PostgresPro

Как стать разработчиком ядра PostgreSQL

Первые шаги

Лекция 2. Регистровые типы данных

Николай Шаплов

Август 2025

О докладчике

PostgresPro

Шаплов Николай

Фаззинг инженер в компании
Postgres Professional

- C
- C++
- Perl

Последовательный сторонник свободного ПО
в течении более чем 20 лет

/Общие слова/

PostgresPro

- PostgreSQL — Свободное ПО, код которой можно и нужно читать
- В лекции пойдет речь о базовых структурах PostgreSQL, а именно типах данных помещающихся в регситр.
- Вторая лекция из серии «Как стать разработчиком ядра PostgeSQL»

  [qr-код на первую лекцию]
- Считаем что слушатели умеют создавать расширения

Как пользоваться материалом

PostgresPro

1. Целью данных лекций является не только знакомство со структурами PostgreSQL которые будут продемонстрированы на слайдах, не менее важной целью является формирование навыка работы с исходным кодом СУБД. Поэтому крайне рекомендуется все приведенные фрагменты кода PostgreSQL самостоятельно найти в исходных текстах, и изучить сопутствующие ему фрагменты.

2. Все приведенные примеры следует самостоятельно вкомпилировать в расширение, запустить и «поиграть» с кодом. (Не поиграли, значит пошло мимо)

3. Исходные тексты примеров (для справки) выложены на gitlab вместе с прещентацией. (Ссылка на следующем слайде)

4. Задачи со звездочками предназначены для самостоятельного выполнения для получения более глубокого понимания материала

Ссылка на слайды

PostgresPro

https://gitlab.com/dhyannataraj/articles/-/tree/master/slides

Регистровые типы данных

PostgresPro

Есть типы данных значения которых полностью помещается в регистре процессора:

- целые числа (`char`, `int`, `long int`);
- числа с плавающей точкой (`real`, `double`);
- Дата-время (`time_t`)

Их можно назвать «регистровые типы данных»

Регистровые типы данных

PostgresPro

Исторически в коде PostgreSQL используются свой набор имен для регистровых типов данных. Например для целых чисел

```
/* Historical names for types in <stdint.h>. */
typedef int8_t int8;
typedef int16_t int16;
typedef int32_t int32;
typedef int64_t int64;
typedef uint8_t uint8;
typedef uint16_t uint16;
typedef uint32_t uint32;
typedef uint64_t uint64;
```

src/include/c.h

- Когда писали Postgres95 стандарта C99 с `int32_t` еще не было
- И нельзя точно знать сколько байт займет `signed int` из ANSI C

PostgresPro

С-программист, помни!
Макрос – твой лучший друг!!

Волшебные макросы PGXS

PostgresPro

Создание расширения начинается с вызова макроса

```
PG_MODULE_MAGIC;
```

Но магия на этом только начинается.

Этот макрос создает функцию проверяющую совместимость расширения с ядром PostgreSQL, на этапе загрузки расширения. При создании расширения этот макрос следует указывать только один раз.

(****) Текст макроса можно почитать в `src/include/fmgr.h`, но это для сильных духом.

Аргументы pg-функции

PostgresPro

Вспомним пример C-функции из прошлой лекции

```
PG_FUNCTION_INFO_V1(my_get_two);

Datum
my_get_two(PG_FUNCTION_ARGS)
{
    PG_RETURN_INT32(2);
}
```

И рассмотрим ее макросы. Начав с макроса аргументов pg-функции `PG_FUNCTION_ARGS`

Аргументы pg-функции

PostgresPro

Макрос `PG_FUNCTION_ARGS` разворачивается в следующий код

```
/* Standard parameter list for fmgr-compatible functions */
#define PG_FUNCTION_ARGS    FunctionCallInfo fcinfo
```

src/include/fmgr.h

Структура типа `FunctionCallInfo` содержит информацию о всех передаваемых в функцию аргументах.

(**) Задача: посмотреть по коду как именно передаются аргументы внутри этой структуры

С этой структурой не работают напрямую, а так же посредством специального набора макросов.

