

# 数据库设计中对JSON的使用

孙鹏，英资教育，CEO



BEIJING YINGZIEDU TECHNOLOGY CO., LTD.

1

为何PostgreSQL要支持JSON ?

2

数据库设计中对JSON的3种使用方式

3

JSONB的存储和访问效率

4

PostgreSQL中验证JSON Schema

5

Node. js使用PostgreSQL中的JSON数据

# 1. 为何PostgreSQL要支持JSON ?

JSON规范: <http://www.ietf.org/rfc/rfc4627.txt>, GOOGLE “JSON” 指数:

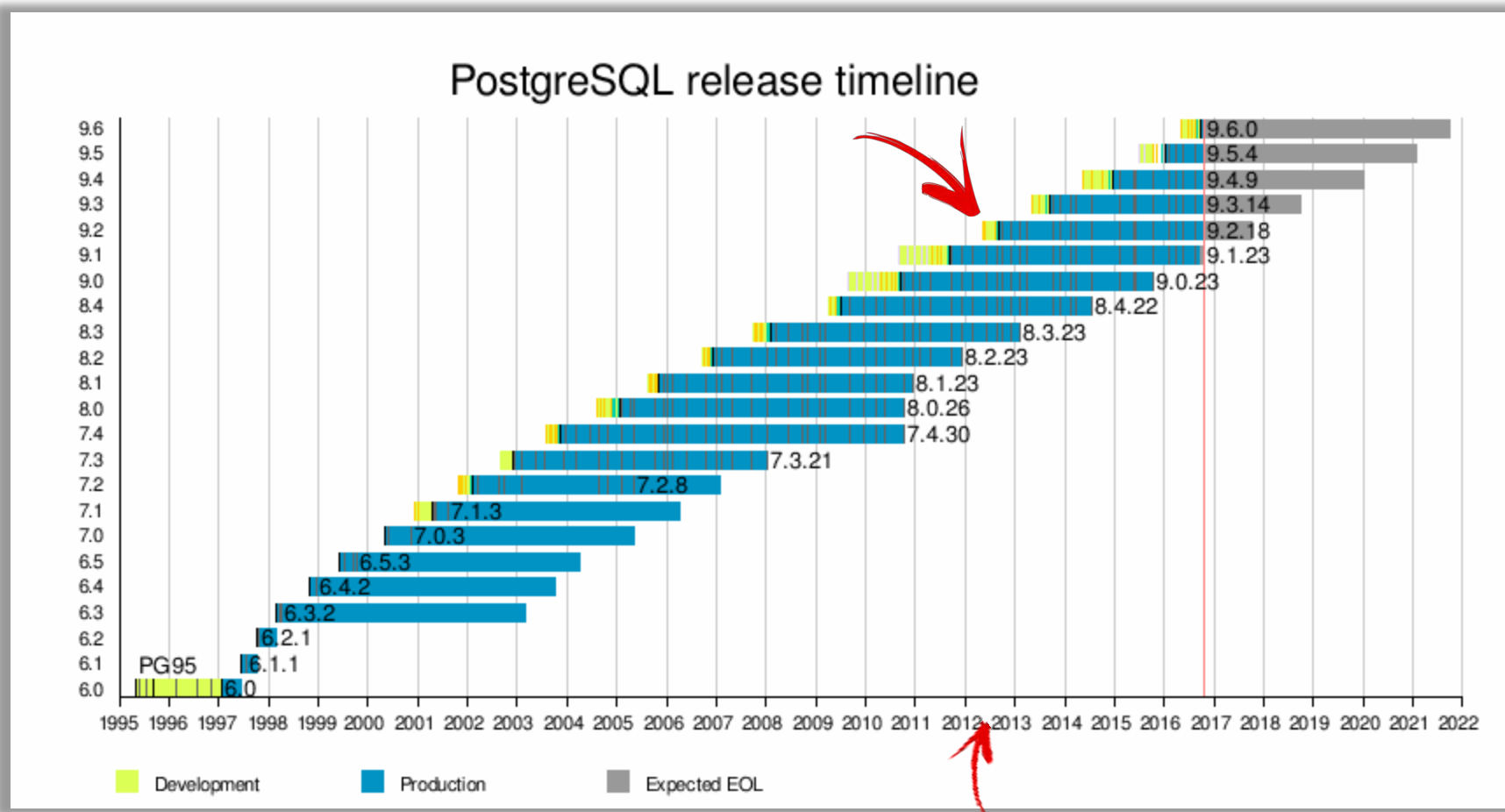


[1] <https://datatracker.ietf.org/doc/search/?name=json&activeDrafts=on&rfcs=on>

[2] The GeoJSON Format, <https://datatracker.ietf.org/doc/rfc7946/>

# 1. 为何PostgreSQL要支持JSON ?

PostgreSQL中对JSON的支持, 从9.2版本 (2012-09-10) 开始, 作为native [JSON](#) support:



[1] <https://en.wikipedia.org/wiki/PostgreSQL>

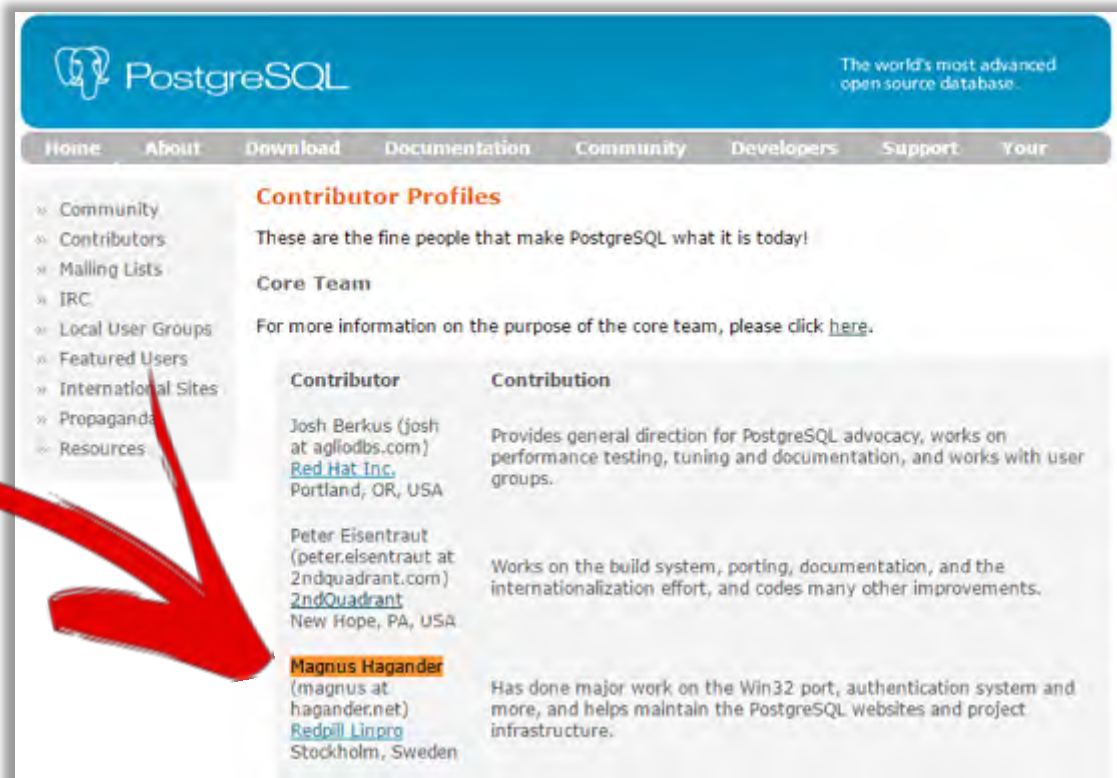
# 1. 为何PostgreSQL要支持JSON ?

谁先发起的? Google Summer of Code Project, 2010, Be committed as an extension for PostgreSQL 9.1

— [https://wiki.postgresql.org/wiki/JSON\\_datatype\\_GSoC\\_2010](https://wiki.postgresql.org/wiki/JSON_datatype_GSoC_2010)

Student name: Joseph Adams

Mentor name: **Magnus Hagander**



[1] Web-Scale PostgreSQL , <http://s3.amazonaws.com/ppt-download/postgresqlwebscale-140812090723-phpapp02.pdf?response-content-disposition=attachment&Signature=YhRscQIoU3MvVVAZyJ8HbDCTVZU%3D&Expires=1477362999&AWSAccessKeyId=AKIAJ6D6SEMXSASXHDAQ>

# 1. 为何PostgreSQL要支持JSON ?

支持情况如何？（红色表示新增）

version	Data Type	Operators	Functions	Aggregate Functions	Indexing
9.2.18	json		array_to_json row_to_json		
9.3.14	json	-> ->> #> #>>	array_to_json row_to_json to_json json_array_length json_each json_each_text json_extract_path json_extract_path_text json_object_keys json_populate_record json_populate_recordset json_array_elements	json_agg	

[1] PG历代版本对JSON的支持情况（PG JSON系列7），  
<http://blog.sciencenet.cn/home.php?mod=space&uid=419883&do=blog&quickforward=1&id=1010771>

# 1. 为何PostgreSQL要支持JSON ?

续上表

9.4.9	json jsonb	jsonb Operators: 1. @> 2. <@ 3. ? 4. ?  5. ?&	JSON Creation Functions: 1. array_to_json 2. json_build_array 3. json_build_object 4. json_object Processing Functions: 1. json_array_elements_text 2. jsonb_array_elements_text 3. json_typeof 4. jsonb_typeof 5. json_to_record 6. jsonb_to_record 7. json_to_recordset 8. jsonb_to_recordset	json_object_agg	Support
9.5.4	json jsonb	jsonb Operators: 1.    2. - 3. #-	JSON Creation Functions: 1. to_jsonb 2. jsonb_build_array 3. jsonb_build_object 4. jsonb_object Processing Functions: 1. json_strip_nulls 2. jsonb_strip_nulls 3. jsonb_set 4. jsonb_pretty	jsonb_agg jsonb_object_agg	Support
9.6.0	json jsonb	无变化	JSON Creation Functions:无变化 Processing Functions: 1. jsonb_insert	无变化	Support

# 1. 为何PostgreSQL要支持JSON ?

为何要支持JSON:

- 个人觉得一个很重要的原因是, 现在客户端 (Web/APP/微信) 编程都用H5, 服务端编程逐渐要求是RESTful API的风格接口, 并不要求服务端编程花费大量的精力, 大都是增删改查数据库, 没有以往2B业务复杂的中间层业务逻辑, 如果数据库支持JSON, 则大大简化中间层的工作量, 进而跟上当前的开发潮流, 催生全栈工程师的需求, 从而降低了中小企业的用人成本

Josh Berkus (<https://www.postgresql.org/community/contributors/>) 写的一篇博文, Why HStore2/jsonb is the most important patch of 9.4 :

- Open source databases rise and fall on the popularity of the programming languages which use those databases. ...
- If you've watched database adoption trends for the last 20 years like I have, this is alarming. We are in danger of being sidelined....



