Процедуры и функции

Реляционная модель

Преподаватель:

канд. тех. наук, доц. Озерова Г.П.

Хранимые процедуры и функции

Хранимая процедура/функция – это объект базы данных, который хранит набор операторов языка SQL, реализующих определённый алгоритм.

Используются для реализации бизнес логики, алгоритмов, расчетов, а также для создания различных инструкций администрирования баз данных и сервера, которые периодически необходимо выполнять.

Другими словами, **хранимые процедуры/функции** – это своего рода программы внутри базы данных, которые хранят реализованный пользователем алгоритм, и в случае запуска этих программ, выполняют этот алгоритм.

Преимущества хранимых процедур и функций

Повышение безопасности. При вызове хранимой процедуры/функции пользователям доступна только внешняя информация (название и необходимые параметры). Имена таблиц и других объектов базы данных не видны, это позволяет скрыть используемые объекты и детали реализации. Таким образом, невозможно внести случайные или неправомерные изменения в алгоритм, выполняемый хранимой процедурой или функцией.

Устранение дублирования кода. SQL код, который многократно используется, можно вынести в хранимую процедуру или функцию, тем самым устранить необходимость копирования одного и того же кода. Как результат, уменьшится общий объем кода, так как в нем не будет повторяющихся фрагментов.

Преимущества хранимых процедур и функций

Легкое сопровождение кода. Если необходимо внести изменение в алгоритм работы с данными, который реализован в хранимой процедуре или функции, достаточно один раз внести изменение в ее код.

Повышение производительности. SQL-сервер компилирует хранимую процедуру и создает план выполнения один раз, а затем повторно использует этот план выполнения. Это приводит к повышению производительности запросов в случае многократного вызова хранимой процедуры.

Хранимые функции

Хранимая функция – это объект базы данных, который хранит набор операторов языка SQL, реализующих определённый алгоритм, и возвращающая одно значение.

Хранимая функция на входе получает список параметров с указанием их типа, выполняет определенную последовательность действий и возвращают значение определенного типа.

Хранимые функции

Синтаксис описания функции следующий:

```
CREATE FUNCTION имя(параметр_1 ТИП, параметр_2 ТИП, ...)
RETURNS тип_результата
BEGIN
  операторы_функции;
  RETURN результат;
END;
```

Определенная таким образом функция возвращает одно значение (результат) и может быть использована в любом выражении SQL запроса, где допустимо значение того типа, который возвращает данная функция.

Хранимые функции

Для обращения к функции используется запись:

имя(фактический_параметр_1, фактический_параметр_2, ...)

При этом должны выполняться следующие правила соответствия между **формальными** (параметрами из заголовка описания функции) и **фактическими параметрами**:

- количество формальных и фактических параметров должно совпадать;
- параметры должны быть совместимых типов (то есть если формальный параметр имеет тип DECIMAL, то фактический параметр может быть либо типа DECIMAL, либо типа INT), но рекомендуется, чтобы типы формальных и фактических параметров совпадали;
- порядок следования формальных параметров, должен совпадать с порядком следования фактических.

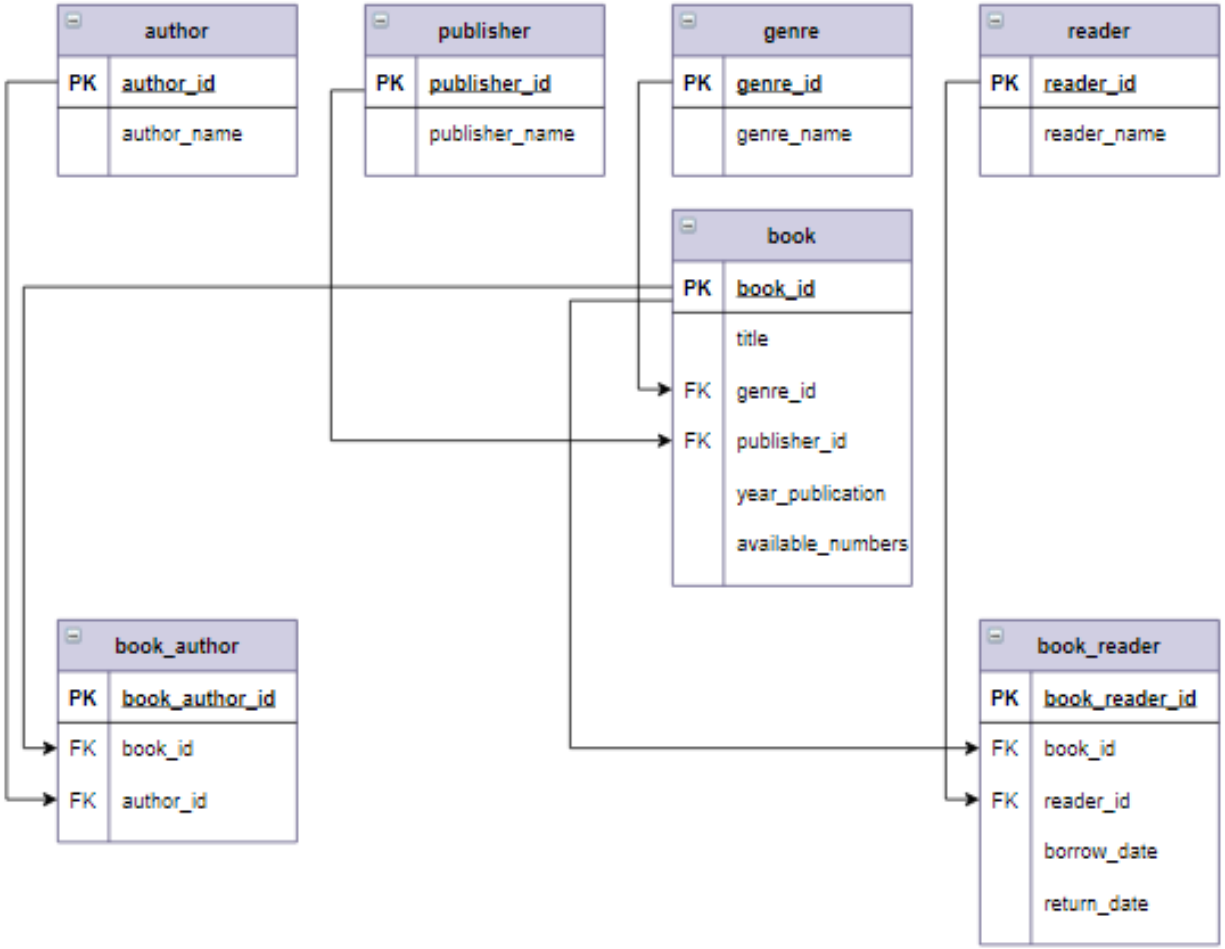
Предметная область

В библиотеке хранятся **книги**. Каждая книга относится к одному **жанру**, опубликована в одном **издательстве**, может иметь одного или несколько **авторов**. Также о книге известна *дата ее публикации*. Библиотека располагает некоторым *количеством экземпляров* каждой книги.

Каждый человек может стать **читателем** в библиотеке. Читатель может взять одну или несколько книг на некоторое время. При этом в библиотеке сохраняется информация о *дате выдачи* книги и *дате ее возврата*. Когда читатель берет книгу, количество доступных экземпляров уменьшается, когда возвращает - увеличивается.

