JSON（JavaScript Object Notation） 属于非关系数据类型。

**JSON类型简介**

JSON又分为JSON和JSONB

JSON：把输入的数据原封不动地存放到数据库中， 使用时需要重新解析数据，写入性能好，查询性能差。JSON不支持创建索引。

JSONB：在存储时就把JSON解析成二进制格式， 使用时就无须再次解析， 写入性能差，查询性能好。JSONB支持创建索引。

因为JSON类型是把输入的整个字符串原封不改动地保存到数据库中， 因此JSON串中key之间多余的空格也会被保留下来。 而且， 如果JSON串中有重复的key， 这些重复的key也会保留下来， 同时也会保留输入时JSON串中各个key的顺序。 而JSONB类型则恰恰相反， 既不会保留多余的空格， 也不会保留key的顺序和重复的key。

JSON和JSONB在使用过程中还存在差异，JSONB输出的键的顺序和输入的不一样，而JSON输出的键的顺序和输入的完全一样。

在大多数应用场景中建议使用JSONB。

**JSON类型的使用**

select '{"id":1,"name":"zhangsan","age":22}'::json;

select json '{"id":1,"name":"zhangsan","age":22}';

select '{"id":1,"name":"zhangsan","age":22}'::jsonb;

**JSON类型的操作符**

json 和 jsonb 操作符

|  |  |  |
| --- | --- | --- |
| 操作符 | 描述 | 例子 |
| json -> text → json  jsonb -> text → jsonb | 提取对象字段 | '{"id":1,"name":"zhangsan","age":22}' -> 'name' → "zhangsan" |
| json ->> text → text  jsonb ->> text → text | 提取对象字段 | '{"id":1,"name":"zhangsan","age":22}' ->> 'name' → zhangsan |

**仅适用于JSONB类型的操作符**

|  |  |  |
| --- | --- | --- |
| 操作符 | 描述 | 例子 |
| @> | 包含 | '{"id":1,"name":"zhangsan","age":22}'::jsonb @> '{"name":"zhangsan"}'::jsonb → t |
| <@ | 被包含 | '{"name":"zhangsan"}'::jsonb <@ '{"id":1,"name":"zhangsan","age":22}'::jsonb → t |
| - | 删除键(以及它的值) | '{"id":1,"name":"zhangsan","age":22}'::jsonb - 'age' → {"id": 1, "name": "zhangsan"} |
| || | 连接 | '{"id":1,"name":"zhangsan"}'::jsonb || '{"age":22}'::jsonb →  {"id": 1, "age": 22, "name": "zhangsan"} |

**JSON类型的函数**

|  |  |  |
| --- | --- | --- |
| 函数名 | 描述 | 例子 |
| to\_json  to\_jsonb | 转换为json或jsonb | to\_json((1,2,3)) → {"f1":1,"f2":2,"f3":3}  to\_jsonb((1,2,3)) → {"f1": 1, "f2": 2, "f3": 3} |
| row\_to\_json | 将行转换为JSON对象 | row\_to\_json(row(1,'zhangsan',22)) → {"f1":1,"f2":"zhangsan","f3":22} |

**JSON/JSONB类型数据的处理函数**

|  |  |  |
| --- | --- | --- |
| 函数名 | 描述 | 例子 |
| json\_each  jsonb\_each | 将对象展开 | select \* from json\_each('{"id":1,"name":"zhangsan","age":22}') →  key | value  ------+------------  id | 1  name | "zhangsan"  age | 22 |
| json\_each\_text  jsonb\_each\_text | 将对象展开 | select \* from json\_each('{"id":1,"name":"zhangsan","age":22}') →  key | value  ------+------------  id | 1  name | zhangsan  age | 22 |
| jsonb\_pretty | 格式化输出 | jsonb\_pretty('{"id":1,"name":"zhangsan","age":22}'::jsonb) →  { +  "id": 1, +  "age": 22, +  "name": "zhangsan"+  } |

其他JSON/JSONB类型数据的处理函数：http://www.postgres.cn/docs/14/functions-json.html

**JSON类型的索引使用方法**

JSON类型无法直接创建索引，但可以在的列上创建函数索引。

JSONB类型可以直接创建GIN索引。 GIN索引可以高效地从JSONB内部的key/value对中搜索数据。

在JSONB上创建GIN索引的方式：

1.使用默认的jsonb\_ops操作符创建。

CREATE INDEX idx\_name ON table\_name USING gin (index\_col);

2.使用jsonb\_path\_ops操作符创建。

CREATE INDEX idx\_name ON table\_name USING gin (index\_col jsonb\_path\_ops);

这两种方式的区别在于， 采用jsonb\_ops创建的索引的每个key和value都是作为一个单独的索引项， 而jsonb\_path\_ops则只为每个value创建一个索引项。

例如： 有一个项“{”foo“:{”bar“:”baz“}}”， 对于jsonb\_path\_ops来说， 是把“foo”“bar”和“baz”组合成一个Hash值作为索引项， 而jsonb\_ops则会分别为“foo”“bar”“baz”创建3个索引项。 因为少了很多索引项， 所以通常jsonb\_path\_ops的索引要比jsonb\_ops的小很多， 这样当然也就会带来性能的提升。

为JSONB创建GIN索引：

CREATE INDEX ON t1 USING gin (col1);

CREATE INDEX ON t1 USING gin (col2 jsonb\_path\_ops);

EXPLAIN ANALYZE SELECT \* FROM t1 WHERE col1 @> '{"name":"xxx"}';

EXPLAIN ANALYZE SELECT \* FROM t1 WHERE col2 @> '{"name":"xxx"}';

SELECT \* FROM t1 WHERE col1 @> '{"name":"xxx"}'

SELECT \* FROM t1 WHERE col2 @> '{"name":"xxx"}'

查看两个索引的大小：

select pg\_relation\_size('idx1');

select pg\_relation\_size('idx2');

jsonb\_path\_ops类型的索引要比jsonb\_ops的小

**JSON/JSONB性能测试**

创建基表

create table users(id int,name text,age int);

insert into users select n,n||'\_user',(random()\*10)::int+20 from generate\_series(1,1000000) n;

创建json/jsonb表

create table user\_json(id serial,info json);

create table user\_jsonb(id serial,info jsonb);

插入性能测试

insert into user\_json(info) select to\_json(users) from users;

insert into user\_jsonb(info) select to\_jsonb(users) from users;

占用空间对比

\dt+ user\_json

\dt+ user\_jsonb

查询性能测试

select \* from user\_json where info->>'name'='1\_user';

select \* from user\_jsonb where info->>'name'='1\_user';