04 非结构存储：用好 JSON 这张牌

前面几讲，我已经带你了解了 MySQL 数据库中常见的 3 种类型：数字类型、字符串类型和日期类型。然而，它们都属于传统关系型设计的范畴。

关系型的结构化存储存在一定的弊端，因为它需要预先定义好所有的列以及列对应的类型。但是业务在发展过程中，或许需要扩展单个列的描述功能，这时，如果能用好 JSON 数据类型，那就能打通关系型和非关系型数据的存储之间的界限，为业务提供更好的架构选择。

当然，很多同学在用 JSON 数据类型时会遇到各种各样的问题，**其中最容易犯的误区就是将类型 JSON 简单理解成字符串类型。** 但当你学完今天的内容之后，会真正认识到 JSON 数据类型的威力，从而在实际工作中更好地存储非结构化的数据。

JSON 数据类型

JSON（JavaScript Object Notation）主要用于互联网应用服务之间的数据交换。MySQL 支持RFC 7159定义的 JSON 规范，主要有**JSON 对象**和**JSON 数组**两种类型。下面就是 JSON 对象，主要用来存储图片的相关信息：

```
{
 "Image": {
   "Width": 800,
   "Height": 600,
   "Title": "View from 15th Floor",
   "Thumbnail": {
     "Url": "http://www.example.com/image/481989943",
     "Height": 125,
     "Width": 100
   },
```

```
  "IDs": [116, 943, 234, 38793]
  }
 }
```

从中你可以看到， JSON 类型可以很好地描述数据的相关内容，比如这张图片的宽度、高度、标题等（这里使用到的类型有整型、字符串类型）。

JSON对象除了支持字符串、整型、日期类型，JSON 内嵌的字段也支持数组类型，如上代码中的 IDs 字段。

另一种 JSON 数据类型是数组类型，如：

```
[
   {
     "precision": "zip",
     "Latitude": 37.7668,
     "Longitude": -122.3959,
     "Address": "",
     "City": "SAN FRANCISCO",
     "State": "CA",
     "Zip": "94107",
     "Country": "US"
   },
   {
     "precision": "zip",
     "Latitude": 37.371991,
     "Longitude": -122.026020,
     "Address": "",
     "City": "SUNNYVALE",
     "State": "CA",
     "Zip": "94085",
```

```
    "Country": "US"
  }
]
```

上面的示例演示的是一个 JSON 数组，其中有 2 个 JSON 对象。

到目前为止，可能很多同学会把 JSON 当作一个很大的字段串类型，从表面上来看，没有错。但本质上，JSON 是一种新的类型，有自己的存储格式，还能在每个对应的字段上创建索引，做特定的优化，这是传统字段串无法实现的。JSON 类型的另一个好处是**无须预定义字段**，字段可以无限扩展。而传统关系型数据库的列都需预先定义，想要扩展需要执行 ALTER TABLE ... ADD COLUMN ... 这样比较重的操作。

需要注意是，JSON 类型是从 MySQL 5.7 版本开始支持的功能，而 8.0 版本解决了更新 JSON 的日志性能瓶颈。如果要在生产环境中使用 JSON 数据类型，强烈推荐使用 MySQL 8.0 版本。

讲到这儿，你已经对 JSON 类型的基本概念有所了解了，接下来，我们进入实战环节：如何在业务中用好JSON类型?

业务表结构设计实战

用户登录设计

在数据库中，**JSON 类型比较适合存储一些修改较少、相对静态的数据**，比如用户登录信息的存储如下：

```
DROP TABLE IF EXISTS UserLogin;
CREATE TABLE UserLogin (
    userId BIGINT NOT NULL,
    loginInfo JSON,
    PRIMARY KEY(userId)
);
```

由于当前业务的登录方式越来越多样化，如同一账户支持手机、微信、QQ 账号登录，所以这里可以用 JSON 类型存储登录的信息。

接着，插入下面的数据：

```
SET @a = '
{
        "cellphone" : "13918888888",
        "wxchat" : "破产码农",
    "QQ" : "82946772"
}
';
INSERT INTO UserLogin VALUES (1,@a);
SET @b = '
{
        "cellphone" : "15026888888"
}
';
INSERT INTO UserLogin VALUES (2,@b);
```

从上面的例子中可以看到，用户 1 登录有三种方式：手机验证码登录、微信登录、QQ 登录，而用户 2 只有手机验证码登录。

而如果不采用 JSON 数据类型，就要用下面的方式建表：

```
CREATE TABLE UserLogin (
    userId              BIGINT NOT NULL,
    cellphone   VARCHAR(255),
    wechat              VARCHAR(255)
    QQ                  VARCHAR(255),
    PRIMARY KEY(userId)
);
```