Предметная область

Логическая схема базы данных



Хранимые функции, пример

Пример 1. Вывести те экземпляры книг, которые уже сданы и были на руках больше 14 дней. Указать, сколько именно дней их держали читатели. Информацию отсортировать по убыванию количества дней.

В данном примере в виде **хранимой функции** можно реализовать вычисление количества дней, в которые экземпляры книг были "на руках" у читателей.

В эту функцию будет передано **два параметра**: *дата выдачи* и *дата возврата книги*. **Результатом** станет **целое число** - количество дней между этими датами.

Хранимые функции, пример

Пример 1. Вывести те экземпляры книг, которые уже сданы и были на руках больше 14 дней. Указать, сколько именно дней их держали читатели. Информацию отсортировать по убыванию количества дней.

```
/* Назначение: вычисляет количество дней между двумя датами по формуле:
дата_2 - дата_1 + 1
Входные данные:
    дата_1, тип - DATE
   дата_2, тип - DATE
Результат: количество дней, тип - INT */

CREATE FUNCTION get_count_day(date_beg DATE, date_end DATE)
RETURNS INT
BEGIN
    RETURN DATEDIFF(date_end, date_beg) + 1;
 END;
```

Хранимые функции, пример

Пример 1. Вывести те экземпляры книг, которые уже сданы и были на руках больше 14 дней. Указать, сколько именно дней их держали читатели. Информацию отсортировать по убыванию количества дней.

```
CREATE FUNCTION get_count_day(date_beg DATE, date_end DATE)
RETURNS INT
BEGIN
   RETURN DATEDIFF(date_end, date_beg) + 1;
 END;

SELECT book_id,
   get_count_day(borrow_date, return_date) AS Количество_дней
FROM book_reader
WHERE get_count_day(borrow_date, return_date) > 14
            AND return_date IS NOT NULL
ORDER BY 2 DESC;
```

Хранимые функции, пример

Пример 1. Вывести те экземпляры книг, которые уже сданы и были на руках больше 14 дней. Указать, сколько именно дней их держали читатели. Информацию отсортировать по убыванию количества дней.

```
SELECT book_id,
    get_count_day(borrow_date, return_date) AS Количество_дней
FROM book_reader
WHERE get_count_day(borrow_date, return_date) > 14
            AND return_date IS NOT NULL
ORDER BY 2 DESC;
```

Созданная функция будет вызываться как из раздела SELECT, так и из раздела WHERE SQL-запроса.

Хранимые функции, пример

Пример 1. Вывести те экземпляры книг, которые уже сданы и были на руках больше 14 дней. Указать, сколько именно дней их держали читатели. Информацию отсортировать по убыванию количества дней.

При этом к этой функции будет выполнено обращение для **КАЖДОЙ ЗАПИСИ**, отобранной в запросе.

Хранимые функции, пример

Пример 1. Вывести те экземпляры книг, которые уже сданы и были на руках больше 14 дней. Указать, сколько именно дней их держали читатели. Информацию отсортировать по убыванию количества дней.

FROM book_reader

book_reader_id	book_id	reader_id	borrow_date	return_date
1	4	4	2020-09-11	2020-09-24
2	12	6	2020-09-11	NULL
3	29	5	2020-09-17	2020-10-10
4	27	6	2020-09-18	2020-10-14
5	15	4	2020-09-18	2020-10-04
6	18	1	2020-09-21	2020-10-09
7	22	4	2020-09-25	2020-10-10
		...		
45	21	3	2020-11-29	2020-12-21
46	5	3	2020-11-29	NULL

Хранимые функции, пример

Пример 1. Вывести те экземпляры книг, которые уже сданы и были на руках больше 14 дней. Указать, сколько именно дней их держали читатели. Информацию отсортировать по убыванию количества дней.

```
FROM book_reader
WHERE get_count_day(borrow_date, return_date) > 14
        AND return_date IS NOT NULL
```

book_reader_id	book_id	reader_id	borrow_date	return_date
1	4	4	2020-09-11	2020-09-24
2	12	6	2020-09-11	NULL
3	29	5	2020-09-17	2020-10-10
4	27	6	2020-09-18	2020-10-14
5	15	4	2020-09-18	2020-10-04
6	18	1	2020-09-21	2020-10-09
7	22	4	2020-09-25	2020-10-10
		...		
45	21	3	2020-11-29	2020-12-21
46	5	3	2020-11-29	NULL

2020-09-24 – 2020-09-11 + 1 = 14 - не подходит

- не подходит

2020-10-10 – 2020-09-17 + 1 = 24 - подходит

Хранимые функции, пример

Пример 1. Вывести те экземпляры книг, которые уже сданы и были на руках больше 14 дней. Указать, сколько именно дней их держали читатели. Информацию отсортировать по убыванию количества дней.

```
SELECT book_id,
    get_count_day(borrow_date, return_date) AS Количество_дней
FROM book_reader
WHERE get_count_day(borrow_date, return_date) > 14
            AND return_date IS NOT NULL
```

```
+---------+-----------------+
| book_id | Количество_дней |
+---------+-----------------+
| 29      | 24              |
| 27      | 27              |
| 15      | 17              |
…
+---------+-----------------+
```

Хранимые функции, пример

Пример 1. Вывести те экземпляры книг, которые уже сданы и были на руках больше 14 дней. Указать, сколько именно дней их держали читатели. Информацию отсортировать по убыванию количества дней.

```
SELECT book_id,
    get_count_day(borrow_date, return_date) AS Количество_дней
FROM book_reader
WHERE get_count_day(borrow_date, return_date) > 14
            AND return_date IS NOT NULL
ORDER BY 2 DESC;
```

```
+---------+-----------------+
| book_id | Количество_дней |
+---------+-----------------+
| 20      | 28              |
| 27      | 27              |
| 29      | 24              |
…
+---------+-----------------+
```

Локальные переменные

Локальные переменные – это переменные, которые создаются и используются внутри процедуры или функции.

Локальные переменные необходимо описать перед первым их использованием.

При описании переменной указывается ее имя, тип и значение по умолчанию, которое может отсутствовать.

DECLARE имя_переменной ТИП;

или

DECLARE имя_переменной ТИП DEFAULT значение_по_умолчанию;

Локальные переменные

Локальные переменные – это переменные, которые создаются и используются внутри процедуры или функции.

Локальные переменные необходимо описать перед первым их использованием.

При описании переменной указывается ее имя, тип и значение по умолчанию, которое может отсутствовать.

DECLARE имя_переменной ТИП;

или

DECLARE имя_переменной ТИП DEFAULT значение_по_умолчанию;

Локальные переменные

Если в функции используются несколько переменных одного типа, их можно объявить в одной конструкции DECLARE:

DECLARE имя_переменной_1, имя_переменной_2, ... ТИП;

или

DECLARE имя_переменной_1, имя_переменной_2, ... ТИП DEFAULT значение_по_умолчанию;

Значение по умолчанию будет присвоено всем переменным, перечисленным в конструкции DECLARE.

Оператор присваивания

Синтаксис оператора присваивания:

SET имя_переменной = выражение;

С помощью оператора присваивания вычисляется выражение, а затем его значение заносится в переменную.

При этом выражение и переменная должны быть совместимых типов.