[1] <http://www.databasesoup.com/2014/02/why-hstore2jsonb-is-most-important.html>



1

为何PostgreSQL要支持JSON ?

2

数据库设计中对JSON的3种使用方式

3

JSONB的存储和访问效率

4

PostgreSQL中验证JSON Schema

5

Node. js使用PostgreSQL中的JSON数据

## 2. 数据库设计中对JSON的3种使用方式

分为：

1. 只使用row\_to\_json()及相关函数获取JSON数据，不使用json/jsonb定义表中的字段
2. 表中的一整行用一个jsonb字段代替，去范式化（外键等），增加扩展性
3. 整个表用一个jsonb数组表示，消除join

## 2. 数据库设计中对JSON的3种使用方式

### 2.1 只使用row\_to\_json()及相关函数

查询结果只有一行数据：

- `SELECT row_to_json(a.*) from TUsers a where a.id = 1`

那如果我获得的不是a.\*，即不需要一整行数据（如密码不想返回）怎么办？如下会报错：

- `SELECT row_to_json(a.id,a.realName) from TUsers a where a.id = 1`

解决办法有：

- `SELECT row_to_json(a.*) from (select b.id,b.realName from TUsers b where b.id = 1 ) a`

或者使用WITH：

- `WITH myInfo AS (select a.id,a.realName from TUsers a where a.id = 1 )  
SELECT row_to_json(b.*) from myInfo b`

或者用 -：

- `SELECT row_to_json(a.*)::jsonb-'id' from TUsers a where a.id = 1`

[1] <http://blog.sciencenet.cn/home.php?mod=space&uid=419883&do=blog&quickforward=1&id=1007839>

## 2. 数据库设计中对JSON的3种使用方式

### 2.1 只使用row\_to\_json()及相关函数

查询结果有多行数据，然后形成json数组：

- WITH myProjects AS (select a.id,a.title from TProject a where a.creatorId = 1)  
SELECT row\_to\_json(b.\*) from  
(SELECT array\_to\_json(array(select row\_to\_json(myProjects.\*) from myProjects),false) as myProjects) b

## 2. 数据库设计中对JSON的3种使用方式

### 2.1 只使用row\_to\_json()及相关函数

返回某些表的一行数据，加上某些表的多行数据：如获取用户基本信息以及用户参与的项目：

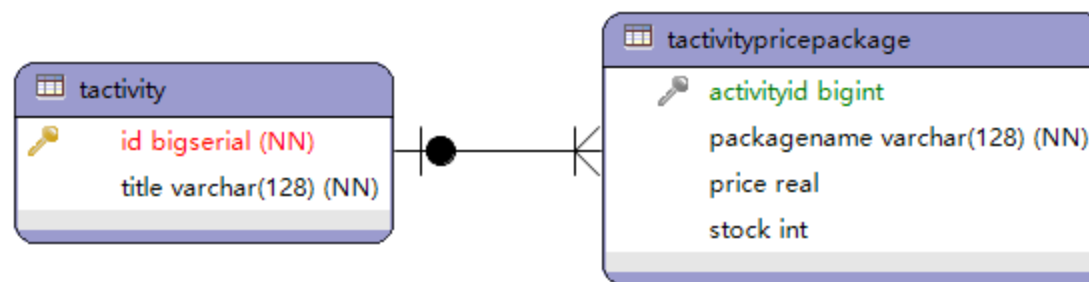
- WITH myInfo AS (select id,realName from TUsers where id = 1 ), -- 一行数据  
     myProjects AS (select a.id,a.title from TProject a,myInfo b where a.creatorId = b.id) -- 多行数据  
 SELECT row\_to\_json(x.\*) from(  
     select c.\*,d.\* from myInfo c,  
     (SELECT array\_to\_json(array(select row\_to\_json(myProjects.\*)) from myProjects),false) as myProjects)  
 d -- 把多行数据生成一行一列json数组  
 ) x