可以看到，虽然用传统关系型的方式也可以完成相关数据的存储，但是存在两个问题。

- 有些列可能是比较稀疏的，一些列可能大部分都是 NULL 值；

- 如果要新增一种登录类型，如微博登录，则需要添加新列，而 JSON 类型无此烦恼。

因为支持了新的JSON类型，MySQL 配套提供了丰富的 JSON 字段处理函数，用于方便地操作 JSON 数据，具体可以见 MySQL 官方文档。

其中，最常见的就是函数 JSON_EXTRACT，它用来从 JSON 数据中提取所需要的字段内容，如下面的这条 SQL 语句就查询用户的手机和微信信息。

```
SELECT
    userId,
    JSON_UNQUOTE(JSON_EXTRACT(loginInfo,"$.cellphone")) cellphone,
    JSON_UNQUOTE(JSON_EXTRACT(loginInfo,"$.wxchat")) wxchat
FROM UserLogin;
+--------+-------------+--------------+
| userId | cellphone   | wxchat       |
+--------+-------------+--------------+
|      1 | 13918888888 | 破产码农     |
|      2 | 15026888888 | NULL         |
+--------+-------------+--------------+
2 rows in set (0.01 sec)
```

当然了，每次写 JSON_EXTRACT、JSON_UNQUOTE 非常麻烦，MySQL 还提供了 ->> 表达式，和上述 SQL 效果完全一样：

```
SELECT
    userId,
    loginInfo->>"$.cellphone" cellphone,
    loginInfo->>"$.wxchat" wxchat
FROM UserLogin;
```

当 JSON 数据量非常大，用户希望对 JSON 数据进行有效检索时，可以利用 MySQL 的**函数索引**功能对 JSON 中的某个字段进行索引。

比如在上面的用户登录示例中，假设用户必须绑定唯一手机号，且希望未来能用手机号码进行用户检索时，可以创建下面的索引：

```
ALTER TABLE UserLogin ADD COLUMN cellphone VARCHAR(255) AS (loginInfo->>"$.cellphon

ALTER TABLE UserLogin ADD UNIQUE INDEX idx_cellphone(cellphone);
```

上述 SQL 首先创建了一个虚拟列 cellphone，这个列是由函数 loginInfo->>"$.cellphone" 计算得到的。然后在这个虚拟列上创建一个唯一索引 idx_cellphone。这时再通过虚拟列 cellphone 进行查询，就可以看到优化器会使用到新创建的 idx_cellphone 索引：

```
EXPLAIN SELECT  *  FROM UserLogin

WHERE cellphone = '13918888888'\G

*************************** 1. row ***************************

           id: 1

  select_type: SIMPLE

        table: UserLogin

   partitions: NULL

         type: const

possible_keys: idx_cellphone

          key: idx_cellphone

      key_len: 1023

          ref: const

         rows: 1

     filtered: 100.00

        Extra: NULL

1 row in set, 1 warning (0.00 sec)
```

当然，我们可以在一开始创建表的时候，就完成虚拟列及函数索引的创建。如下表创建的列 cellphone 对应的就是 JSON 中的内容，是个虚拟列；uk_idx_cellphone 就是在虚拟列 cellphone 上所创建的索引。

```
CREATE TABLE UserLogin (
```

```
    userId BIGINT,
    loginInfo JSON,
    cellphone VARCHAR(255) AS (loginInfo->>"$.cellphone"),
    PRIMARY KEY(userId),
    UNIQUE KEY uk_idx_cellphone(cellphone)
);
```

用户画像设计

某些业务需要做用户画像（也就是对用户打标签），然后根据用户的标签，通过数据挖掘技术，进行相应的产品推荐。比如：

- 在电商行业中，根据用户的穿搭喜好，推荐相应的商品；
- 在音乐行业中，根据用户喜欢的音乐风格和常听的歌手，推荐相应的歌曲；
- 在金融行业，根据用户的风险喜好和投资经验，推荐相应的理财产品。

在这，我强烈推荐你用 JSON 类型在数据库中存储用户画像信息，并结合 JSON 数组类型和多值索引的特点进行高效查询。假设有张画像定义表：

```
CREATE TABLE Tags (
    tagId bigint auto_increment,
    tagName varchar(255) NOT NULL,
    primary key(tagId)
);
SELECT * FROM Tags;
+-------+--------------+
| tagId | tagName      |
+-------+--------------+
|     1 | 70后         |
|     2 | 80后         |
|     3 | 90后         |
|     4 | 00后         |
```

```
|     5 | 爱运动       |
|     6 | 高学历       |
|     7 | 小资         |
|     8 | 有房         |
|     9 | 有车         |
|    10 | 常看电影     |
|    11 | 爱网购       |
|    12 | 爱外卖       |
+-------+--------------+
```