Оператор присваивания, пример

Пример. Для каждой несданной книги посчитать, сколько дней она находится на руках у читателя на сегодняшний день.

```
/* Назначение: вычисляет количество дней между двумя датами по формуле:
текущая_дата - дата + 1
Входные данные:
   дата, тип - DATE
Результат: количество дней, тип - INT */

CREATE FUNCTION get_count_day(date_beg DATE)
RETURNS INT
BEGIN
   DECLARE date_current DATE;
   DECLARE count_day INT;
   SET date_current = NOW();
   SET count_day = DATEDIFF(current_date, date_beg) + 1;
   RETURN count_day;
END;
```

Оператор присваивания, пример

Пример. Для каждой несданной книги посчитать, сколько дней она находится на руках у читателя на сегодняшний день.

```
CREATE FUNCTION get_count_day(date_beg DATE)
RETURNS INT
BEGIN
    DECLARE date_current DATE;
    DECLARE count_day INT;
    SET date_current = NOW();
    SET count_day = DATEDIFF(current_date, date_beg) + 1;
    RETURN count_day;
END;

SELECT get_count_day("2022-08-31");
```

```
+----------------------------+
| get_count_day("2022-08-31") |
+----------------------------+
| 13                         |
+----------------------------+
```

Оператор присваивания, пример

Пример. Для каждой несданной книги посчитать, сколько дней она находится на руках у читателя на сегодняшний день.

```
CREATE FUNCTION get_count_day(date_beg DATE)
RETURNS INT
BEGIN
    DECLARE date_current DATE;
    DECLARE count_day INT;
    SET date_current = NOW();
    SET count_day = DATEDIFF(current_date, date_beg) + 1;
    RETURN count_day;
END;

SELECT book_id, get_count_day(borrow_date) AS Количество
FROM book_reader
WHERE return_date IS NULL;
```

Оператор присваивания, пример

Пример. Для каждой несданной книги посчитать, сколько дней она находится на руках у читателя на сегодняшний день.

Результат:

book_id	Количество
12	732
18	707
3	698
8	694
18	693
…	

Условный оператор

В функциях можно использовать условный оператор следующей структуры:

```
IF условие_if THEN
    запрос_или_оператор_sql_sql_if;
    ...
ELSEIF условие_elseif_1 THEN
    запрос_или_оператор_sql_elif_1;
    ...
ELSEIF условие_elseif_2 THEN
   запрос_или_оператор_sql_elif_2;
   ...
...
ELSE
    запрос_или_оператор_sql_else;
...
END IF;
```

Условный оператор

Оператор **IF** выполняются следующим образом:

- вычисляется **условие_if**, если условие истина, то выполняются действия после **THEN**;
- если условие ложно, вычисляется **условие_elseif_1** после **ELSEIF**, если вычисленное условие истина - выполняются действия после **THEN**;
- аналогично обрабатываются все остальные **ELSEIF**;
- если все условия ложны, выполняются действия после **ELSE**, (при этом разделы **ELSEIF** и **ELSE** являются необязательными).

Условный оператор, пример

Пример. Вывести список всех книг, хранящихся в библиотеке. Указать их название и доступное количество. Названия книг, длиной меньше или равных **n** символов оставить без изменения, а остальные обрезать до **n-3** символа и добавить многоточие ("...") в конце. Столбцы назвать **Книга** и **Количество**. Информацию отсортировать по названию книг в алфавитном порядке.

Условный оператор, пример

Создадим функцию, которая изменяет длину названия:

```
/* Входные данные: название книги, тип - VARCHAR(80)
                   количество знаков, тип - INT
Результат: новое название книги, тип - VARCHAR(80)
*/

CREATE FUNCTION get_book_name(book_name VARCHAR(80), amount_char INT)
RETURNS VARCHAR(80)
BEGIN
    DECLARE new_name VARCHAR(80) DEFAULT book_name;
    IF CHAR_LENGTH(book_name) > amount_char THEN
        SET new_name = CONCAT(LEFT(book_name, amount_char - 3), "...");
    END IF;
    RETURN new_name;
END;
```

Условный оператор, пример

Проверим работу функции:

```
CREATE FUNCTION get_book_name(book_name VARCHAR(80), amount_char INT)
RETURNS VARCHAR(80)
BEGIN
    DECLARE new_name VARCHAR(80) DEFAULT book_name;
    IF CHAR_LENGTH(book_name) > amount_char THEN
        SET new_name = CONCAT(LEFT(book_name, amount_char - 3), "...");
    END IF;
    RETURN new_name;
END;

SELECT get_book_name("Декомпозиция отношений", 10);
SELECT get_book_name("Декомпозиция отношений", 40);
```

```
+-----------------------------------------+
| get_book_name("Декомпозиция отношений", 10) |
+-----------------------------------------+
| Декомпо...                              |
+-----------------------------------------+
```

```
+-----------------------------------------+
| get_book_name("Декомпозиция отношений", 40) |
+-----------------------------------------+
| Декомпозиция отношений                  |
+-----------------------------------------+
```

Условный оператор, пример

Используем функцию в основном запросе:

```
CREATE FUNCTION get_book_name(book_name VARCHAR(80), amount_char INT)
RETURNS VARCHAR(80)
BEGIN
    DECLARE new_name VARCHAR(80) DEFAULT book_name;
    IF CHAR_LENGTH(book_name) > amount_char THEN
        SET new_name = CONCAT(LEFT(book_name, amount_char - 3), "...");
    END IF;
    RETURN new_name;
END;

SELECT get_book_name(title, 20) AS Книга,
      available_numbers AS Количество
FROM book
ORDER BY 1;
```

Условный оператор, пример

Пример. Вывести список всех книг, хранящихся в библиотеке. Указать их название и доступное количество. Названия книг, длиной меньше или равных **n** символов оставить без изменения, а остальные обрезать до **n-3** символа и добавить многоточие ("...") в конце. Столбцы назвать **Книга** и **Количество**. Информацию отсортировать по названию книг в алфавитном порядке.

Результат:

```
+-------------------------+------------+
| Книга                   | Количество |
+-------------------------+------------+
| Бородино                | 0          |
| Вокруг света за 80 ...  | 5          |
| Герой нашего времени    | 2          |
                  …
| Этюд в багровых тон...  | 0          |
+-------------------------+------------+
```

Запросы в функциях

В функциях можно использовать SQL запросы, возвращающих одно значение.

Результаты их выполнения, как правило, сохраняются в локальных переменных, которые после некоторой обработки могут стать результатом функции.

