значение в указанном списке. Список может быть составлен явно из константных значений или быть результатом выполнения подзапроса.

Пример 7. Выбрать сведения о сотрудниках с указанными именами.

```
SELECT * FROM EMPLOYEE
     WHERE FIRST_NAME IN (’Pete’,’Ann’,’Roger’);
```

1.3 Выражение CASE

Выражение CASE позволяет определить значение результирующего столбца зависимым от результата вычисления группы взаимоисключающих условий.

```
CASE {<выражение 1>|<пустое предложение>}
     WHEN {{NULL|<значение 1>}|<предикат поиска>}
          THEN {<результат 1>|NULL}
     WHEN . . .THEN {<результат 2>|NULL}
     [WHEN . . .THEN {<результат n>|NULL}]
     [ELSE {<результат n+1>|NULL}]
END
```

Фраза WHEN . . .THEN является ключевыми словами в каждом предложении, связывающем условие и результат. Требуется, по крайней мере, одно предложение условие/результат. Фраза ELSE предшествует необязательному последнему результату, который возвращается, если не выполняется ни одно предыдущее предложение.

Аргумент ***выражение 1*** является выражением, значение которого определяет результат предложения CASE. Если оно указано, то результат выражения будет последовательно сравниваться со значениями указанными в предложениях WHEN. Если ***выражение 1*** опущено, то в предложении WHEN должны указываться предикаты.

Предложение CASE возвращает единственное значение. Если не выполняется ни одно из условий и не указано предложение ELSE, то возвращаемый результат будет NULL.

Пример 8. В таблице «Заказы» (SALES) указывается состояние, в котором находится заказ (order_status). Выдать список номеров заказов закодировав состояние заказа следующим образом: 0 – new, 1 – open, 2 – shipped, 3 – waiting.

```
SELECT  s.po_number,
     CASE  s.order_status
          WHEN 'new'     THEN 0
          WHEN 'open'    THEN 1
          WHEN 'shipped' THEN 2
          WHEN 'waiting' THEN 3
     END
     FROM sales S;
```

1.4 Функции

Функции могут использоваться при построении выражений.

Функция CAST() позволяет преобразовывать элемент данных одного типа данных в другой тип.

```
CAST(<значение> AS <тип данных>)
```

Функция EXTRACT() позволяет выделять часть информации из полей типа DATE, TIME и TIMESTAMP. Результат возвращается в виде числа.

```
EXTRACT (<часть> FROM <поле>)
```

Выделяемая часть определяется следующими ключевыми словами:

YEAR | MONTH | DAY | HOUR | MINUTE | SECOND | WEEKDAY| YEARDAY

Функция SUBSTRING() возвращает подстроку, начиная с указанной позиции.

```
SUBSTRING (<строка> FROM <начальная позиция> [FOR <длина>])
```

Функция UPPER() преобразует все символы строки в верхний регистр. Противоположная ей функция LOWER() появилась в версии Firebird 2.0.

```
UPPER(<строка>)
LOWER(<строка>)
```

Функции CHAR_LENGTH() и BIT_LENGTH() возвращают длину строки в символах/битах соответственно (появились в версии Firebird 2.0).

```
CHAR_LENGTH(<строка>)
BIT_LENGTH(<строка>)
```

Функция TRIM() удаляет начальные и концевые пробелы в строке.

```
TRIM(<строка>)
```

Функция GEN_ID() вычисляет и возвращает очередное значение указанного генератора. Использование генераторов больше связано с программированием процедур и триггеров.

```
GEN_ID(<генератор>,<шаг>)
```

Агрегатные функции чаще всего используются в комбинации группировкой для вычисления итогов для групп. Агрегатными функциями являются SUM(), MAX(), MIN(), AVG(), COUNT(). При вычислении агрегатных функций значения NULL не учитываются.

Функция COALESCE() позволяет вычислить значение результата с использованием серии выражений. Результат будет равен значению первого выражения в списке, которое окажется непустым.

```
COALESCE (<выражение 1> {,<выражение 2>[,…<выражение n>]})
```

Функция NULLIF() сравнивает значения двух аргументов и возвращает NULL, если они равны, или первый аргумент, если они не равны.

```
NULLIF(<выражение 1>,<выражение 2>)
```

Функция IIF()принимает на вход три параметра. Если первый параметр равен TRUE, функция возвращает значение второго параметра,

иначе возвращает значение третьего параметра (появилась в версии Firebird 2.0)..

```
IIF (<условие>, результат1, результат2)
```

Пример 9. Найти заказчиков, в названии которых есть аббревиатура ‘Ltd’. Анализ таблицы CUSTOMER показывает, что при написании заглавные и малые буквы используются по-разному.

```
SELECT c.CUSTOMER
     FROM CUSTOMER c
     WHERE UPPER(c.CUSTOMER) CONTAINING 'LTD';
```

Пример 10. Выдать список сотрудников, принятых на работу в 1992 году.

```
SELECT *
     FROM EMPLOYEE s
     WHERE EXTRACT(YEAR FROM s.HIRE_DATE)=1992;
```

Пример 11. Добавить в таблицу, полученную в примере 1 после номера отдела. номер внутреннего телефона сотрудника.

```
SELECT  A.FULL_NAME,
'DEPT_NO='||A.DEPT_NO|| COALESCE('PHONE = '||a.PHONE_EXT,' ')
FROM EMPLOYEE A
```

Пример 12. Для каждого сотрудника указать, работает ли он более 20 полных лет или нет.

```
SELECT m.FULL_NAME, m.HIRE_DATE ,
     IIF(EXTRACT(YEAR FROM CURRENT_DATE)-
          EXTRACT(YEAR FROM m.HIRE_DATE)>20,'Более 20','Менее 20')
     FROM EMPLOYEE m;
```

Пример 13. Для каждого сотрудника указать, кто из них работает более 20 лет, более 15 лет и остальные

```
SELECT m.FULL_NAME, m.HIRE_DATE ,
  CASE
     WHEN
```

```
        EXTRACT(YEAR FROM CURRENT_DATE)-
        EXTRACT(YEAR FROM m.HIRE_DATE)>20
    THEN 'Более 20'
    WHEN
        EXTRACT(YEAR FROM CURRENT_DATE)-
        EXTRACT(YEAR FROM m.HIRE_DATE)>15
    THEN 'Более 15'
    ELSE 'Менее 15'
END
    FROM EMPLOYEE m;
```