

??? ????????????? PL/Python, ????????? ?????????? ? ?????????????

Чтобы использовать **PL/Python**, необходимо, чтобы модуль Python был установлен в вашей операционной системе для вашей версии PostgreSQL (например, пакет `postgresql-plpython3` в Linux), после чего его можно включить в нужной базе данных.

1. ????????? PL/Python

Активировать расширение нужно в целевой базе данных от имени суперпользователя (`postgres`):

SQL

```
CREATE EXTENSION IF NOT EXISTS plpython3u;
```

“ **Важная деталь:** Буква `u` на конце расширения означает *untrusted* (недоверенный). Это значит, что функции на PL/Python могут обращаться к файловой системе сервера, сети и операционной системе, поэтому создавать функции на PL/Python может только суперпользователь СУБД.

2. ??????? ???????? (Function)

Функции в PostgreSQL возвращают значение. В PL/Python вы можете импортировать любые сторонние библиотеки Python (например, `json`, `math`, `requests`, `pandas` и др.).

Пример функции для парсинга и валидации JSON-строки с расчетом итоговой стоимости:

SQL

```
CREATE OR REPLACE FUNCTION calculate_cart_total(cart_json text)
RETURNS numeric AS $$
    import json

    try:
        data = json.loads(cart_json)
        items = data.get("items", [])
```

```

total = 0.0
for item in items:
    price = item.get("price", 0)
    quantity = item.get("quantity", 1)
    total += price * quantity

return round(total, 2)
except Exception as e:
    # raise_notice или log позволяют отправлять сообщения в клиент/лог базы
    plpy.notice(f"Ошибка при обработке JSON: {e}")
    return 0.0
$$ LANGUAGE plpython3u;

```

Вызов функции:

SQL

```

SELECT calculate_cart_total('{"items": [{"price": 100.5, "quantity": 2}, {"price": 50.0,
"quantity": 1}]}');
-- Вернет: 251.00

```

3. ?????? ?????????? (Procedure)

Процедуры в PostgreSQL не возвращают результат и поддерживают управление транзакциями (`plpy.commit()`, `plpy.rollback()`).

Специальный модуль `plpy` внутри PL/Python позволяет выполнять SQL-запросы внутри базы.

Пример процедуры, которая находит устаревшие записи, отправляет их во внешнюю систему (имитация) и удаляет их:

SQL

```

CREATE OR REPLACE PROCEDURE cleanup_old_logs(days_old int)
AS $$
    import time

    # Выполняем SQL-запрос к БД с помощью plpy.execute
    plan = plpy.prepare(
        "SELECT id, message FROM system_logs WHERE created_at < NOW() - ($1 || '
days')::INTERVAL",

```

```

        ["int"]
    )
    result = plpy.execute(plan, [days_old])

    plpy.info(f"Найдено записей для удаления: {len(result)}")

    if len(result) > 0:
        # Логика обработки записей на Python
        for row in result:
            log_id = row["id"]
            # Здесь можно вызвать внешний API, сохранить резервную копию и т.д.

        # Удаление записей
        delete_plan = plpy.prepare(
            "DELETE FROM system_logs WHERE created_at < NOW() - ($1 || ' days')::INTERVAL",
            ["int"]
        )
        plpy.execute(delete_plan, [days_old])

        # Фиксация изменений
        plpy.commit()
$$ LANGUAGE plpython3u;

```

Вызов процедуры:

SQL

■ ■

```
CALL cleanup_old_logs(30);
```

???????????????? ???? plpy

Внутри PL/Python вам автоматически доступен встроенный объект `plpy`:

- `plpy.execute("SELECT ...")` — выполнение произвольного SQL-запроса.
- `plpy.prepare(query, [types])` — подготовка SQL-запроса для многократного выполнения.
- `plpy.notice(msg)`, `plpy.info(msg)`, `plpy.warning(msg)` — вывод сообщений пользователю.

- `plpy.error(msg)` — генерация исключения, отменяющего транзакцию.

?????????? ?? ??????? PL/Python ??????
????? (recordsets) ??????? ??????
??????????????? ? SQL-??????????

Да, функции PL/Python могут возвращать наборы строк. Их можно использовать непосредственно в блоке `FROM` SQL-запроса, как обычную таблицу или представление (View).

Возвращать наборы данных из PL/Python можно двумя основными способами:

1. **Список словарей / списков / кортежей** — если функция формирует и отдает все данные сразу.
2. **Итератор / Генератор (`yield`)** — если данных много и их нужно отдавать по одной строке (ленивые вычисления), чтобы не перегружать оперативную память.

???????? 1. ????????? ????? ?????? ?????? (List of Dicts)

Для возврата таблицы используется конструкция `RETURNS TABLE(...)` или `RETURNS SETOF record_type`. Внутри Python-кода вы возвращаете обычный список Python-словарей, где ключи соответствуют именам колонок.

SQL
■ ■

```
CREATE OR REPLACE FUNCTION generate_user_report(min_age int)
RETURNS TABLE(user_id int, full_name text, status text) AS $$
# В реальности данные могут запрашиваться из внешнего API или рассчитываться
users = [
    {"user_id": 1, "full_name": "Алексей Иванов", "age": 25},
    {"user_id": 2, "full_name": "Мария Сидорова", "age": 19},
    {"user_id": 3, "full_name": "Иван Петров", "age": 34}
]

result = []
for user in users:
    if user["age"] >= min_age:
        result.append({
            "user_id": user["user_id"],
            "full_name": user["full_name"],
```

```

        "status": "Adult" if user["age"] >= 21 else "Young"
    })

    return result
$$ LANGUAGE plpython3u;

```

????????????? ?????? ? SQL:

SQL

```

SELECT *
FROM generate_user_report(20)
WHERE status = 'Adult';

```

???????? 2. ?????????? ?????? ?????? (yield)

Если функция должна выгрузить 100 000+ строк из внешнего ресурса, сохранение их всех в список в памяти вызовет высокую нагрузку на RAM. В PL/Python можно использовать встроенный оператор Python `yield`.

SQL

```

CREATE OR REPLACE FUNCTION stream_numbers(limit_num int)
RETURNS TABLE(num int, square int) AS $$
    for i in range(1, limit_num + 1):
        # 0тдаем по одной строке за раз
        yield {"num": i, "square": i * i}
$$ LANGUAGE plpython3u;

```

????????????? ? SQL-????????:

SQL

```

SELECT num, square
FROM stream_numbers(5)
WHERE square > 10;

```

Результат:

num	square
4	16

num	square
5	25

3. SQL- (plpy.execute)

Если функция выполняет внутренний SQL-запрос с помощью plpy.execute(), возвращаемый объект PLPyResult уже является итерируемым набором строк, поэтому его можно вернуть напрямую:
SQL

```
CREATE OR REPLACE FUNCTION get_active_users()
RETURNS TABLE(id int, email text) AS $$
    # plpy.execute возвращает результат, который совместим с SETOF/TABLE
    res = plpy.execute("SELECT id, email FROM users WHERE is_active = true")
    return res
$$ LANGUAGE plpython3u;
```