Примерный шаблон использования запроса, возвращающего одно значение, в хранимой функции:

```
/* переменная для хранения результата запроса */
DECLARE имя_переменной ТИП;

/* заносим результат запроса в переменную */
SET переменная = (SELECT выражение
                  FROM ...
                  ...);
```

Запросы в функциях, пример

Пример. Вывести список всех книг, хранящихся в библиотеке. Для каждой книги вывести информацию об ее авторах и жанре в следующем формате:

```
"автор(ы): " строка
"жанр: " строка
```

Запросы в функциях, пример

В данном примере в виде хранимой функции реализуем формирование информации по каждой книге. В эту функцию будет передано **id** книги, а результатом станет строка с информацией о книге.

```
CREATE FUNCTION get_authors_genre(id_book INT)
RETURNS VARCHAR(100)
BEGIN
    DECLARE author_names VARCHAR(100);
    DECLARE genre VARCHAR(30);
   SET author_names = (SELECT GROUP_CONCAT(author_name)
                              FROM author JOIN book_author USING(author_id)
                               WHERE book_id = id_book GROUP BY book_id);
    SET genre = (SELECT genre_name
                      FROM genre JOIN book USING(genre_id)
                      WHERE book_id = id_book);
    RETURN CONCAT("автор(ы): ", author_names, "\nжанр: ", genre);
END;
```

Запросы в функциях, пример

Проверим работу функции:

```
CREATE FUNCTION get_authors_genre(id_book INT)
RETURNS VARCHAR(100)
BEGIN
   DECLARE author_names VARCHAR(100);
   DECLARE genre VARCHAR(30);
  SET author_names = (SELECT GROUP_CONCAT(author_name)
                             FROM author JOIN book_author USING(author_id)
                              WHERE book_id = id_book GROUP BY book_id);
   SET genre = (SELECT genre_name
                    FROM genre JOIN book USING(genre_id)
                    WHERE book_id = id_book);
   RETURN CONCAT("автор(ы): ", author_names, "\nжанр: ", genre);
END;

SELECT get_authors_genre(26);
```

```
+----------------------------------+
| get_authors_genre(26)            |
+----------------------------------+
| автор(ы): Ильф И.А.,Петров Е.П.  |
| жанр: Роман                      |
+----------------------------------+
```

Запросы в функциях, пример

Выведем информацию о книгах:

```
CREATE FUNCTION get_authors_genre(id_book INT)
RETURNS VARCHAR(100)
BEGIN
    DECLARE author_names VARCHAR(100);
    DECLARE genre VARCHAR(30);
   SET author_names = (SELECT GROUP_CONCAT(author_name)
                                FROM author JOIN book_author USING(author_id)
                                 WHERE book_id = id_book GROUP BY book_id);
    SET genre = (SELECT genre_name
                        FROM genre JOIN book USING(genre_id)
                        WHERE book_id = id_book);
    RETURN CONCAT("автор(ы): ", author_names, "\nжанр: ", genre);
END;

SELECT title AS Книга, get_authors_genre(book_id) AS Описание
FROM book;
```

Запросы в функциях, пример

Пример. Вывести список всех книг, хранящихся в библиотеке. Для каждой книги вывести информацию об ее авторах и жанре в следующем формате:

"автор(ы): " строка
"жанр: " строка

Результат:

Книга	Описание
Двенадцать стульев	автор(ы): Ильф И.А.,Петров Е.П. жанр: Роман
Золотой теленок	автор(ы): Ильф И.А.,Петров Е.П. жанр: Роман
Мастер и Маргарита	автор(ы): Булгаков М.А. жанр: Роман
…	

Запросы в функциях, пустое значение

В функциях можно использовать SQL запросы, возвращающие одно значение. Результаты их выполнения, как правило, сохраняются в локальных переменных, которые после некоторой обработки могут стать результатом функции.

В случае если запрос ничего не отбирает, переменная получает значение NULL. Это позволяет предусмотреть какое-то специальное действие для этого случая.

Например, сразу прервать работу функции, вернув из нее сообщение, или выполнить какие-то другие операции.

Запросы в функциях, пустое значение

Примерный шаблон использования запроса, возвращающего пустое значение:

```
/* переменная для хранения результата запроса */
 DECLARE имя_переменной ТИП;

/* заносим результат запроса в переменную */
SET переменная = (SELECT выражение
                  FROM ...
                  ...);

/* если запрос ничего не возвращает */
IF переменная IS NULL THEN
    RETURN "сообщение";
END IF;
...
```

Запросы в функциях, пустое значение

Примерный шаблон использования запроса, возвращающего пустое значение:

```
/* переменная для хранения результата запроса */
 DECLARE имя_переменной ТИП;

/* заносим результат запроса в переменную */
SET переменная = (SELECT выражение
                  FROM ...
                  ...);

/* если запрос ничего не возвращает */
IF переменная IS NULL THEN
    RETURN "сообщение";
END IF;
...
```

В случае если запрос ничего не отберёт, **переменная** получит значение NULL, выполнение функции прекратится, и она вернет **сообщение**, написанное после RETURN.

В остальных случаях - функция продолжит свое выполнение.

Запросы в функциях, пример

Пример. Для каждого читателя вывести все жанры, книги относящиеся к которым, он брал в библиотеке.

В виде функции реализуем создание списка жанров книг, которые брал читатель.

Запросы в функциях, пример

В функцию будет передано **id** читателя, а результатом станет строка с информацией о жанрах его книг.

```
CREATE FUNCTION get_reader_genres(id_reader INT)
RETURNS VARCHAR(100)
BEGIN
   DECLARE genres VARCHAR(100);
   SET genres = (SELECT GROUP_CONCAT(DISTINCT genre_name)
                        FROM genre JOIN book USING(genre_id)
                                   JOIN book_reader USING(book_id)
                        WHERE reader_id = id_reader
                        GROUP BY reader_id);
   IF genres IS NULL THEN
      RETURN "не брал книг";
   END IF;
   RETURN genres;
END;
```

Запросы в функциях, пример

Проверим, как работает функция:

```
CREATE FUNCTION get_reader_genres(id_reader INT)
RETURNS VARCHAR(100)
BEGIN
   DECLARE genres VARCHAR(100);
   SET genres = (SELECT GROUP_CONCAT(DISTINCT genre_name)
                 FROM genre JOIN book USING(genre_id)
                            JOIN book_reader USING(book_id)
                 WHERE reader_id = id_reader
                 GROUP BY reader_id);
   IF genres IS NULL THEN
      RETURN "не брал книг";
   END IF;
   RETURN genres;
END;

SELECT get_reader_genres(1);
```

```
+----------------------+
| get_reader_genres(1) |
+----------------------+
|Детектив,Лирика,Роман |
+----------------------+
```

Запросы в функциях, пример

Проверим, как работает функция:

```
CREATE FUNCTION get_reader_genres(id_reader INT)
RETURNS VARCHAR(100)
BEGIN
   DECLARE genres VARCHAR(100);
   SET genres = (SELECT GROUP_CONCAT(DISTINCT genre_name)
                 FROM genre JOIN book USING(genre_id)
                            JOIN book_reader USING(book_id)
                 WHERE reader_id = id_reader
                 GROUP BY reader_id);
   IF genres IS NULL THEN
      RETURN "не брал книг";
   END IF;
   RETURN genres;
END;

SELECT get_reader_genres(10);
```

```
+----------------------+
| get_reader_genres(10)|
+----------------------+
|не брал книг          |
+----------------------+
```

Запросы в функциях, пример

Выведем информацию по читателям:

```
CREATE FUNCTION get_reader_genres(id_reader INT)
RETURNS VARCHAR(100)
BEGIN
   DECLARE genres VARCHAR(100);
   SET genres = (SELECT GROUP_CONCAT(DISTINCT genre_name)
                        FROM genre JOIN book USING(genre_id)
                                   JOIN book_reader USING(book_id)
                        WHERE reader_id = id_reader
                        GROUP BY reader_id);
   IF genres IS NULL THEN
      RETURN "не брал книг";
   END IF;
   RETURN genres;
END;

SELECT reader_name AS Читатель, get_reader_genres(reader_id) AS Жанры
FROM reader;
```

Запросы в функциях, пример

Пример. Для каждого читателя вывести все жанры, книги относящиеся к которым, он брал в библиотеке.