Пример использования аргумента

PostgresPro

```c
#include "postgres.h"
#include "fmgr.h"

PG_FUNCTION_INFO_V1(warn_me);

Datum
warn_me(PG_FUNCTION_ARGS)
{
    int32 n = PG_GETARG_INT32(0);
    elog(WARNING, "Предупреждаю: %i", n);
    PG_RETURN_INT32(n);
}
```

Пример использования аргумента

PostgresPro

Добавляем в схему:

```
CREATE FUNCTION warn_me(int) RETURNS int
AS 'MODULE_PATHNAME', 'warn_me'
LANGUAGE C;
```

Проверяем в консоли:

```
# select warn_me(12);
WARNING:  Предупреждаю: 12
 warn_me
---------
      12
(1 row)
```

Пример использования аргумента

PostgresPro

```c
#include "postgres.h"
#include "fmgr.h"

PG_FUNCTION_INFO_V1(warn_me);

Datum
warn_me(PG_FUNCTION_ARGS)
{
    int32 n = PG_GETARG_INT32(0);
    elog(WARNING, "Предупреждаю: %i", n);
    PG_RETURN_INT32(n);
}
```

Пример использования аргумента

PostgresPro

Смотрим на реализацию `PG_GETARG_INT32`

```
/* Macros for fetching arguments of standard types */

#define PG_GETARG_DATUM(n)      (fcinfo->args[n].value)
#define PG_GETARG_INT32(n)      DatumGetInt32(PG_GETARG_DATUM(n))
#define PG_GETARG_UINT32(n)     DatumGetUInt32(PG_GETARG_DATUM(n))
#define PG_GETARG_INT16(n)      DatumGetInt16(PG_GETARG_DATUM(n))
#define PG_GETARG_UINT16(n)     DatumGetUInt16(PG_GETARG_DATUM(n))
#define PG_GETARG_CHAR(n)       DatumGetChar(PG_GETARG_DATUM(n))
#define PG_GETARG_BOOL(n)       DatumGetBool(PG_GETARG_DATUM(n))
```

src/include/fmgr.h

Все сводится к `fcinfo->args[n].value` имеющему тип `Datum`

`fcinfo` — содержит информацию об рагуметнтах

`DatumGetInt32` видимо функция преобразования

(*) убедитесь, что `fcinfo->args[n].value` имеющему тип `Datum`

Пример использования аргумента

PostgresPro

Смотрим на реализацию `PG_GETARG_INT32`

```
/* Macros for fetching arguments of standard types */

#define PG_GETARG_DATUM(n)      (fcinfo->args[n].value)
#define PG_GETARG_INT32(n)      DatumGetInt32(PG_GETARG_DATUM(n))
#define PG_GETARG_UINT32(n)  DatumGetUInt32(PG_GETARG_DATUM(n))
#define PG_GETARG_INT16(n)      DatumGetInt16(PG_GETARG_DATUM(n))
#define PG_GETARG_UINT16(n)  DatumGetUInt16(PG_GETARG_DATUM(n))
#define PG_GETARG_CHAR(n)       DatumGetChar(PG_GETARG_DATUM(n))
#define PG_GETARG_BOOL(n)       DatumGetBool(PG_GETARG_DATUM(n))
```

src/include/fmgr.h

Все сводится к `fcinfo->args[n].value` имеющему тип `Datum`

`fcinfo` — содержит информацию об рагуметнтах

`DatumGetInt32` видимо функция преобразования

(*) убедитесь, что `fcinfo->args[n].value` имеющему тип `Datum`

Пример использования аргумента

PostgresPro

Смотрим на реализацию `PG_GETARG_INT32`

```
/* Macros for fetching arguments of standard types */

#define PG_GETARG_DATUM(n)       (fcinfo->args[n].value)
#define PG_GETARG_INT32(n)       DatumGetInt32(PG_GETARG_DATUM(n))
#define PG_GETARG_UINT32(n)  DatumGetUInt32(PG_GETARG_DATUM(n))
#define PG_GETARG_INT16(n)       DatumGetInt16(PG_GETARG_DATUM(n))
#define PG_GETARG_UINT16(n)  DatumGetUInt16(PG_GETARG_DATUM(n))
#define PG_GETARG_CHAR(n)        DatumGetChar(PG_GETARG_DATUM(n))
#define PG_GETARG_BOOL(n)        DatumGetBool(PG_GETARG_DATUM(n))
```

