

ЛР7. Пространственные данные. JSON

Пространственные данные

<https://dev.mysql.com/doc/refman/8.0/en/spatial-types.html>

Пространственные расширения MySQL позволяют создавать, хранить и анализировать следующие объекты:

- Типы данных для представления пространственных значений;
- Функции, выполняющие операции над пространственными данными;
- Пространственная индексация для ускорения времени выполнения пространственных операций.

MySQL поддерживает следующие типы данных, которые соответствуют классам OpenGIS:

1. GEOMETRY – пространственные значения любых типов.
2. POINT – точка.
3. LINESTRING – линия – одномерный объект, представляющий последовательность точек и соединяющих их линейных сегментов.
4. POLYGON – полигон – двумерная поверхность, хранящаяся в виде последовательности точек, определяющих внешнее ограничивающее кольцо, и внутренние кольца (последние могут отсутствовать).

Остальные типы пространственных данных позволяют хранить множество значений:

1. MULTIPOINT
2. MULTILINESTRING
3. MULTIPOLYGON
4. GEOMETRYCOLLECTION

Аналогично, GEOMETRYCOLLECTION позволяет хранить множество объектов любого типа. Остальные – только множество объектов соответствующего им типа.

На рисунке 1 представлены примеры объектов различных типов пространственных данных.




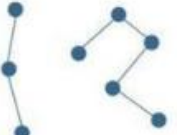


POINT		MULTIPOINT	
LINESTRING		MULTI-LINESTRING	
POLYGON		MULTI-POLYGON	

Рисунок 1 – Примеры объектов пространственных данных разных типов

Пространственные данные в MySQL хранятся в специальном формате. Чтобы преобразовать пространственные данные в этот формат используется одна из следующих функций (в зависимости от типа пространственных данных):

- ST_GeomFromText,
- ST_PointFromText,
- ST_LineStringFromText,
- ST_PolygonFromText,
- ST_GeomCollFromText.

Данные функции преобразует описание геометрии из Well-Known Text формата в используемый БД формат. Например,

```
INSERT INTO geom VALUES (ST_GeomFromText('POINT(1 1)'));
```

Для извлечения пространственных данных из таблицы в формате Well-Known Text используется функция ST_AsText:

```
SELECT ST_AsText(g) FROM geom;
```

Некоторые функции, выполняющие операции над пространственными данными

<https://dev.mysql.com/doc/refman/8.0/en/spatial-analysis-functions.html>

ST_X(point) – возвращает x координату точки *point*.

ST_Y(point) – возвращает y координату точки *point*.

ST_Buffer(g, d) – принимает объект геометрии *g* и расстояние *d*, возвращает объект геометрии, который представляет буфер вокруг объекта.

Пример (рисунок 2):

```
SELECT ST_Buffer(ST_GeomFromText('POINT (10 10)'),10);
```

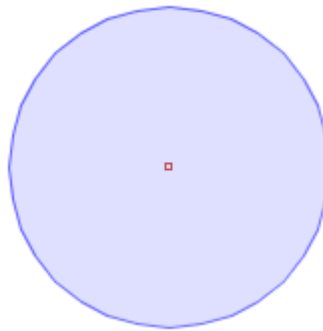


Рисунок 2 – Результат работы функции ST_Buffer

ST_Distance(g1, g2) – возвращает расстояние между двумя объектами *g1* и *g2*.

ST_Envelope(g) – возвращает минимальный ограничивающий прямоугольник для объекта *g*.

ST_Union(g1, g2) – возвращает результат объединения объектов геометрии *g1* и *g2*.

Функции, определяющие пространственные соотношения объектов:

- **ST_Contains**(g1, g2) - возвращает 1, если *g1* полностью содержит *g2*, и 0 в противном случае.
- **ST_Crosses**(g1, g2) - возвращает 1 или 0, указывая, пересекает ли пространственно *g1* объект *g2*.

Два объекта геометрии пространственно пересекаются, если их пространственное отношение обладает следующими свойствами:

1. Если $g1$ и $g2$ не имеют размерности 1: $g1$ пересекает $g2$, если внутренняя часть $g2$ имеет точки, общие с внутренней частью $g1$, но $g2$ не покрывает всю внутреннюю часть $g1$.

2. Если и $g1$, и $g2$ имеют размерность 1: если линии пересекаются друг с другом в конечном количестве точек (то есть, нет общих отрезков, только общие точки).

- **ST_Disjoint**($g1$, $g2$) - возвращает 1, если пересечение двух геометрий представляет собой пустое множество. В противном случае возвращается 0.
- **ST_Intersects**($g1$, $g2$) - возвращает 1, если пересечение двух геометрий не образует пустого множества. В противном случае возвращается значение 0.
- **ST_Overlaps**($g1$, $g2$) - возвращает значение 1, если пересечение объектов приводит к получению объекта геометрии той же размерности, но не равного исходному объекту. В противном случае возвращается значение 0.
- **ST_Touches**($g1$, $g2$) - возвращает значение 1, если никакие из общих точек геометрий не пересекают их внутренних частей. В противном случае возвращается значение 0.
- **ST_Within**($g1$, $g2$) - возвращает значение 1, если $g1$ пространственно находится в границах $g2$. В противном случае возвращается значение 0.