Результат:

Читатель	Жанры
Иванов М.С.	Детектив,Лирика,Роман
Петров Ф.С.	Детектив,Роман,Фантастика
Федоров П.Р.	Детектив,Приключения,Роман
Абрамова А.А.	Детектив,Лирика,Приключения,Роман
Самарин С.С.	Детектив,Лирика,Приключения,Роман
Туполев И.Д.	Детектив,Лирика,Роман,Фантастика
Баранов П.В.	не брал книг

Просмотр значений локальной

Переменная, которой присваивается результат запроса, может хранить как значение константы любого типа, так и значение NULL.

Это зависит от передаваемых параметров, наличия в запросе группировки, типа условия и других факторов.

Если необходима обработка такой переменной в зависимости от полученного значения, рекомендуется при отладке функции посмотреть значение этой переменной.

Самые простой способ - вернуть ее значение с помощью оператора RETURN.

Просмотр значений локальной пере

Функция возвращает название жанра по его **id**.

```
CREATE FUNCTION test(id_genre INT)
RETURNS CHAR(50)
BEGIN
   DECLARE var_test CHAR(50);
   SET var_test = (SELECT genre_name
                   FROM genre
                   WHERE genre_id = id_genre);
   RETURN var_test;
END;

SELECT test(8); # результат – NULL

SELECT test(1); # результат – “Роман”
```

Использование функций в других функ

В хранимой функции можно обращаться к другой, ранее созданной хранимой функции.

Например, пусть функция **get_test()** возвращает целое число. Ее можно вызывать:

- в операторе присваивания:

```
SET переменная = get_test() * 2;
```

- в запросах, в тех разделах, где допустимо использовать выражения (SELECT, WHERE, ORDER BY и др):

```
SET переменная = (SELECT ..., get_test(),...FROM ...)
```

- в операторе RETURN:

```
RETURN get_test() * 2;
```

Использование функций в других функ

Пример. Вывести список книг, которые брали читатели библиотеки. Для каждой книги указать их авторов. В запрос включить информацию о том, сколько раз экземпляры книги брали читатели.

В данном примере в виде хранимых функций реализуем:

- формирование информации об авторах каждой книги;
- формирование строки - название книги и ее авторы;
- вычисление значения - сколько раз каждую книгу брали читатели.

Использование функций в других функ

Шаг 1. Формирование информации об авторах каждой книги.

```
/* Входные данные: код книги, тип – INT
 Результат: информация об авторах, тип - VARCHAR(700)
 */
CREATE FUNCTION get_book_authors(id_book INT)
RETURNS VARCHAR(70)
BEGIN
   DECLARE author_names VARCHAR(70);
   SET author_names = (SELECT  GROUP_CONCAT(author_name)
                              FROM author JOIN book_author USING(author_id)
                              WHERE book_id = id_book
                              GROUP BY book_id);
   RETURN author_names;
END;
```

Использование функций в других функ

Шаг 1. Формирование информации об авторах каждой книги.

```
/* Входные данные: код книги, тип – INT
 Результат: информация об авторах, тип - VARCHAR(700)
 */
CREATE FUNCTION get_book_authors(id_book INT)
RETURNS VARCHAR(70)
BEGIN
   DECLARE author_names VARCHAR(70);
   SET author_names = (SELECT  GROUP_CONCAT(author_name)
                              FROM author JOIN book_author USING(author_id)
                              WHERE book_id = id_book
                              GROUP BY book_id);
   RETURN author_names;
END;

SELECT get_book_authors(1);
```

```
+-----------------------+
| get_book_authors(1)   |
+-----------------------+
| Ильф И.А.,Петров Е.П. |
 -----------------------+
```

Использование функций в других функ

Шаг 2. Формирование строки - название книги и ее авторы.

```
/* Входные данные: код книги, тип - INT
   Результат: строка, тип - VARCHAR(150) */

CREATE FUNCTION get_authors_book(id_book INT)
RETURNS VARCHAR(150)
BEGIN
    DECLARE authors_all VARCHAR(170);
    DECLARE title_book VARCHAR(150);
    SET authors_all = get_book_authors(id_book);
    SET title_book = (SELECT title
                      FROM book
                      WHERE book_id = id_book);
    RETURN CONCAT(title_book, "(", authors_all, ")");
END;

SELECT get_authors_book(1);
```

```
+-----------------------------------------+
| get_authors_book(1)                     |
+-----------------------------------------+
| Двенадцать стульев(Ильф И.А.,Петров Е.П.) |
+-----------------------------------------+
```

Использование функций в других функ

Шаг 2. Формирование строки - название книги и ее авторы.

```
/* Входные данные: код книги, тип - INT
   Результат: строка, тип - VARCHAR(150) */

CREATE FUNCTION get_authors_book(id_book INT)
RETURNS VARCHAR(150)
BEGIN
   DECLARE title_book VARCHAR(150);

   SET title_book = (SELECT title
                     FROM book
                     WHERE book_id = id_book);

   RETURN CONCAT(title_book, "(", get_book_authors(id_book), ")");
END;

SELECT get_authors_book(1);
```

```
+-----------------------------------------+
| get_authors_book(1)                     |
+-----------------------------------------+
| Двенадцать стульев(Ильф И.А.,Петров Е.П.) |
+-----------------------------------------+
```

Использование функций в других функ

Шаг 3. Вычислим, сколько раз каждую книгу брали читатели.

```
/* Входные данные: код книги, тип - INT
   Результат: число, тип - INT
*/

CREATE FUNCTION get_count_book_borrow(id_book INT)
RETURNS INT
BEGIN
    RETURN (SELECT COUNT(*)
                FROM book_reader
                WHERE book_id = id_book
);
END;

SELECT get_count_book_borrow(1);
```

```
+--------------------------+
| get_count_book_borrow(1) |
+--------------------------+
| 1                        |
+--------------------------+
```

Использование функций в других функ

Шаг 4. Выведем список книг, которые брали читатели библиотеки.

```
SELECT get_authors_book(book_id) AS Книга,
            get_count_book_borrow(book_id) AS Количество
FROM book
WHERE get_count_book_borrow(book_id) > 0
ORDER BY 2 DESC;
```

Использование функций в других функ

Пример. Вывести список книг, которые брали читатели библиотеки. Для каждой книги указать их авторов. В запрос включить информацию о том, сколько раз экземпляры книги брали читатели.

Результат:

Книга	Количество
Пуаро ведет следствие(Агата Кристи)	4
Смерть поэта(Лермонтов М.Ю.)	3
Собачье сердце(Булгаков М.А.)	3
Трудно быть богом(Стругацкий А.Н.,Стругацкий Б.Н.)	3
. . .	
Золотой теленок(Ильф И.А.,Петров Е.П.)	1

Хранимые процедуры

SQL позволяет создавать хранимые процедуры, которые на входе получают список параметров с указанием их типа, выполняют определенную последовательность действий.

Результат - выполнение реализованных в процедуре запросов.