src/include/fmgr.h

Все сводится к `fcinfo->args[n].value` имеющему тип `Datum`

`fcinfo` — содержит информацию об рагуметнтах

`DatumGetInt32` видимо функция преобразования

(*) убедитесь, что `fcinfo->args[n].value` имеющему тип `Datum`

Пример использования аргумента

PostgresPro

Смотрим на реализацию `PG_GETARG_INT32`

```
/* Macros for fetching arguments of standard types */

#define PG_GETARG_DATUM(n)      (fcinfo->args[n].value)
#define PG_GETARG_INT32(n)      DatumGetInt32(PG_GETARG_DATUM(n))
#define PG_GETARG_UINT32(n)  DatumGetUInt32(PG_GETARG_DATUM(n))
#define PG_GETARG_INT16(n)      DatumGetInt16(PG_GETARG_DATUM(n))
#define PG_GETARG_UINT16(n)  DatumGetUInt16(PG_GETARG_DATUM(n))
#define PG_GETARG_CHAR(n)       DatumGetChar(PG_GETARG_DATUM(n))
#define PG_GETARG_BOOL(n)       DatumGetBool(PG_GETARG_DATUM(n))
```

src/include/fmgr.h

Все сводится к `fcinfo->args[n].value` имеющему тип `Datum`

`fcinfo` — содержит информацию об рагуметнтах

`DatumGetInt32` видимо функция преобразования

(*) убедитесь, что `fcinfo->args[n].value` имеющему тип `Datum`

Пример использования аргумента

PostgresPro

Смотрим на реализацию `PG_GETARG_INT32`

```
/* Macros for fetching arguments of standard types */

#define PG_GETARG_DATUM(n)      (fcinfo->args[n].value)
#define PG_GETARG_INT32(n)      DatumGetInt32(PG_GETARG_DATUM(n))
#define PG_GETARG_UINT32(n)  DatumGetUInt32(PG_GETARG_DATUM(n))
#define PG_GETARG_INT16(n)      DatumGetInt16(PG_GETARG_DATUM(n))
#define PG_GETARG_UINT16(n)  DatumGetUInt16(PG_GETARG_DATUM(n))
#define PG_GETARG_CHAR(n)       DatumGetChar(PG_GETARG_DATUM(n))
#define PG_GETARG_BOOL(n)       DatumGetBool(PG_GETARG_DATUM(n))
```

src/include/fmgr.h

Все сводится к `fcinfo->args[n].value` имеющему тип `Datum`

`fcinfo` — содержит информацию об рагуметнтах

`DatumGetInt32` видимо функция преобразования

(*) убедитесь, что `fcinfo->args[n].value` имеющему тип `Datum`

Пример возвращаемого значения

PostgresPro

```c
#include "postgres.h"
#include "fmgr.h"

PG_FUNCTION_INFO_V1(warn_me);

Datum
warn_me(PG_FUNCTION_ARGS)
{
    int32 n = PG_GETARG_INT32(0);
    elog(WARNING, "Предупреждаю: %i", n);
    PG_RETURN_INT32(n);
}
```

Так же в нашем примере есть макрос `PG_RETURN_INT32` для которого разворачивается похожая история

Пример возвращаемого значения

PostgresPro

```
/* Macros for returning results of standard types */

#define PG_RETURN_DATUM(x)       return (x)
#define PG_RETURN_INT32(x)       return Int32GetDatum(x)
#define PG_RETURN_UINT32(x)  return UInt32GetDatum(x)
#define PG_RETURN_INT16(x)       return Int16GetDatum(x)
#define PG_RETURN_UINT16(x)  return UInt16GetDatum(x)
#define PG_RETURN_CHAR(x)        return CharGetDatum(x)
#define PG_RETURN_BOOL(x)        return BoolGetDatum(x)
```

src/include/fmgr.h

Макрос `PG_RETURN_INT32` возвращает свой аргумент перед этим (предположительно) преобразовав его к типу `Datum`