JSON

JSON— текстовый формат обмена данными, широко используемый для обмена данными между программными системами.

В качестве значений в JSON могут быть использованы:

- Запись — это неупорядоченное множество пар ключ:значение, заключённое в фигурные скобки «{ }». Ключ описывается строкой, между ним и значением стоит символ «:». Пары ключ-значение отделяются друг от друга запятыми.
- Массив (одномерный) — это упорядоченное множество значений. Массив заключается в квадратные скобки «[]». Значения разделяются

запятые. Массив может быть пустым, т.е. не содержать ни одного значения.

- Число (целое или вещественное).
- Литералы `true` (логическое значение «истина»), `false` (логическое значение «ложь») и `null`.
- Строка — это упорядоченное множество из нуля или более символов юникода, заключённое в двойные кавычки. Символы могут быть указаны с использованием escape-последовательностей, начинающихся с обратной косой черты «\» (поддерживаются варианты `\'`, `\"`, `\\`, `\`, `\t`, `\n`, `\r`, `\f` и `\b`), или записаны шестнадцатеричным кодом в кодировке Unicode в виде `\uFFFF`.

Следующий пример показывает JSON-представление данных об объекте, описывающем человека. В данных присутствуют строковые поля имени и фамилии, информация об адресе и массив, содержащий список телефонов. Как видно из примера, JSON поддерживает вложенные структуры.

```
{
  "firstName": "Иван",
  "lastName": "Иванов",
  "address": {
    "streetAddress": "Московское ш., 101, кв.101",
    "city": "Ленинград",
    "postalCode": 101101
  },
  "phoneNumbers": [
    "812 123-1234",
    "916 123-4567"
  ]
}
```

Использование JSON в MySQL

<https://dev.mysql.com/doc/refman/8.0/en/json-functions.html>

Для работы с JSON-полями MySQL поддерживает тип данных JSON.

Добавление данных

Добавление записи в БД, содержащей json поле, может быть выполнено несколькими способами:

1. Для добавления записи, содержащей поле с json данными, достаточно добавить правильно сформированную json строку в значение этого поля в INSERT запросе.
2. Используя функцию JSON_OBJECT можно сформировать json строку в процессе добавления данных. Эта функция принимает список пар ключ-значение вида

```
JSON_OBJECT(key1, value1, key2, value2, ... key(n), value(n))
```

и возвращает JSON объект. При этом если значение является массивом, используется функция

```
JSON_ARRAY(value1, value2, ..., value(n)),
```

которая возвращает массив json объектов.

Чтение данных:

Для выбора нужных данных по json полю применяется функция JSON_EXTRACT(column, path), которая принимает в качестве аргумента поле таблицы с json данными и путь для перемещения по JSON объекту. Такие же действия выполняет конструкция вида "column->path".

Например, запросы

1. SELECT * FROM product WHERE attr->'\$.screen' > 30
2. SELECT * FROM product WHERE JSON_EXTRACT(attr, '\$.screen') > 30

являются эквивалентными.

Обновление данных:

Для обновления JSON значений используются следующие функции: `JSON_INSERT(column, path, value)`, добавляет новый ключ и соответствующее ему значение, `JSON_REPLACE(column, path, value)` заменяет значение уже существующего ключа, `JSON_SET(column, path, value)` заменяет существующие значения и добавляет несуществующие значения. Данные функции возвращают json объект с примененными изменениями.

Удаление данных:

Для удаления данных из json используется функция `JSON_REMOVE(column, path)`, которая удаляет данные по указанному пути `path` возвращает измененный json.

ЛР7. Задание

Пространственные данные

Задания на пространственные данные выполняются с БД sakila. Пространственные данные хранятся в поле `location` таблицы `address`

1. Написать функцию проверки валидности данных: значения долготы должны находиться в диапазоне $(-180, 180]$, значения широты должны находиться в диапазоне $[-90, 90]$. Координаты $(0, 0)$ также считать невалидными. В следующих запросах учитывать только адреса с валидной геометрией.
2. Найти всех покупателей, проживающих внутри заданного полигона (например, "POLYGON ((-60 -40, -57.9 -37.3, -57.9 -34.3, -59.1 -34.3, -60 -40))").
3. Для первого покупателя из указанной страны определить количество покупателей, проживающих на заданном расстоянии (в градусах), используя функцию `ST_Buffer`.
4. Для страны, в которой живет наибольшее число покупателей, сформировать полигон, являющийся минимальным ограничивающим прямоугольником координат места жительства покупателей из этой страны.
5. Определить пространственные соотношения полигонов, полученных в заданиях №3 и №4.

JSON

Задания выполняются в своей БД.

6. Создать новую таблицу или изменить существующую, добавив поле типа JSON, заполнить таблицу данными. Минимум одно из значений записи должно представлять из себя вложенную структуру, одно – массив. Каждая запись должна содержать не менее 5 ключей.
7. Выполнить запрос, возвращающий содержимое данной таблицы, соответствующее некоторому условию, проверяющему значение атрибута вложенной структуры.
8. Выполнить запрос, добавляющие новую пару «ключ-значение» к заданной строке таблицы, причем «значение» является массивом.
9. Выполнить запрос, изменяющий значение по некоторому существующему ключу в заданной строке таблицы.
10. Выполнить запрос, осуществляющий удаление созданной пары «ключ-значение».