可以看到，表 Tags 是一张画像定义表，用于描述当前定义有多少个标签，接着给每个用户打标签，比如用户 David，他的标签是 80 后、高学历、小资、有房、常看电影；用户 Tom，90 后、常看电影、爱外卖。

若不用 JSON 数据类型进行标签存储，通常会将用户标签通过字符串，加上分割符的方式，在一个字段中存取用户所有的标签：

```
+-------+----------------------------------------+
|用户    |标签                                    |
+-------+----------------------------------------+
|David  |80后 ； 高学历 ； 小资 ； 有房 ；常看电影   |
|Tom    |90后 ；常看电影 ； 爱外卖                 |
+-------+----------------------------------------+
```

这样做的缺点是： 不好搜索特定画像的用户，另外分隔符也是一种自我约定，在数据库中其实可以任意存储其他数据，最终产生脏数据。

用 JSON 数据类型就能很好解决这个问题：

```
DROP TABLE IF EXISTS UserTag;
CREATE TABLE UserTag (
    userId bigint NOT NULL,
```

```
    userTags JSON,
    PRIMARY KEY (userId)
);
INSERT INTO UserTag VALUES (1,'[2,6,8,10]');
INSERT INTO UserTag VALUES (2,'[3,10,12]');
```

其中，userTags 存储的标签就是表 Tags 已定义的那些标签值，只是使用 JSON 数组类型进行存储。

MySQL 8.0.17 版本开始支持 Multi-Valued Indexes，用于在 JSON 数组上创建索引，并通过函数 member of、json_contains、json_overlaps 来快速检索索引数据。所以你可以在表 UserTag 上创建 Multi-Valued Indexes：

```
ALTER TABLE UserTag
ADD INDEX idx_user_tags ((cast((userTags->"$") as unsigned array)));
```

如果想要查询用户画像为常看电影的用户，可以使用函数 MEMBER OF：

```
EXPLAIN SELECT * FROM UserTag
WHERE 10 MEMBER OF(userTags->"$")\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: UserTag
   partitions: NULL
         type: ref
possible_keys: idx_user_tags
          key: idx_user_tags
      key_len: 9
          ref: const
         rows: 1
     filtered: 100.00
```

```
        Extra: Using where
1 row in set, 1 warning (0.00 sec)
SELECT * FROM UserTag
WHERE 10 MEMBER OF(userTags->"$");
+--------+--------------+
| userId | userTags     |
+--------+--------------+
|      1 | [2, 6, 8, 10] |
|      2 | [3, 10, 12]   |
+--------+--------------+
2 rows in set (0.00 sec)
```

如果想要查询画像为 80 后，且常看电影的用户，可以使用函数 JSON_CONTAINS：

```
EXPLAIN SELECT * FROM UserTag
WHERE JSON_CONTAINS(userTags->"$", '[2,10]')\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: UserTag
   partitions: NULL
         type: range
possible_keys: idx_user_tags
          key: idx_user_tags
      key_len: 9
          ref: NULL
         rows: 3
     filtered: 100.00
        Extra: Using where
```

```
1 row in set, 1 warning (0.00 sec)

SELECT * FROM UserTag
WHERE JSON_CONTAINS(userTags->"$", '[2,10]');
+--------+--------------+
| userId | userTags     |
+--------+--------------+
|      1 | [2, 6, 8, 10] |
+--------+--------------+
1 row in set (0.00 sec)
```

如果想要查询画像为 80 后、90 后，且常看电影的用户，则可以使用函数 JSON_OVERLAP：

```
EXPLAIN SELECT * FROM UserTag
WHERE JSON_OVERLAPS(userTags->"$", '[2,3,10]')\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: UserTag
   partitions: NULL
         type: range
possible_keys: idx_user_tags
          key: idx_user_tags
      key_len: 9
          ref: NULL
         rows: 4
     filtered: 100.00
        Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

```
SELECT * FROM UserTag

WHERE JSON_OVERLAPS(userTags->"$", '[2,3,10]');

+--------+---------------+

| userId | userTags      |

+--------+---------------+

|      1 | [2, 6, 8, 10] |

|      2 | [3, 10, 12]   |

+--------+---------------+

2 rows in set (0.01 sec)
```

总结

JSON 类型是 MySQL 5.7 版本新增的数据类型，用好 JSON 数据类型可以有效解决很多业务中实际问题。最后，我总结下今天的重点内容：

- 使用 JSON 数据类型，推荐用 MySQL 8.0.17 以上的版本，性能更好，同时也支持 Multi-Valued Indexes；
- JSON 数据类型的好处是无须预先定义列，数据本身就具有很好的描述性；
- 不要将有明显关系型的数据用 JSON 存储，如用户余额、用户姓名、用户身份证等，这些都是每个用户必须包含的数据；
- JSON 数据类型推荐使用在不经常更新的静态数据存储。

上一页 下一页