## 2. 数据库设计中对JSON的3种使用方式

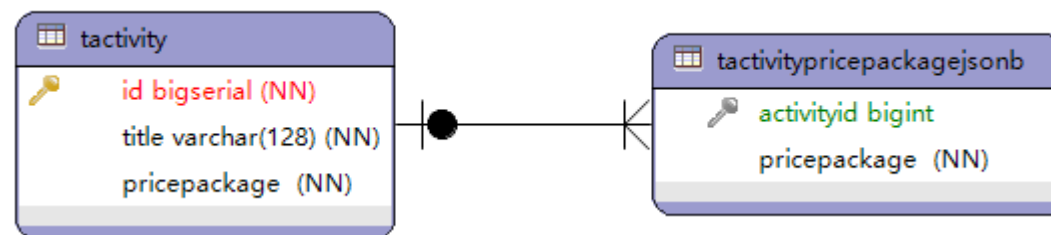
### 2.2 表中的一整行用一个 jsonb 字段代替



一个应用场景：那设计数据库的时候，就有 jsonb 方案。一种是常规的做一个关联表。如下图：



另外一种就是用 jsonb **对象** 来表示 TActivityPricePackage 表中的一行记录，增加灵活性，将来方便扩展新属性：



## 2. 数据库设计中对JSON的3种使用方式

### 2.2 表中的一整行用一个jsonb字段代替

Jsonb对象来表示该项活动的票价套餐：

- --演出活动套餐价格表，用JSONB表示

```
CREATE TABLE TActivityPricePackageJSONB (  
  activityId bigint DEFAULT NULL REFERENCES TActivity (id) match simple on delete CASCADE  
  pricepackage jsonb NOT NULL -- 价格套餐  
)WITH ( OIDS=FALSE);
```

- insert into TActivityPricePackageJSONB values(1, '{"packagename":"成人票","price":189,"stock":100}');

## 2. 数据库设计中对JSON的3种使用方式

### 2.2 表中的一整行用一个 jsonb 字段代替

增（增加 jsonb 字段中的一个属性，更复杂的可以使用 jsonb\_insert() 函数）：

- `update TActivityPricePackageJSONB set pricepackage = pricepackage || '{"location":"北京"}';`

删（删除 jsonb 字段中的一个属性）：

- `update TActivityPricePackageJSONB set pricepackage = pricepackage - 'stock';`

改：

- `update TActivityPricePackageJSONB set pricepackage = jsonb_set(pricepackage, '{packagename}', '"大人"', true);`
- 注意：'{packagename}' 不能是 'packagename'

查：

- `select * from TActivityPricePackageJSONB where pricepackage @> '{"packagename":"大人"}'`

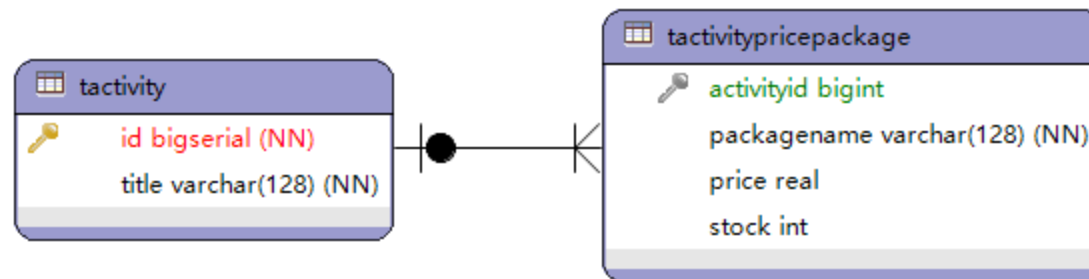


## 2. 数据库设计中对JSON的3种使用方式

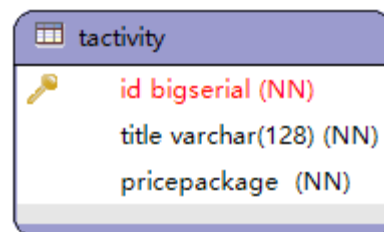
### 2.3 整个表用一个jsonb数组表示，消除join



一个应用场景：那设计数据库的时候，就有jsonb方案。一种是常规的做一个关联表。如下图：



另外一种就是用jsonb数组来表示该项活动的票价套餐，消除TActivityPricePackage表：



[1] <http://blog.sciencenet.cn/blog-419883-1008040.html>

## 2. 数据库设计中对JSON的3种使用方式

### 2.3 整个表用一个jsonb数组表示，消除join

jsonb来表示该项活动的票价套餐：

- -- 演出活动表

```
CREATE TABLE TActivity (
```

```
  id bigint DEFAULT nextval('jstest_uuid_seq') PRIMARY KEY,-- 活动id
```

```
  title character varying(128) NOT NULL,-- 活动名称
```

```
  pricepackage jsonb NOT NULL -- 价格套餐)WITH ( OIDS=FALSE);
```

- insert into TActivity values(1,'演出活动标题1',["packagename":"成人票", "price":189,"stock":100}, {"packagename":"儿童票（12岁以下）", "price":66,"stock":20}, {"packagename":"成人+儿童套票", "price":128,"stock":10}]);

## 2. 数据库设计中对JSON的3种使用方式

### 2.3 整个表用一个jsonb数组表示，消除join

增（增加jsonb数组中的一个jsonb对象）：

- `update tactivity set pricepackage = '{"packagename":"成人票新增3","price":189,"stock":100}' || pricepackage where id = 1; -- 新值在前面`
- `update tactivity set pricepackage = jsonb_insert(pricepackage, '-1', '{"packagename":"成人票新增100","price":189,"stock":100}', true) where id = 1; -- 使用函数jsonb_insert`

删（删除jsonb数组中的一个jsonb对象）：

- `update tactivity set pricepackage = pricepackage - 0 where id = 1;`
- `update tactivity set pricepackage = pricepackage #- '{0,stock}' where id = 1;`

