Procedural Structured Query Language

Курс «Базы данных»

Вадим Цесько

Санкт-Петербургский государственный политехнический университет

29 марта 2012 г.

Содержание

1. Представления
2. SQL-программирование

Представления

Представление

Именованный SELECT-запрос, хранимый в БД:

```
CREATE VIEW <имя_представления> [(<имя_столбца>, ...)]
AS <запрос>
```

Представления

Представление

Именованный SELECT-запрос, хранимый в БД:

```
CREATE VIEW <имя_представления> [(<имя_столбца>, ...)]
AS <запрос>
```

Пример

```
CREATE VIEW student_groups (fname, lname, number) AS
SELECT fname, lname, number
    FROM students, groups
    WHERE students.group_id = groups.id;
```

Особенности представлений

Особенности представлений

- Доступно только для чтения

Особенности представлений

- Доступно только для чтения
- Автоматическое обновление данных

Особенности представлений

- Доступно только для чтения
- Автоматическое обновление данных
- Может использоваться как обычная таблица

Особенности представлений

- Доступно только для чтения
- Автоматическое обновление данных
- Может использоваться как обычная таблица
- Виртуальная таблица (копии данных не создается)

Особенности представлений

- Доступно только для чтения
- Автоматическое обновление данных
- Может использоваться как обычная таблица
- Виртуальная таблица (копии данных не создается)
- Поддерживаются вложенные представления

Вложенные представления

Представление group_size

```
CREATE VIEW group_size(gid, scount) AS
SELECT groups.id AS gid, COUNT(students.id) AS s_count
FROM groups, students
WHERE students.group_id = groups.id
GROUP BY groups.id
ORDER BY s_count DESC;
```

Вложенные представления

Представление group_size

```
CREATE VIEW group_size(gid, scount) AS
SELECT groups.id AS gid, COUNT(students.id) AS s_count
FROM groups, students
WHERE students.group_id = groups.id
GROUP BY groups.id
ORDER BY s_count DESC;
```

Представление bgroup

```
CREATE VIEW bgroup(number) AS
SELECT groups.number AS number
FROM group_size, groups
WHERE groups.id = group_size.gid
ROWS 1;
```

Использование представлений

Использование представлений

- Подготовка данных для клиентского приложения

Использование представлений

- Подготовка данных для клиентского приложения
- Выполнение часто используемых запросов

Использование представлений

- Подготовка данных для клиентского приложения
- Выполнение часто используемых запросов
- Декомпозиция сложных запросов

SQL-программирование

- Процедуры, триггеры, генераторы, ...
- Входные, выходные и внутренние переменные
- Композиция операторов SQL
- Операторы ветвления и перехода
- Операторы цикла
- Операторы прекращения цикла и выхода из процедуры
- Оператор возврата результата выполнения
- Исключения
- Внешние пользовательские функции (User Defined Functions, UDF)

Оператор SUSPEND

Оператор SUSPEND

Возврат результатов выполнения процедуры:

```
SUSPEND;
```

Оператор SUSPEND

Оператор SUSPEND

Возврат результатов выполнения процедуры:

```
SUSPEND;
```

Особенности:

Оператор SUSPEND

Оператор SUSPEND

Возврат результатов выполнения процедуры:

```
SUSPEND;
```

Особенности:

- В процедуре может быть несколько SUSPEND

Оператор SUSPEND

Оператор SUSPEND

Возврат результатов выполнения процедуры:

```
SUSPEND;
```

Особенности:

- В процедуре может быть несколько SUSPEND
- В процедуре может не быть SUSPEND

Оператор SUSPEND

Оператор SUSPEND

Возврат результатов выполнения процедуры:

```
SUSPEND;
```

Особенности:

- В процедуре может быть несколько SUSPEND
- В процедуре может не быть SUSPEND
- Не прерывает процедуру

Оператор SUSPEND

Оператор SUSPEND

Возврат результатов выполнения процедуры:

```
SUSPEND;
```

Особенности:

- В процедуре может быть несколько SUSPEND
- В процедуре может не быть SUSPEND
- Не прерывает процедуру
- Один SUSPEND может срабатывать несколько раз

Оператор ветвления IF

Оператор ветвления IF

```
IF (<условие>) THEN
BEGIN
    ...
END ELSE BEGIN
    ...
END;
```

Оператор ветвления IF

Оператор ветвления IF

```
IF (<условие>) THEN
BEGIN
    ...
END ELSE BEGIN
    ...
END;
```

Пример

```
IF (A > 10) THEN
BEGIN
    A = 0;
END;
```

Оператор FOR

Оператор FOR

```
FOR <выборка> INTO <:переменная1, ...> DO
BEGIN
    ...
END;
```

Выбираемые переменные помещаются в список после INTO.