Пример возвращаемого значения

PostgresPro

```
/* Macros for returning results of standard types */

#define PG_RETURN_DATUM(x)        return (x)
#define PG_RETURN_INT32(x)        return Int32GetDatum(x)
#define PG_RETURN_UINT32(x)  return UInt32GetDatum(x)
#define PG_RETURN_INT16(x)        return Int16GetDatum(x)
#define PG_RETURN_UINT16(x)  return UInt16GetDatum(x)
#define PG_RETURN_CHAR(x)         return CharGetDatum(x)
#define PG_RETURN_BOOL(x)         return BoolGetDatum(x)
```

src/include/fmgr.h

Макрос `PG_RETURN_INT32` возвращает свой аргумент перед этим (предположительно) преобразовав его к типу `Datum`

`Datum` великий и ужасный

PostgresPro

```
/*
 * A Datum contains either a value of a pass-by-value type or a pointer to a
 * value of a pass-by-reference type.  Therefore, we require:
 *
 * sizeof(Datum) == sizeof(void *) == 4 or 8
 *
 */
```

```
typedef uintptr_t Datum;
```

src/include/postgres.h

Переменная типа `Datum` это контейнер который либо кранит значение (если оно влезает) либо указатель на область памяти в которой значение хранится (если значение большое)

`Datum` великий и ужасный

PostgresPro

```
/*
 * A Datum contains either a value of a pass-by-value type or a pointer to a
 * value of a pass-by-reference type.  Therefore, we require:
 *
 * sizeof(Datum) == sizeof(void *) == 4 or 8
 *
 */
```

```
typedef uintptr_t Datum;
```

src/include/postgres.h

Переменная типа `Datum` это контейнер который либо кранит значение (если оно влезает) либо указатель на область памяти в которой значение хранится (если значение большое)

`Datum` великий и ужасный

PostgresPro

Если мы заглянем в документацию на `uintptr_t` то там будет написано:

```
$ man uintptr_t

`uintptr_t` is an unsigned integer type such that any valid (void *)
value can be converted to this type and then converted back.
```

`uintptr_t` — целочисленный тип данных, такой что любой валидный указатель может быть приведен к этому типу данных, и наоборот.

Т.е. по сути это целый тип данных той же разрядности, что и указатели в данной архитектуре.

`Datum` и регистровые типы данных

PostgresPro

Функции приведения регистровых типов данных к Datum и обратно

```
/*
 * DatumGetInt32
 *              Returns 32-bit integer value of a datum.
 */
static inline int32
DatumGetInt32(Datum X)
{
        return (int32) X;
}
```

src/include/postgres.h

`Datum` и регистровые типы данных

PostgresPro

Функции приведения регистровых типов данных к Datum и обратно

```
/*
 * Int32GetDatum
 *              Returns datum representation for a 32-bit integer.
 */
static inline Datum
Int32GetDatum(int32 X)
{
        return (Datum) X;
}
```

src/include/postgres.h

`Datum` и регистровые типы данных

PostgresPro

Функции приведения регистровых типов данных к Datum и обратно

```
/*
 * Int32GetDatum
 *              Returns datum representation for a 32-bit integer.
 */
static inline Datum
Int32GetDatum(int32 X)
{
        return (Datum) X;
}
```

src/include/postgres.h

(**) Почему для приведения используются inline-функции, а не макросы?