Рекомендуется в виде процедур реализовывать последовательность SQL-запросов, выполняющих некоторый алгоритм действий.

Хранимые процедуры могут включать как запросы на выборку, так и запросы корректировки данных.

Хранимые процедуры

Синтаксис описания хранимой процедуры следующий:

```
CREATE PROCEDURE имя(параметр_1 ТИП, параметр_2 ТИП, ...)
BEGIN
   операторы_процедуры;
END;
```

Хранимые процедуры

Синтаксис описания хранимой процедуры следующий:

```
CREATE PROCEDURE имя(параметр_1 ТИП, параметр_2 ТИП, ...)
BEGIN
   операторы_процедуры;
END;
```

Для вызова процедуры используется запись:

```
CALL имя(фактический_параметр_1, фактический_параметр_2, ...);
```

Между формальными и фактическими параметрами должно быть установлено соответствие.

Хранимые процедуры

Операторы процедуры:

- описание локальных переменных;
- операторы присваивания ;
- условные операторы;
- запросы на выборку;
- запросы корректировки данных (на добавление, обновление, удаление данных, создание, удаление таблиц).

Хранимые процедуры

Если нужно переопределить хранимую процедуру, перед этим ее необходимо удалить с помощью оператора DROP:

```
CREATE PROCEDURE имя(...)
BEGIN
    ...;
END;

DROP PROCEDURE имя;

CREATE PROCEDURE имя(...)
BEGIN
    ...;
END;
```

Предметная область

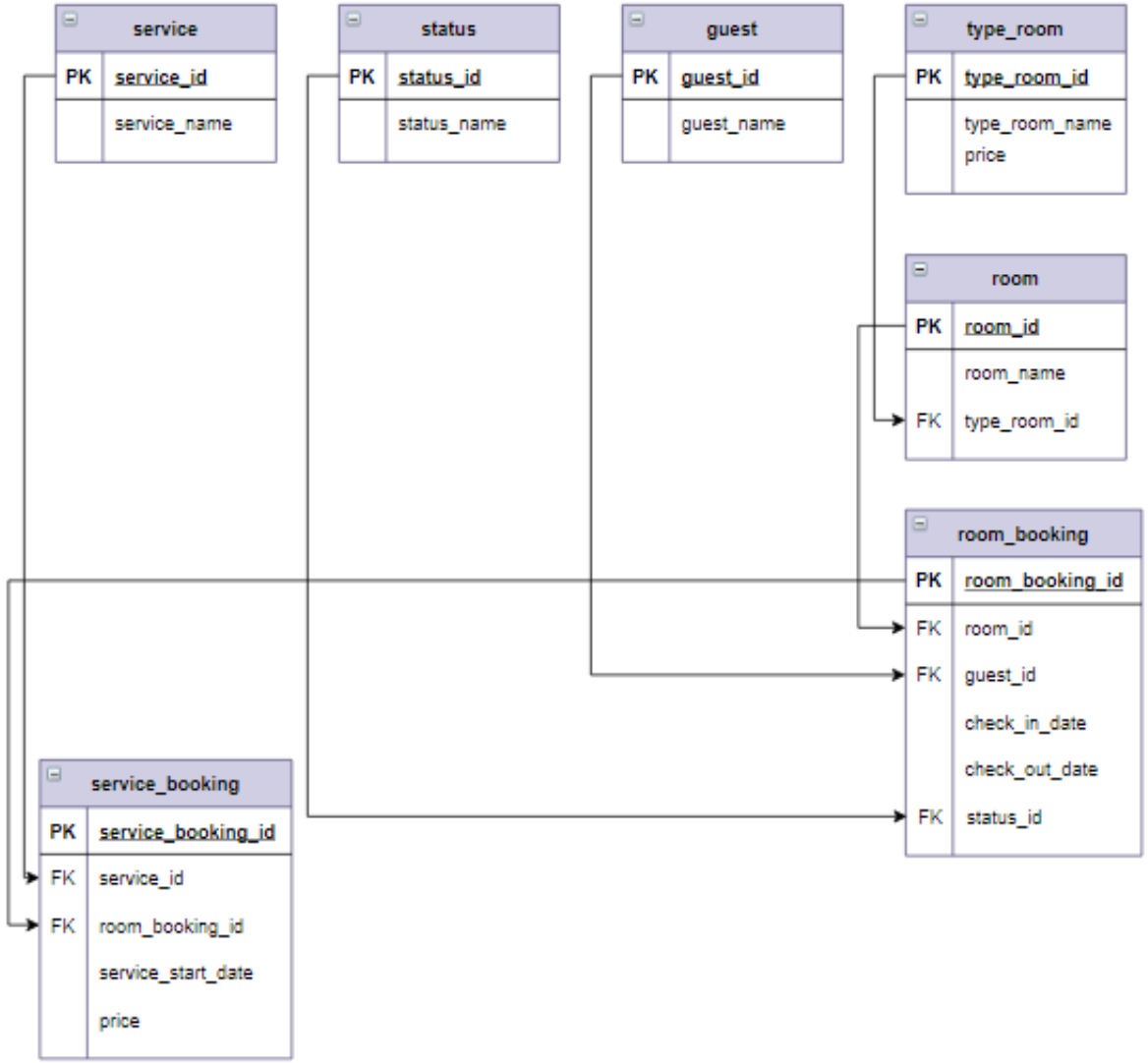
В гостинице есть несколько **номеров**. Каждый номер относится к определенному **типу**. Для каждого типа номера определена цена проживания за день.

Номера в гостинице можно **забронировать**. Для этого **гости** выбирают номер, указывают дату заселения и выселения. Если гость приезжает в гостиницу - **статус** его номера меняется на "*Занят*", если же отменяет бронирование - статус номера становится "*Бронирование отменено*".

Гости, проживающие в гостинице, могут пользоваться дополнительными **платными услугами**. При этом указывается дата получения услуги и сумма за услугу.

Предметная область

Логическая схема базы данных:



Хранимые процедуры, пример

Пример 1. Вывести, сколько раз и какие услуги заказывали гости в заданный период времени.

Если этот запрос используется часто работниками гостиницы, его целесообразно оформить в виде хранимой процедуры. И вызывать ее с разными датами по мере необходимости.

Хранимые процедуры, пример

Описание процедуры:

```
/* Назначение: выводит сколько раз каждую услугу заказывали гости в
заданный интервал времени
Входные данные: дата_1, тип - DATE
                дата_2, тип - DATE*/
CREATE PROCEDURE service_count(date_beg DATE, date_end DATE)
BEGIN
   SELECT service_name AS Услуга, COUNT(*) AS Количество
   FROM service JOIN service_booking USING(service_id)
   WHERE service_start_date BETWEEN date_beg AND date_end
   GROUP BY service_name
   ORDER BY 2 DESC;
END;
```

Хранимые процедуры, пример

Описание процедуры и ее вызов:

```
/* Назначение: выводит сколько раз каждую услугу заказывали гости в
заданный интервал времени
Входные данные: дата_1, тип - DATE
                дата_2, тип - DATE*/
CREATE PROCEDURE service_count(date_beg DATE, date_end DATE)
BEGIN
   SELECT service_name AS Услуга, COUNT(*) AS Количество
   FROM service JOIN service_booking USING(service_id)
   WHERE service_start_date BETWEEN date_beg AND date_end
   GROUP BY service_name
   ORDER BY 2 DESC;
END;

CALL service_count('2021-01-01', '2021-02-01');
```

Хранимые процедуры, пример

Вызов процедуры:

```
CALL service_count('2021-01-01', '2021-02-01');
```

Результат:

```
+---------------------------------+------------+
| Услуга                          | Количество |
+---------------------------------+------------+
| Трансфер от/до аэропорта        | 2          |
| Спа и оздоровительный центр     | 2          |
| Экскурсии                       | 2          |
| Сейф                            | 1          |
| Доставка еды и напитков в номер | 1          |
+---------------------------------+------------+
```

Типы параметров процедуры

Параметры, заданные в заголовке процедуры, могут быть трех типов:

- **входные** - перед ним указывается ключевое слово **IN**, входными считаются параметры, если перед ним ничего не указано;
- **выходные** - перед ними указывается ключевое слово **OUT**;
- **комбинированные** - перед ним указывается ключевое слово **INOUT**.