Оператор FOR

Оператор FOR

```
FOR <выборка> INTO <:переменная1, ...> DO
BEGIN
    ...
END;
```

Выбираемые переменные помещаются в список после INTO.

Пример

```
FOR SELECT fname FROM students WHERE group_id IS NOT NULL
    ORDER BY fname INTO :out_fname DO
BEGIN
    IF (CHAR_LENGTH(out_fname) > 5) THEN
        SUSPEND;
END;
```

Оператор WHILE

Оператор WHILE

```
WHILE (<условие>) DO
BEGIN
    ...
END;
```

Оператор WHILE

Оператор WHILE

```
WHILE (<условие>) DO
BEGIN
    ...
END;
```

Пример

```
WHILE (A < 100) DO
BEGIN
    INSERT INTO students(fname, lname)
    VALUES ('Иван', 'Иванов');
    A = A + 1;
END;
```

Операторы BREAK и EXIT

Оператор BREAK — выход из цикла

```
FOR SELECT age, fname FROM students INTO :var, :name DO
BEGIN
    IF (var > 20) THEN
    BEGIN
        SUSPEND;
        BREAK;
    END;
END;
```

Операторы BREAK и EXIT

Оператор BREAK — выход из цикла

```
FOR SELECT age, fname FROM students INTO :var, :name DO
BEGIN
    IF (var > 20) THEN
    BEGIN
        SUSPEND;
        BREAK;
    END;
END;
```

Оператор EXIT — выход из процедуры

```
IF (var > 10) THEN
    EXIT;
```

Генераторы

Именованный счётчик

```
CREATE GENERATOR <gen_name>;
SET GENERATOR <gen_name> TO <value>;
GEN_ID(<gen_name>, <increment_value>);
```

Генераторы

Именованный счётчик

```
CREATE GENERATOR <gen_name>;
SET GENERATOR <gen_name> TO <value>;
GEN_ID(<gen_name>, <increment_value>);
```

Пример

```
CREATE GENERATOR gen_stud_id;
SET GENERATOR gen_stud_id TO 45;
GenValue = GEN_ID(gen_stud_id, 1);
```

Генераторы

Именованный счётчик

```
CREATE GENERATOR <gen_name>;
SET GENERATOR <gen_name> TO <value>;
GEN_ID(<gen_name>, <increment_value>);
```

Пример

```
CREATE GENERATOR gen_stud_id;
SET GENERATOR gen_stud_id TO 45;
GenValue = GEN_ID(gen_stud_id, 1);
```

Использование генераторов:

Генераторы

Именованный счётчик

```
CREATE GENERATOR <gen_name>;
SET GENERATOR <gen_name> TO <value>;
GEN_ID(<gen_name>, <increment_value>);
```

Пример

```
CREATE GENERATOR gen_stud_id;
SET GENERATOR gen_stud_id TO 45;
GenValue = GEN_ID(gen_stud_id, 1);
```

Использование генераторов:

- В триггерах и процедурах

Генераторы

Именованный счётчик

```
CREATE GENERATOR <gen_name>;
SET GENERATOR <gen_name> TO <value>;
GEN_ID(<gen_name>, <increment_value>);
```

Пример

```
CREATE GENERATOR gen_stud_id;
SET GENERATOR gen_stud_id TO 45;
GenValue = GEN_ID(gen_stud_id, 1);
```

Использование генераторов:

- В триггерах и процедурах
- Для заполнения автоинкрементых полей

Генераторы

Именованный счётчик

```
CREATE GENERATOR <gen_name>;
SET GENERATOR <gen_name> TO <value>;
GEN_ID(<gen_name>, <increment_value>);
```

Пример

```
CREATE GENERATOR gen_stud_id;
SET GENERATOR gen_stud_id TO 45;
GenValue = GEN_ID(gen_stud_id, 1);
```

Использование генераторов:

- В триггерах и процедурах
- Для заполнения автоинкрементных полей
- Для выдачи уникального номера клиентскому приложению

Триггеры

Триггер

Процедура обработки определённых событий на сервере:

```
CREATE TRIGGER <имя_триггера>
FOR <имя_таблицы> <тип_триггера>
POSITION <номер_позиции> AS
BEGIN
    ...
END;
```

Триггеры

Триггер

Процедура обработки определённых событий на сервере:

```
CREATE TRIGGER <имя_триггера>
FOR <имя_таблицы> <тип_триггера>
POSITION <номер_позиции> AS
BEGIN
    ...
END;
```

События и соответствующие типы триггеров:

- INSERT — BEFORE INSERT, AFTER INSERT
- UPDATE — BEFORE UPDATE, AFTER UPDATE
- DELETE — BEFORE DELETE, AFTER DELETE

Пример триггера

Переменные

OLD и NEW — версии строки таблицы, над которой выполняется операция.