改：

- `update tactivity set pricepackage = jsonb_set(pricepackage, '{0}', '{"price":189,"packagename":"成人票00","stock":100}', false) where id = 1;`
- -- 9.6.0版本里jsonb\_set()有个bug，截至到2016.10.12，社区2天后已fixed（赞效率），见我的博客，参考[1]
- `update tactivity set pricepackage = jsonb_set(pricepackage, '{0}', '{"price":189,"packagename":"成人票00","stock":100}', true) where id = 1;`
- `update tactivity set pricepackage = jsonb_set(pricepackage, '{0,packagename}', '"成人票000"', false) where id = 1`

[1] <http://blog.sciencenet.cn/home.php?mod=space&uid=419883&do=blog&quickforward=1&id=1008263>

[2] <https://www.postgresql.org/message-id/1269.1476332809@sss.pgh.pa.us> , Yeah, this is broken. Fixed, thanks for the report! regards, tom lane

## 2. 数据库设计中对JSON的3种使用方式

### 2.3 整个表用一个jsonb数组表示，消除join

查（找出符合条件 “package name = 儿童票（3-5岁）” 的那些活动？）：

- `select * from TActivity a where a.pricepackage @> '[{"package name": "儿童票（3-5岁）"}']::jsonb;`

查（找出符合条件 “数组中第0个元素的price < 100” 的那些活动？）：

- `select * from TActivity a where cast(a.pricepackage->0->>'price' as int) < 100;`

查（找出符合条件 “数组中所有元素的price < 100” 的那些活动？）：

- `WITH rows_filtered AS ( SELECT DISTINCT id FROM ( select a.id, b.* from tactivity a, jsonb_to_recordset(a.pricepackage) as b(package name text, price numeric, stock numeric)) c WHERE c.price < 100)  
SELECT * FROM TActivity a WHERE a.id IN (SELECT * FROM rows_filtered);`

1

为何PostgreSQL要支持JSON ?

2

数据库设计中对JSON的3种使用方式

3

JSONB的存储和访问效率

4

PostgreSQL中验证JSON Schema

5

Node. js使用PostgreSQL中的JSON数据

### 3. JSONB的存储和访问效率

我们的第1个问题是：相对于普通表，JSONB的存储大小是否过大？还是变化不大。应用场景：



```
CREATE TABLE
R_PIC_PERSON_PLAINTAB
LE (
id serial,-- 照片id
position text, -- 如"前1排左1"
name text,--如"张三"
phone text,
city text,
organization text,
attendanceYear numeric,
graduationYear numeric
)WITH (OIDS=FALSE);
```

```
CREATE TABLE
R_PIC_PERSON_JSONB_NOAR
RAY (
id serial,-- 照片id
person jsonb NOT NULL -- 格式为:
{"position":"前1排左1","name":"张
晓陆
","phone":"18910180100","city":"济
南","organization":"中科院计算所
","attendanceYear":2012,"graduatio
nYear":2016}
)WITH (OIDS=FALSE);
```

```
CREATE TABLE
R_PIC_PERSON_JSONB_USEARRAY (
id serial,-- 照片id
persons jsonb NOT NULL -- 格式为:
[{"position":"前1排左1","name":"张晓陆
","phone":"18910180100","city":"济南
","organization":"中科院计算所
","attendanceYear":2012,"graduationYear":2016},
{"position":"前1排左2","name":"张晓陆
","phone":"18910180100","organization":"中科院
计算所
","attendanceYear":2012,"graduationYear":2016}]
)WITH (OIDS=FALSE);
```

[1] <http://blog.sciencenet.cn/home.php?mod=space&uid=419883&do=blog&quickforward=1&id=1009363> , PG中JSONB存储和访问效率 (PG JSON系列5)

[2] [http://www.yingziedu.com/facelink/publish/index\\_city.html?activityid=18](http://www.yingziedu.com/facelink/publish/index_city.html?activityid=18), 校园时代

### 3. JSONB的存储和访问效率

灌数据:

```
insert into
R_PIC_PERSON_PLAINTABLE
select r/20,
round(random()*200000) || '_position',
round(random()*200000) || '_name',
round(random()*200000) || '_phone',
round(random()*200000) || '_city',
round(random()*200000) ||
'_organization',
round(random()*200000) ,
round(random()*200000)
  from generate_series(1,200000) as r;
```

```
insert into
R_PIC_PERSON_JSONB_NOARRAY
SELECT id,
row_to_json(R_PIC_PERSON_PLAINTABLE)::
jsonb - 'id' from
R_PIC_PERSON_PLAINTABLE;
```

```
insert into
R_PIC_PERSON_JSONB_USEARRAY
select id,array_to_json(array_agg(person))
  from R_PIC_PERSON_JSONB_NOARRAY
 group by id;
```

[1] francs的 PostgreSQL9.4: jsonb 性能测试, <http://francs3.blog.163.com/blog/static/40576727201452293027868/>



### 3. JSONB的存储和访问效率

测试结果:

	2K	2万	20万	200万	2000万
PLAINTABLE	248	2264	23	240	2521
JSONB_NOARRAY	496	4856	49	488	5042
JSONB_USEARRAY	136	1176	13	156	异常
单位	KB	KB	MB	MB	MB

小结：总体上来讲本例中，在没有建索引的情况下，R\_PIC\_PERSON\_JSONB\_NOARRAY表大概是普通的R\_PIC\_PERSON\_PLAINTABLE表的2倍大的空间，R\_PIC\_PERSON\_JSONB\_USEARRAY表则为R\_PIC\_PERSON\_JSONB\_NOARRAY表的50-65%左右