Пример использования `Datum`

PostgresPro

```c
#include "postgres.h"
#include "fmgr.h"

PG_FUNCTION_INFO_V1(my_factorial);

Datum
my_factorial(PG_FUNCTION_ARGS)
{
    int32 res, n = PG_GETARG_INT32(0);
    Datum d;

    if (n < 0)
        elog(ERROR, "Так нельзя!");
    if (n == 0)
        PG_RETURN_INT32(1);

    d = DirectFunctionCall1(my_factorial, Int32GetDatum(n - 1));
    res = DatumGetInt32(d) * n;
    PG_RETURN_INT32(res);
}
```

Пример использования `Datum`

PostgresPro

```c
#include "postgres.h"
#include "fmgr.h"

PG_FUNCTION_INFO_V1(my_factorial);

Datum
my_factorial(PG_FUNCTION_ARGS)
{
    int32 res, n = PG_GETARG_INT32(0);
    Datum d;

    if (n < 0)
        elog(ERROR, "Так нельзя!");
    if (n == 0)
        PG_RETURN_INT32(1);

    d = DirectFunctionCall1(my_factorial, Int32GetDatum(n - 1));
    res = DatumGetInt32(d) * n;
    PG_RETURN_INT32(res);
}
```

Пример использования `Datum`

PostgresPro

```c
#include "postgres.h"
#include "fmgr.h"

PG_FUNCTION_INFO_V1(my_factorial);

Datum
my_factorial(PG_FUNCTION_ARGS)
{
    int32 res, n = PG_GETARG_INT32(0);
    Datum d;

    if (n < 0)
        elog(ERROR, "Так нельзя!");
    if (n == 0)
        PG_RETURN_INT32(1);

    d = DirectFunctionCall1(my_factorial, Int32GetDatum(n - 1));
    res = DatumGetInt32(d) * n;
    PG_RETURN_INT32(res);
}
```

Пример использования `Datum`

PostgresPro

```c
#include "postgres.h"
#include "fmgr.h"

PG_FUNCTION_INFO_V1(my_factorial);

Datum
my_factorial(PG_FUNCTION_ARGS)
{
    int32 res, n = PG_GETARG_INT32(0);
    Datum d;

    if (n < 0)
        elog(ERROR, "Так нельзя!");
    if (n == 0)
        PG_RETURN_INT32(1);

    d = DirectFunctionCall1(my_factorial, Int32GetDatum(n - 1));
    res = DatumGetInt32(d) * n;
    PG_RETURN_INT32(res);
}
```

Пример использования `Datum`

PostgresPro

```c
#include "postgres.h"
#include "fmgr.h"

PG_FUNCTION_INFO_V1(my_factorial);

Datum
my_factorial(PG_FUNCTION_ARGS)
{
    int32 res, n = PG_GETARG_INT32(0);
    Datum d;

    if (n < 0)
        elog(ERROR, "Так нельзя!");
    if (n == 0)
        PG_RETURN_INT32(1);

    d = DirectFunctionCall1(my_factorial, Int32GetDatum(n - 1));
    res = DatumGetInt32(d) * n;
    PG_RETURN_INT32(res);
}
```

Пример использования `Datum`

PostgresPro

```c
#include "postgres.h"
#include "fmgr.h"

PG_FUNCTION_INFO_V1(my_factorial);

Datum
my_factorial(PG_FUNCTION_ARGS)
{
    int32 res, n = PG_GETARG_INT32(0);
    Datum d;

    if (n < 0)
        elog(ERROR, "Так нельзя!");
    if (n == 0)
        PG_RETURN_INT32(1);

    d = DirectFunctionCall1(my_factorial, Int32GetDatum(n - 1));
    res = DatumGetInt32(d) * n;
    PG_RETURN_INT32(res);
}
```

Пример использования `Datum`

PostgresPro

```c
#include "postgres.h"
#include "fmgr.h"

PG_FUNCTION_INFO_V1(my_factorial);

Datum
my_factorial(PG_FUNCTION_ARGS)
{
    int32 res, n = PG_GETARG_INT32(0);
    Datum d;

    if (n < 0)
        elog(ERROR, "Так нельзя!");
    if (n == 0)
        PG_RETURN_INT32(1);

    d = DirectFunctionCall1(my_factorial, Int32GetDatum(n - 1));
    res = DatumGetInt32(d) * n;
    PG_RETURN_INT32(res);
}
```