Пример

```
CREATE TRIGGER StudTrigger
FOR Students BEFORE INSERT
POSITION 0 AS
BEGIN
    NEW.ID = GEN_ID(gen_stud_id, 1);
END;
```

Особенности триггеров

- Триггер срабатывает для одной таблицы
- Таблица может иметь несколько триггеров
- Изнутри триггера можно обращаться к другим таблицам
- Можно выполнять несколько запросов
- Можно вызывать процедуры

Особенности триггеров

- Триггер срабатывает для одной таблицы
- Таблица может иметь несколько триггеров
- Изнутри триггера можно обращаться к другим таблицам
- Можно выполнять несколько запросов
- Можно вызывать процедуры

Доступ к `OLD` и `NEW` в различных типах триггеров:

	OLD	NEW
BEFORE INSERT	-	RW
AFTER INSERT	-	R
BEFORE UPDATE	R	RW
AFTER UPDATE	R	R
BEFORE DELETE	R	-
AFTER DELETE	R	-

Использование триггеров

Использование триггеров

- Для заполнения полей при `INSERT` и `UPDATE`

Использование триггеров

- Для заполнения полей при `INSERT` и `UPDATE`
- Для контроля целостности

Использование триггеров

- Для заполнения полей при `INSERT` и `UPDATE`
- Для контроля целостности
- В целях отладки

Исключения

Исключение

Используется для обработки ошибок и выхода из процедур:

```
CREATE EXCEPTION <имя_исключения> <сообщение>;

EXCEPTION <имя_исключения>;

WHEN EXCEPTION <имя_исключения> DO
BEGIN
    ...
END;
```

Отлов исключений

Системные исключения:

- Ошибки SQL
- Ошибки Firebird

Отлов исключений

Системные исключения:

- Ошибки SQL
- Ошибки Firebird

Отлов системных исключений

```
WHEN {GDSCODE | SQLCODE} <код_ошибки> DO
BEGIN
    ...
END;
```

Отлов исключений

Системные исключения:

- Ошибки SQL
- Ошибки Firebird

Отлов системных исключений

```
WHEN {GDSCODE | SQLCODE} <код_ошибки> DO
BEGIN
    ...
END;
```

Отлов всех исключений

```
WHEN ANY DO
BEGIN
    ...
END;
```

Хранимые процедуры

Хранимая процедура (ХП)

Позволяет выполнять один или несколько операторов SQL. Может иметь входные и/или выходные параметры:

```
CREATE PROCEDURE <имя_ХП> ([<вх_переменная1>, ...])
RETURNS [<вых_переменная1>, ...] AS
DECLARE VARIABLE [<внутр_переменная>];
BEGIN
    ...
END;
```

Особенности хранимых процедур

- Несколько операторов SQL, разделённых ;
- `SET TERM`
- Процедура может возвращать один или несколько наборов переменных
- Возможен вызов процедур внутри процедур (в т. ч. рекурсия)

Особенности хранимых процедур

- Несколько операторов SQL, разделённых ;
- SET TERM
- Процедура может возвращать один или несколько наборов переменных
- Возможен вызов процедур внутри процедур (в т. ч. рекурсия)

Пример

```
CREATE PROCEDURE group_size (number VARCHAR(10))
RETURNS (scount INTEGER) AS
BEGIN
    SELECT COUNT(*) FROM groups, students
    WHERE students.group_id = groups.id AND
          groups.number = :number
    GROUP BY groups.id INTO :scount;
    SUSPEND;
END;
```

Использование хранимых процедур

- Для реализация бизнес-логики на стороне сервера
- Для реализации бизнес-логики, не описываемой одним запросом
- Вызов из процедур, триггеров и клиентского приложения

Использование хранимых процедур

- Для реализация бизнес-логики на стороне сервера
- Для реализации бизнес-логики, не описываемой одним запросом
- Вызов из процедур, триггеров и клиентского приложения

Вызов процедуры

```
EXECUTE PROCEDURE <имя_процедуры> ([<параметр1, ...>]);
```

Использование хранимых процедур

- Для реализация бизнес-логики на стороне сервера
- Для реализации бизнес-логики, не описываемой одним запросом
- Вызов из процедур, триггеров и клиентского приложения

Вызов процедуры

```
EXECUTE PROCEDURE <имя_процедуры> ([<параметр1, ...>]);
```

Пример

```
CREATE PROCEDURE fill_avg_mark AS
FOR SELECT id FROM students INTO :sid DO
BEGIN
    UPDATE students SET avg_mark = (
        SELECT avg(cast(mark as float))
        FROM stud_results
        WHERE student_id = :sid) WHERE id = :sid;
END;
```

Пример рекурсии (1)