### 3. JSONB的存储和访问效率

在单一字段上建立的Btree和GIN索引，测试结果：

Btree	2K	2万	20万	200万
PLAINTABLE	0.078125	0.617188	6.039063	60.179688
JSONB_NOARRAY	0.09375	0.789063	7.773438	77.484375
JSONB_USEARRAY	0.15625	1.609375	19.53906	244.11719
单位	MB	MB	MB	MB

GIN	2K	2万	20万	200万
PLAINTABLE	0.078125	0.617188	6.039063	60.17969
JSONB_NOARRAY	0.109375	0.960938	9.578125	62.58594
JSONB_USEARRAY	0.617188	6.117188	44.97656	476.8828
单位	MB	MB	MB	MB

小结：在单一字段上建立的Btree和GIN索引，R\_PIC\_PERSON\_PLAINTABLE表和R\_PIC\_PERSON\_JSONB\_NOARRAY表建立索引占用的存储空间略大一些（如上表中标红色的部分），但差别不大。

### 3. JSONB的存储和访问效率

问题 2.JSONB的建立索引后，访问效率提升是多少？：

```
CREATE INDEX idx_gin_jsonb_all_noarray ON R_PIC_PERSON_JSONB_NOARRAY USING GIN ((person));  
CREATE INDEX idx_gin_jsonb_all_userarray ON R_PIC_PERSON_JSONB_USEARRAY USING GIN ((persons));
```

在R\_PIC\_PERSON\_JSONB\_NOARRAY 上面执行（在200万行的记录里）：

```
explain analyze select * from R_PIC_PERSON_JSONB_NOARRAY where person @> '{"name":"1349018_name"}';
```

QUERY PLAN

-----  
Bitmap Heap Scan on r\_pic\_person\_jsonb\_noarray (cost=59.50..6912.05 rows=2000 width=222

) (actual time=0.167..0.170 rows=2 loops=1)

Recheck Cond: (person @> '{"name": "1349018\_name"}'::jsonb)

Heap Blocks: exact=2

-> Bitmap Index Scan on idx\_gin\_jsonb\_all\_noarray (cost=0.00..59.00 rows=2000 width=0) (actual time=0.155..0.155 rows=2 loops=1)

Index Cond: (person @> '{"name": "1349018\_name"}'::jsonb)

Planning time: 0.065 ms

Execution time: 0.196 ms

(7 rows)

**Time: 0.593 ms**

### 3. JSONB的存储和访问效率

在R\_PIC\_PERSON\_JSONB\_USEARRAY上面执行（在200万行的记录里）：

```
explain analyze select * from R_PIC_PERSON_JSONB_USEARRAY where personS @>
'["name":"1349018_name"]';
```

QUERY PLAN

-----  
 Bitmap Heap Scan on r\_pic\_person\_jsonb\_usearray (cost=44.78..424.81 rows=100  
 width=1365

) (actual time=0.099..0.114 rows=2 loops=1)

Recheck Cond: (persons @> '["name": "1349018\_name"]'::jsonb)

Heap Blocks: exact=2

-> Bitmap Index Scan on idx\_gin\_jsonb\_all\_userarray (cost=0.00..44.75 rows=100  
 width

=0) (actual time=0.077..0.077 rows=2 loops=1)

Index Cond: (persons @> '["name": "1349018\_name"]'::jsonb)

Planning time: 0.066 ms

Execution time: 0.140 ms

(7 rows)

**Time: 0.571 ms**

### 3. JSONB的存储和访问效率

下面这个SQL语句，没走索引，针对R\_PIC\_PERSON\_JSONB\_NOARRAY：

```
explain analyze SELECT count(*) FROM R_PIC_PERSON_JSONB_NOARRAY a WHERE (a.person->>'attendanceyear')::integer <1998;

QUERY PLAN
-----
Aggregate  (cost=104167.56..104167.57 rows=1 width=8) (actual time=1356.895..1356.895 rows=1 loops=1)
  -> Seq Scan on r_pic_person_jsonb_noarray a  (cost=0.00..102500.90 rows=666665 width=8) (actual time=0.224..1355.766 rows=1991 loops=1)
    Filter: (((person ->> 'attendanceyear')::text)::integer < 1998)
    Rows Removed by Filter: 1998009
Planning time: 0.052 ms
Execution time: 1356.922 ms
(6 rows)
Time: 1357.409 ms
```

小结：建立JSONB的索引后，对字符串相等匹配会有效，数值类操作无效，这是因为就算取到数值型的jsonb，还得要做转换（如上面的(a.person->>'attendanceyear')::integer <1998）。在上面的例子里，从200万行的数据里查询，使用索引后，从全表扫描的Execution time为1356.922 ms降低到0.196 ms，提升还是巨大的。

1

为何PostgreSQL要支持JSON ?