Входные параметры

Входной параметр используется для передачи значений в процедуру.

Внутри процедуры его можно рассматривать как локальную переменную, то есть его можно изменять с помощью операторов присваивания.

Но вне процедуры - фактический параметр остается неизменным.

Фактический параметр для входного параметра может быть как константой, так и переменной.

Входные параметры

Например:

```
CREATE PROCEDURE test(IN number INT)
BEGIN
    SET number = number * 2;
    SELECT number;
END;
```

Процедура в качестве параметра получает целое число.
В процедуре это число удваивается и выводится с помощью с помощью SELECT.

Входные параметры

```
SET @x = 3;
CALL test(@x);
SELECT @x;
```

Перед вызовом процедуры в переменную **@x** заносится значение 3. Результат вызова процедуры - удвоенное значение **@x**.

```
+--------+
| number |
+--------+
| 6      |
+--------+
```

Но если просмотреть значение **@x** после вызова процедуры - оно не изменится:

```
+----+
| @x |
+----+
| 3  |
+----+
```

Выходные параметры

Выходной параметр используется для возвращения значения из процедуры.

Внутри процедуры ему необходимо присвоить какое-то значение.

Фактический параметр для выходного параметра может быть только переменной.

Его значение можно использовать после выполнения процедуры.

Выходные параметры

Например:

```
CREATE PROCEDURE test(OUT number INT)
BEGIN
    number = ROUND(RAND() * 100);
    SELECT number;
END;
```

Процедура генерирует случайное число от 0 до 100 и присваивает полученное значение выходному параметру. Затем это число выводится с помощью с помощью SELECT.

Выходные параметры

```
CALL test(@x);
SELECT @x;
```

При вызове в процедуру передается фактический параметр (переменная) @**x**.

Результат вызова процедуры - случайное число.

```
+--------+
| number |
+--------+
| 82     |
+--------+
```

Если просмотреть значение **@x** после вызова процедуры - оно будет равно сгенерированному случайному числу.

```
+--------+
| number |
+--------+
| 82     |
+--------+
```

Комбинированные параметры

Комбинированный параметр используется как для передачи значений в процедуру, так и возвращения значения из нее.

Внутри процедуры его можно рассматривать как локальную переменную, то есть его можно изменять с помощью операторов присваивания.

После вызова процедуры - соответствующий фактический параметр сохранит свое значение.

Фактический параметр для комбинированного параметра может быть только переменной.

Комбинированные параметры

Например:

```
CREATE PROCEDURE test(INOUT number INT)
BEGIN
    SET number = number * 2;
    SELECT number;
END;
```

Процедура в качестве параметра получает целое число. Затем это число удваивается и выводится с помощью с помощью SELECT.

Комбинированные параметры

```
SET @x = 3;
CALL test(@x);
SELECT @x;
```

Перед вызовом процедуры в переменную **@x** заносится значение 3. Результат вызова процедуры - удвоенное значение **@x**.

```
+--------+
| number |
+--------+
| 6      |
+--------+
```

Если просмотреть значение **@x** после вызова процедуры - оно сохранит значение 6:

```
+----+
| @x |
+----+
| 6  |
+----+
```

Типы параметров, пример

Пример 1. Создать процедуру, которая выводит, сколько раз и какие услуги заказывали гости в заданный период времени, а также вычисляет общую сумму, полученную за услуги в указанный период.

Типы параметров, пример

Описание процедуры:

```
/* Назначение: выводит сколько раз каждую услугу заказывали гости в заданный
интервал, а также сумму всех услуг
Входные параметры: дата_1, тип - DATE; дата_2, тип - DATE
Выходной параметр: сумма всех услуг, тип – INT
*/
CREATE PROCEDURE service_count(date_beg DATE, date_end DATE,
                                                OUT service_sum DECIMAL(10,2))
BEGIN
   SELECT service_name AS Услуга, COUNT(*) AS Количество
   FROM service JOIN service_booking USING(service_id)
   WHERE service_start_date BETWEEN date_beg AND date_end
   GROUP BY service_name
   ORDER BY 2 DESC;

   SET service_sum = (SELECT SUM(price)
                           FROM service_booking
                           WHERE service_start_date BETWEEN date_beg AND date_end);
END;
```

Типы параметров, пример

Вызов процедуры:

```
CALL service_count('2021-01-01', '2021-01-31', @sum_all);
SELECT @sum_all AS Всего_за_январь;
```

Результат:

```
+---------------------------------+------------+
| Услуга                          | Количество |
+---------------------------------+------------+
| Трансфер от/до аэропорта        | 2          |
| Спа и оздоровительный центр     | 2          |
| Экскурсии                       | 2          |
| Сейф                            | 1          |
| Доставка еды и напитков в номер | 1          |
+---------------------------------+------------+

+-----------------+
| Всего_за_январь |
+-----------------+
| 24533.00        |
+-----------------+
```

Вызов функции из процедуры

В хранимых процедурах можно использовать хранимые функции, которые реализованы до описания текущей процедуры.

Эти функции можно использовать как в конструкциях (присваивание, условный оператор), так и внутри запросов.

Это позволяет разбить решение задачи на отдельные части, что облегчит ее разработку, отладку и тестирование.

Вызов функции из процедуры, пример

Пример. Создать процедуру, которая выводит сводную информацию по гостинице, включающую следующие характеристики:

- **Количество гостей** - учитывать только тех гостей, которые либо бронировали, либо отменяли бронь, либо проживали в гостинице;
- **Сумма за проживание** - вычислить общую сумму, которую заплатили гости за проживание в номерах;
- **Сумма за услуги** - вычислить общую сумму, которую заплатили гости за услуги, предоставляемые гостиницей.

Вычисление каждой характеристики реализуем в виде отдельной функции.

Также в функции посчитаем количество дней проживания.