Процедура check_recurse

```
CREATE PROCEDURE check_recurse
RETURNS (recurse INTEGER) AS
DECLARE VARIABLE layer INTEGER;
DECLARE VARIABLE curr_sbj INTEGER;
DECLARE VARIABLE root_sbj INTEGER;
BEGIN
    recurse = 0;
    FOR SELECT subjects.id, ness_subject_id
    FROM subjects, subject_rel
    WHERE subjects.id = subject_rel.subject_id
    INTO :root_sbj, :curr_sbj DO
    BEGIN
        layer = 1;
        EXECUTE PROCEDURE check_subject(root_sbj, curr_sbj, layer);
        WHEN EXCEPTION is_recurse DO
        BEGIN
            recurse = 1;
            BREAK;
        END;
    END;
END;
```

Пример рекурсии (2)

Процедура check_subject

```
CREATE PROCEDURE check_subject(
    root_sbj INTEGER,
    curr_sbj INTEGER,
    layer INTEGER) AS
DECLARE VARIABLE nid INTEGER;
BEGIN
    FOR SELECT ness_subject_id FROM subject_rel
    WHERE subject_id = :curr_sbj
    INTO :nid DO
    BEGIN
        IF ((nid = root_sbj) AND (layer < 10)) THEN
        BEGIN
            EXCEPTION is_recurse;
        END
        layer = layer + 1;
        EXECUTE PROCEDURE check_subject(:root_sbj, :nid, :layer);
    END;
    SUSPEND;
END;
```

Пример возврата множества значений

Возврат множества значений

```
CREATE PROCEDURE mvalues_proc(number VARCHAR(16))
RETURNS (
    id INT,
    fname VARCHAR(32),
    lname varchar(32)) AS
BEGIN
    FOR SELECT id, fname, lname
    FROM students, groups
    WHERE students.group_id = groups.id
    INTO :id, :fname, :lname DO
    BEGIN
        ...
        SUSPEND;
    END;
END;
```

Примеры вызова хранимых процедур

Выбор одного (первого) набора данных

```
EXECUTE PROCEDURE mvalues_proc('4081/1')
RETURNING_VALUES :id, :fname, :lname;
```

Примеры вызова хранимых процедур

Выбор одного (первого) набора данных

```
EXECUTE PROCEDURE mvalues_proc('4081/1')
RETURNING_VALUES :id, :fname, :lname;
```

Выбор множества наборов данных

```
FOR SELECT * FROM mvalues_proc('4081/1')
INTO :fname, :lname DO
BEGIN
    SUSPEND;
END;
```

Примеры вызова хранимых процедур

Выбор одного (первого) набора данных

```
EXECUTE PROCEDURE mvalues_proc('4081/1')
RETURNING_VALUES :id, :fname, :lname;
```

Выбор множества наборов данных

```
FOR SELECT * FROM mvalues_proc('4081/1')
INTO :fname, :lname DO
BEGIN
    SUSPEND;
END;
```

Вызов ХП из запроса

```
SELECT * FROM mvalues_proc('4081/1') WHERE id > 10;
```

События

Событие

Сообщение, которое БД посылает клиентам:

```
POST_EVENT <текст сообщения>;
```

События

Событие

Сообщение, которое БД посылает клиентам:

```
POST_EVENT <текст сообщения>;
```

Примеры

```
POST_EVENT 'It's an event';

DECLARE VARIABLE Message1 VARCHAR(20);
Message1 = 'It's an event';
POST_EVENT :Message1;
```

User Defined Functions

UDF = DLL + SQL

Функция пользователя для расширения возможностей SQL:

```
DECLARE EXTERNAL FUNCTION <имя функции>
    <список типов входных параметров>
RETURNS
    <тип возвращаемого значения>
ENTRY_POINT <имя функции в DLL>
MODULE_NAME <имя DLL>;
```

User Defined Functions

UDF = DLL + SQL

Функция пользователя для расширения возможностей SQL:

```
DECLARE EXTERNAL FUNCTION <имя функции>
    <список типов входных параметров>
RETURNS
    <тип возвращаемого значения>
ENTRY_POINT <имя функции в DLL>
MODULE_NAME <имя DLL>;
```

В поставку Firebird входит набор стандартных функций UDF.

Пример UDF

Определение UDF

```
function strlength(s: PChar): Integer; cdecl; export;
begin
    Result := StrLen(s);
end;
```

Пример UDF

Определение UDF

```
function strlength(s: PChar): Integer; cdecl; export;
begin
    Result := StrLen(s);
end;
```

Объявление UDF

```
DECLARE EXTERNAL FUNCTION strlength CSTRING(100)
RETURNS INTEGER BY VALUE
ENTRY_POINT 'strlength'
MODULE_NAME 'mydll.dll';
```