2

数据库设计中对JSON的3种使用方式

3

JSONB的存储和访问效率

4

PostgreSQL中验证JSON Schema

5

Node. js使用PostgreSQL中的JSON数据

## 4. PostgreSQL中验证JSON Schema

如何让PG验证JSON的Schema？什么是JSON的Schema？参考<http://json-schema.org/> 官方网站。我们期望每次修改jsonb数据的时候，都是符合或者遵循如下一定格式的。

1. 我们先看看，如果不在PG中验证JSON文档的Schema，而是在应用层中（如JAVA）如何做验证（Validate）？

我使用的是<https://github.com/everit-org/json-schema> + <https://github.com/stleary/JSON-java> 库，而没使用：  
<https://github.com/daveclayton/json-schema-validator> + <https://github.com/FasterXML/jackson> 库，后者尝试用  
<http://wilddiary.com/validate-json-against-schema-in-java/> 方法一直没成功。  
故这里还是用<https://github.com/everit-org/json-schema> 方法吧。

[1] <http://blog.sciencenet.cn/home.php?mod=space&uid=419883&do=blog&quickforward=1&id=1009248>

## 4. PostgreSQL中验证JSON Schema

JAVA代码为：

```
public class JsonUtil {  
    public Boolean validate(String jsonInstance){  
        Boolean res = false;  
        try (  
            InputStream inputStream = getClass().getResourceAsStream("personSchema.json")) {  
                JSONObject rawSchema = new JSONObject(new JSONTokener(inputStream));  
                Schema schema = SchemaLoader.load(rawSchema);  
                schema.validate(new JSONObject(jsonInstance)); // throws a ValidationException if this object is invalid  
                res = true;  
            } catch (IOException e) {  
                e.printStackTrace();  
            } catch (JSONException e) {  
                e.printStackTrace();  
            } catch (ValidationException e) {  
                e.printStackTrace();  
            }  
        }  
        return res;  
    }  
}
```

## 4. PostgreSQL中验证JSON Schema

### 2. 接下来我们看看，在PG中能否验证JSON文档的Schema？

<https://github.com/postgrespro/jsquery>

```
cd postgresql-9.6.0
./configure
make world
make install-world
然后createdb之后：
CREATE EXTENSION jsquery;
然后就可以了
```

小结：感觉jsquery还是挺好用的，可以尝试。

```
test=# create table tpic(
id serial,
picUrl character varying(256) default NULL,
persons jsonb NOT NULL,
check(persons @@ '#:(
persons IS STRING AND
name IS STRING AND
phone IS STRING AND
city IS STRING AND
organization IS STRING AND
attendanceYear IS STRING AND
graduationYear IS STRING
)::jsquery)
);
test=# insert into tpic values(1,null,['{"persons":"前1排左1","name": "张晓陆",
"phone": "18910180100","city": "济南","organization": "中科院计算所",
"attendanceYear": "2012","graduationYear": "2016"}']);
```

[1] <http://blog.sciencenet.cn/home.php?mod=space&uid=419883&do=blog&quickforward=1&id=1009248>



1

为何PostgreSQL要支持JSON ?

2

数据库设计中对JSON的3种使用方式

3

JSONB的存储和访问效率

4

PostgreSQL中验证JSON Schema

5

Node. js使用PostgreSQL中的JSON数据

## 4. Node.js使用PostgreSQL中的JSON数据

这里将展示如何在Node.js中构建**RESTful API**的接口，同时使用PostgreSQL中的JSON数据。  
我使用 [Designing a RESTful API With Node and Postgres](#) 里面的例子( Node.js, express-generator, pg-promise , PostgreSQL v9.6, and Bluebird)

设计如下RESTful API:

URLHTTP	Verb	Action
/api/pic/persons	GET	返回该图片中的所有，需要传入参数：图片的id
/api/pic/person/:idx	GET	返回图片中第idx个人
/api/pic/persons	POST	增加一个人
/api/pic/person/:idx	PUT	更新一个人
/api/pic/person/:idx	DELETE	删除一个人

[1] <http://blog.sciencenet.cn/home.php?mod=space&uid=419883&do=blog&quickforward=1&id=1009599>

## 4. Node.js使用PostgreSQL中的JSON数据

在index.js里增加路由:

```
router.get('/api/pic/persons', db.getPicAllPersons);  
router.get('/api/pic/person/:idx', db.getPicSinglePerson);  
router.post('/api/pic/persons', db.createPicPerson);  
router.put('/api/pic/person/:idx', db.updatePicPerson);  
router.delete('/api/pic/person/:idx', db.removePicPerson);
```