Пример использования `Datum`

PostgresPro

```c
#include "postgres.h"
#include "fmgr.h"

PG_FUNCTION_INFO_V1(my_factorial);

Datum
my_factorial(PG_FUNCTION_ARGS)
{
    int32 res, n = PG_GETARG_INT32(0);
    Datum d;

    if (n < 0)
        elog(ERROR, "Так нельзя!");
    if (n == 0)
        PG_RETURN_INT32(1);

    d = DirectFunctionCall1(my_factorial, Int32GetDatum(n - 1));
    res = DatumGetInt32(d) * n;
    PG_RETURN_INT32(res);
}
```

Пример использования Datum

PostgresPro

```c
#include "postgres.h"
#include "fmgr.h"

PG_FUNCTION_INFO_V1(my_factorial);

Datum
my_factorial(PG_FUNCTION_ARGS)
{
    int32 res, n = PG_GETARG_INT32(0);
    Datum d;

    if (n < 0)
        elog(ERROR, "Так нельзя!");
    if (n == 0)
        PG_RETURN_INT32(1);

    d = DirectFunctionCall1(my_factorial, Int32GetDatum(n - 1));
    res = DatumGetInt32(d) * n;
    PG_RETURN_INT32(res);
}
```

Пример использования `Datum`

PostgresPro

```c
#include "postgres.h"
#include "fmgr.h"

PG_FUNCTION_INFO_V1(my_factorial);

Datum
my_factorial(PG_FUNCTION_ARGS)
{
    int32 res, n = PG_GETARG_INT32(0);
    Datum d;

    if (n < 0)
        elog(ERROR, "Так нельзя!");
    if (n == 0)
        PG_RETURN_INT32(1);

    d = DirectFunctionCall1(my_factorial, Int32GetDatum(n - 1));
    res = DatumGetInt32(d) * n;
    PG_RETURN_INT32(res);
}
```

(*) Посмотрите как реализован вызов `DirectFunctionCall1`

Использование `Datum` как указателя

PostgresPro

```c
#include "postgres.h"
#include "fmgr.h"
#include "utils/builtins.h"

PG_FUNCTION_INFO_V1(hello_world);

Datum
hello_world(PG_FUNCTION_ARGS)
{
    char message[] = "Hello, World!";
    Datum datum;

    datum = CStringGetTextDatum(message);

    PG_RETURN_TEXT_P(datum);
}
```

Использование `Datum` как указателя

PostgresPro

```
#include "postgres.h"
#include "fmgr.h"
#include "utils/builtins.h"

PG_FUNCTION_INFO_V1(hello_world);

Datum
hello_world(PG_FUNCTION_ARGS)
{
    char message[] = "Hello, World!";
    Datum datum;

    datum = CStringGetTextDatum(message);

    PG_RETURN_TEXT_P(datum);
}
```

Использование `Datum` как указателя

PostgresPro

```c
#include "postgres.h"
#include "fmgr.h"
#include "utils/builtins.h"

PG_FUNCTION_INFO_V1(hello_world);

Datum
hello_world(PG_FUNCTION_ARGS)
{
    char message[] = "Hello, World!";
    Datum datum;

    datum = CStringGetTextDatum(message);

    PG_RETURN_TEXT_P(datum);
}
```

Использование `Datum` как указателя

PostgresPro

```c
#include "postgres.h"
#include "fmgr.h"
#include "utils/builtins.h"

PG_FUNCTION_INFO_V1(hello_world);

Datum
hello_world(PG_FUNCTION_ARGS)
{
    char message[] = "Hello, World!";
    Datum datum;

    datum = CStringGetTextDatum(message);

    PG_RETURN_TEXT_P(datum);
}
```

Использование `Datum` как указателя

PostgresPro

Более подробно о работе с не-регистровыми данными в PostgreSQL — на следующей лекции

Чуть не забыл

PostgresPro

(*) Один из макросов FMGR так и не был рассмотрен. Найдите какой и изучите его реализацию самостоятельно

PostgresPro

Вопросы

Николай Шаплов

E-mail: n.shaplov@postgrespro.ru
Matix: @dhyan:nataraj.su

PostgresPro

Последняя версия слайдов доступна на

https://gitlab.com/dhyannataraj/articles/-/tree/master/slides