Вызов функции из процедуры, пример

Шаг 1. Вычислим **Количество гостей**, будем учитывать только тех гостей, которые либо бронировали, либо отменяли бронь, либо проживали в гостинице. Оформим вычисление в виде функции.

```
/* Назначение: вычисляет количество дней проживания в номере
Входные параметры: дата заселения, тип - DATE
                   дата выселения, тип - DATE */

CREATE FUNCTION get_count_day(check_in_date DATE, check_out_date DATE) RETURNS
INT
BEGIN
    RETURN DATEDIFF(check_out_date, check_in_date) + 1;
END;
```

Вызов функции из процедуры, пример

Шаг 1. Вычислим **Количество гостей**, будем учитывать только тех гостей, которые либо бронировали, либо отменяли бронь, либо проживали в гостинице. Оформим вычисление в виде функции.

```
CREATE FUNCTION get_count_guest()
RETURNS INT
BEGIN
    RETURN (SELECT COUNT(DISTINCT guest_id)
                FROM room_booking);
END;
```

Вызов функции из процедуры, пример

Шаг 1. Вычислим **Количество гостей**, будем учитывать только тех гостей, которые либо бронировали, либо отменяли бронь, либо проживали в гостинице. Проверим, как работает функция.

```
CREATE FUNCTION get_count_guest()
RETURNS INT
BEGIN
    RETURN (SELECT COUNT(DISTINCT guest_id)
                FROM room_booking);
END;

SELECT get_count_guest();
```

```
+-------------------+
| get_count_guest() |
+-------------------+
| 19                |
+-------------------+
```

Вызов функции из процедуры, пример

Шаг 2. Вычислим количество дней проживания одного гостя. Оформим вычисление в виде функции.

```
/* Назначение: вычисляет количество дней проживания в номере
Входные параметры: дата заселения, тип - DATE
                   дата выселения, тип - DATE */

CREATE FUNCTION get_count_day(check_in_date DATE, check_out_date DATE)
RETURNS INT
BEGIN
    RETURN DATEDIFF(check_out_date, check_in_date) + 1;
END;
```

Вызов функции из процедуры, пример

Шаг 2. Вычислим количество дней проживания одного гостя. Проверим работоспособность функции.

```
/* Назначение: вычисляет количество дней проживания в номере
Входные параметры: дата заселения, тип - DATE
                   дата выселения, тип - DATE */

CREATE FUNCTION get_count_day(check_in_date DATE, check_out_date DATE)
RETURNS INT
BEGIN
    RETURN DATEDIFF(check_out_date, check_in_date) + 1;
END;

SELECT get_count_day("2020-02-12", "2020-03-05")
```

```
+-----------------------------------------+
| get_count_day("2020-02-12", "2020-03-05") |
+-----------------------------------------+
| 23                                      |
+-----------------------------------------+
```

Вызов функции из процедуры, пример

Шаг 3. Вычислим общую сумму, которую заплатили гости за проживание в номерах. Оформим вычисление в виде функции.

```
CREATE FUNCTION get_sum_room()
RETURNS INT
BEGIN
  RETURN (SELECT SUM(get_count_day(check_in_date,check_out_date) * price)
               FROM room_booking JOIN room USING (room_id)
                                 JOIN type_room USING (type_room_id)
                                 JOIN status USING (status_id)
              WHERE status_id = 1);
END;
```

Вызов функции из процедуры, пример

Шаг 3. Вычислим общую сумму, которую заплатили гости за проживание в номерах. Проверим, как работает функция.

```
CREATE FUNCTION get_sum_room()
RETURNS INT
BEGIN
  RETURN (SELECT SUM(get_count_day(check_in_date,check_out_date) * price)
               FROM room_booking JOIN room USING (room_id)
                                 JOIN type_room USING (type_room_id)
                                 JOIN status USING (status_id)
              WHERE status_id = 1);
END;

SELECT get_sum_room();
```

```
+----------------+
| get_sum_room() |
+----------------+
| 3726300        |
+----------------+
```

Вызов функции из процедуры, пример

Шаг 4. Вычислим общую сумму, которую заплатили гости за услуги, предоставляемые гостиницей. Оформим вычисление в виде функции.

```
CREATE FUNCTION get_sum_service()
RETURNS INT
BEGIN
    RETURN (SELECT SUM(price)
                FROM service_booking);
END;
```

Вызов функции из процедуры, пример

Шаг 4. Вычислим общую сумму, которую заплатили гости за услуги, предоставляемые гостиницей. Проверим, как работает функция.

```
CREATE FUNCTION get_sum_service()
RETURNS INT
BEGIN
    RETURN (SELECT SUM(price)
                FROM service_booking);
END;


SELECT get_sum_service();
```

```
+-------------------+
| get_sum_service() |
+-------------------+
| 119873            |
+-------------------+
```

Вызов функции из процедуры, пример

Шаг 5. Создадим процедуру, которая выводит информацию о работе гостиницы.

```
CREATE PROCEDURE result()
BEGIN
    SELECT "Количество гостей" AS Характеристика, get_count_guest() AS Результат
    UNION
    SELECT "Сумма за проживание", get_sum_room()
    UNION
    SELECT "Сумма за услуги" , get_sum_service();
END;
```

Вызов функции из процедуры, пример

Шаг 5. Создадим процедуру, которая выводит информацию о работе гостиницы. Вызовем функцию.

```
CREATE PROCEDURE result()
BEGIN
   SELECT "Количество гостей" AS Характеристика, get_count_guest() AS Результат
   UNION
   SELECT "Сумма за проживание", get_sum_room()
   UNION
   SELECT "Сумма за услуги" , get_sum_service();
END;

CALL result();
```

```
+--------------------+-----------+
| Характеристика      | Результат |
+--------------------+-----------+
| Количество гостей   | 19        |
| Сумма за проживание | 3726300   |
| Сумма за услуги     | 119873    |
+--------------------+-----------+
```

Несколько запросов в процедуре

Хранимые процедуры можно использовать для выполнения нескольких запросов (запросов на выборку или корректировку данных).

Это позволяет реализовать некоторый алгоритм, который часто выполняется с информацией из базы данных.

Синтаксис такой процедуры:

```
CREATE PROCEDURE имя()
BEGIN
    ...
   запрос_1;
   ...
запрос_N;
...
END;
```

Несколько запросов в процедуре, пример

Пример. Создать процедуру, которая отменяет бронирование номера гостем с указанной даты и удаляет все заказанные им услуги на время бронирования номер.

```
/* Назначение: отменяет бронирование гостя
   Входные параметры:
        id гостя, тип - INT
        name_room, тип - VARCHAR(10)
        дата предполагаемого заселения, тип - DATE
*/
```

Несколько запросов в процедуре, пример

```
CREATE PROCEDURE cancel_reservation (id_guest INT,
                                    name_room VARCHAR(10), check_in DATE)
BEGIN
     /* меняем статус номера */
     UPDATE room, room_booking
     SET status_id = (SELECT status_id
                           FROM status
                           WHERE status_name = "Бронирование отменено")
      WHERE guest_id = id_guest and room_name = name_room and
                  check_in_date = check_in;

   /* удаляем услуги*/
      DELETE FROM service_booking
      USING room
                JOIN room_booking USING(room_id)
               JOIN service_booking USING(room_booking_id)
       WHERE guest_id = id_guest and room_name = name_room and
                  check_in_date = check_in;
END;
```

Несколько запросов в процедуре, пример

Пример. Создать процедуру, которая отменяет бронирование номера гостем с указанной даты и удаляет все заказанные им услуги на время бронирования номер.

Вызов процедуры:

```
CALL cancel_reservation(11, "C-0219", "2021-05-04");
```

Результат выполнения:
изменения в таблицах room_booking и serice_booking.

Спасибо за внимание!