## 4. Node.js使用PostgreSQL中的JSON数据

`router.get('/api/pic/persons', db.getPicAllPersons);`

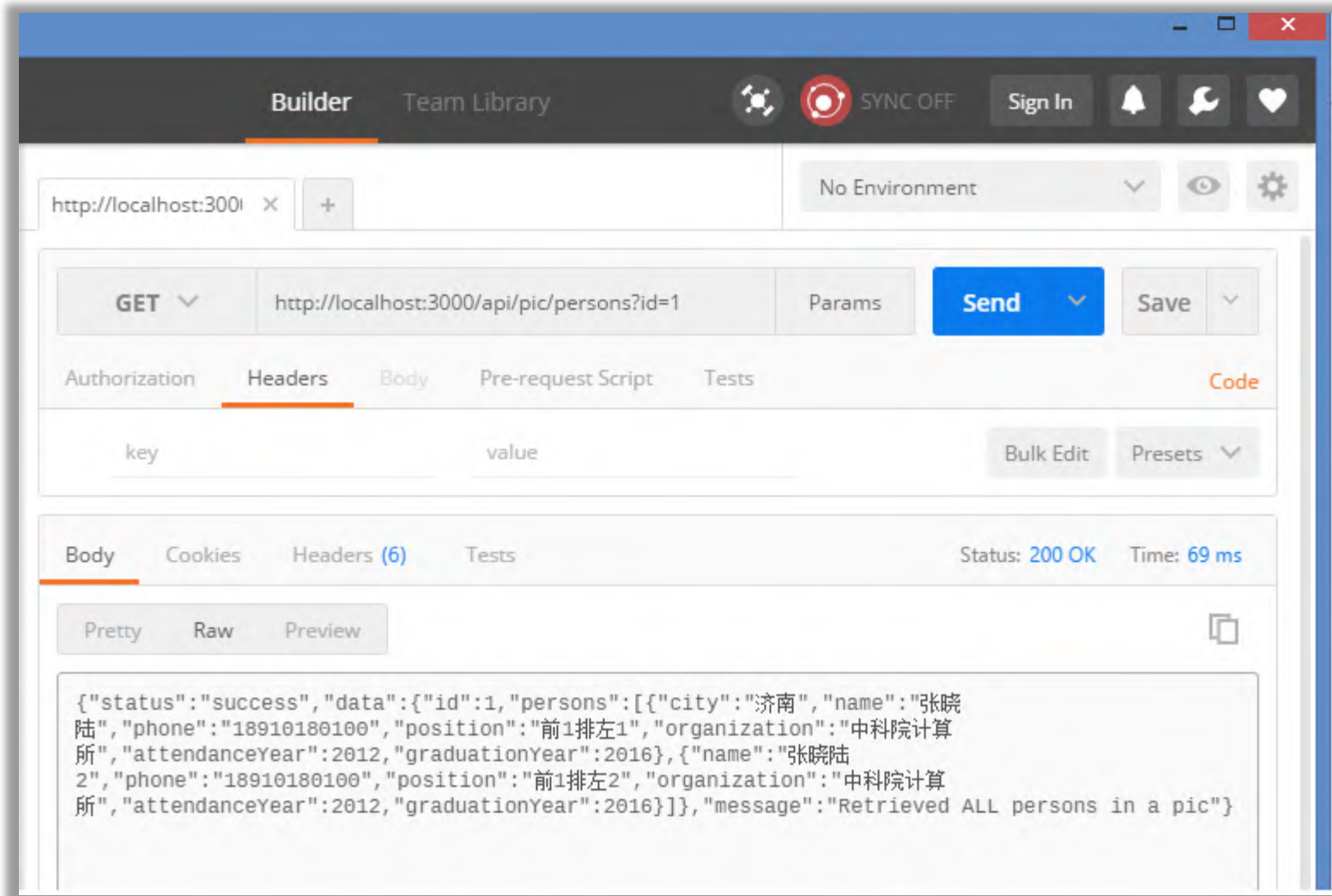
在queries.js里增加处理函数:

```
function getPicAllPersons(req, res, next) {  
  var picID = parseInt(req.query.id);  
  db.one('select * from R_PIC_PERSON_JSONB_USEARRAY where id = $1', picID)  
    .then(function (data) {  
      res.status(200)  
        .json({  
          status: 'success',  
          data: data,  
          message: 'Retrieved ALL persons in a pic'  
        });  
    })  
    .catch(function (err) {  
      return next(err);  
    });  
}
```

在index.js里增加路由:

## 4. Node.js使用PostgreSQL中的JSON数据

上面的req.query.id是指图片的id, 用Postman测试结果如下:



## 4. Node.js使用PostgreSQL中的JSON数据

**/api/pic/person/:idx GET** 返回图片中第idx个人

在queries.js里增加处理函数:

```
function getPicSinglePerson(req, res, next) {
  var personIdx = parseInt(req.params.idx);
  var picID = parseInt(req.query.id);
  db.one('select a.id,a.persons-
>1aspersonfromRPICPERSONJSONBUSEARRAYawherea.id=1aspersonfromRPICPERSONJSONBUSEARRAYawherea.id=2',
[personIdx,picID])
  .then(function (data) {
    res.status(200)
    .json({
      status: 'success',
      data: data,
      message: 'Retrieved ONE PERSON'
    });
  })
  .catch(function (err) {
    return next(err);
  });
}
```

## 4. Node.js使用PostgreSQL中的JSON数据

通过<http://localhost:3000/api/pic/person/1?id=1> 测试一下, 返回结果为:

```
{"status":"success","data":{"id":1,"person":{"name":"张晓陆  
2","phone":"18910180100","position":"前1排左2","organization":"中科院计算  
所","attendanceYear":2012,"graduationYear":2016}},"message":"Retrieved ONE  
PERSON"}
```

## 4. Node.js使用PostgreSQL中的JSON数据

**/api/pic/persons**      **POST**      增加一个人

在queries.js里增加处理函数:

```
function createPicPerson(req, res, next) {var aPersonJson = req.body;
var picID = parseInt(req.query.id);
db.none('update R_PIC_PERSON_JSONB_USEARRAY set persons = persons
|| 1whereid=1whereid=2',[aPersonJson,picID])
.then(function () {
  res.status(200)
  .json({
    status: 'success',
    message: 'Inserted one person'
  });
})
.catch(function (err) {
  return next(err);
});
}
```



在index.js里增加路由:

# 4. Node.js使用PostgreSQL中的JSON数据

用Postman测试结果如下:

