8.1 JSON

对于关系型数据库来说，非结构化数据的存储一直是个难点。JSON数据类型打通了关系型和非关系型数据的存储界限，为业务系统提供了更好的架构选择。

8.1.1 JSON 和 NoSQL

关系型数据库存储数据时需要在数据表中预先定义好所有的列以及每列的数据类型。当业务逻辑发生变化时，数据结构往往也会随之进行更改，需要增加、删除或者重新定义一部分数据列。严格的数据定义保证了业务模型描述的准确性和一致性，但同时也限制了软件功能的扩展。

针对关系型数据库横向扩展性较差的弱点，NoSQL数据库采用了no schema方式，数据存取不再受字段的限制，极大地提高了数据库的横向扩展性。例如，文档型数据库MongoDB采用BSON（二进制JSON）格式进行数据存储，取得了良好的扩展性和读写性能。

采用JSON格式存放的数据在业务功能扩展时无须在数据表中再增加字段，避免了数据库结构变化引起的程序架构改动，数据层和业务层的耦合度也降低了，开发人员可以把精力集中在程序代码的编写上。

达梦数据库也支持对JSON的数据处理，最新版的达梦数据库虽然增加了很多JSON函数，但使用起来并不方便，目前正在完善中。因此，这里只做简单介绍。

8.1.2 JSON 和 XML

提到JSON就不能不说一下XML，这二者都是目前最流行的数据交换格式。JSON（JavaScript Object Notation）是一种轻量级的数据交换格式。而XML（Extensible Markup Language，扩展标记语言）则是一种“重量级”的数据交换格式。

XML的样式跟HTML（Hyper Text Markup Language，超文本标记语言）很相似，很容易让人误以为XML就是HTML的扩展。其实这两种标记语言用途是完全不一样的：HTML是用来展示数据的，目前的网页展示都用的是HTML；XML是用来描述、存储数据的。

XML格式统一、语法要求严格，标准化程度和可读性都非常高。例如：

```
<person>
    <name>张三</name>
    <age>20</age>
    <job>programmer</job>
    <city>南京</city>
    <hobbies>
        <hobby>游泳</hobby>
        <hobby>登山</hobby>
        <hobby>读书</hobby>
        <hobby>旅游</hobby>
```

```
    </hobbies>
</person>
```

写成JSON格式是这样的：

```
{  name:"张三",
    age:20
    job:"programmer",
    city:"南京",
    hobbies:["游泳","登山","读书","旅游"]
}
```

这样一比较，JSON的优点也就一目了然：结构简洁，解析起来更快；占用的存储空间少，网络传输也更快。

正因为有了这些优点，JSON成为了目前最受欢迎的数据交换格式，广泛应用于各个领域。

8.1.3 JSON 数据存取

JSON数据在达梦数据库中以字符串形式存储。用户在创建数据表时需要对JSON数据字段使用约束IS JSON进行验证。

例如，创建学生成绩表：

```
SQL> create table json_t(stu_id int cluster primary key,score varchar(400)
check(score is json));
操作已执行
```

插入测试数据：

```
SQL> select * from json_t;
行号        stu_id      score
1           1           {"语文":80,"数学":76,"外语":84,"物理":65,"化学":92}
2           2           {"语文":68,"数学":96,"外语":80,"物理":95,"化学":92}
3           3           {"语文":77,"数学":41,"外语":84,"历史":90,"地理":72}
4           4           {"语文":97,"数学":81,"外语":94,"历史":80,"地理":92}
```

查询JSON数据的函数json_value（col_name, json_name），两个参数，分别为列名和要查询的JSON路径表达式。

例如：

```
SQL> select stu_id,json_value(score,'$."数学"') math from json_t;
行号        stu_id      math
1           1           76
2           2           96
```

```
3          3          41
4          4          81
```

如果JSON路径中的属性不存在，则返回NULL：

```
SQL> select stu_id,json_value(score,'$."化学"') che from json_t;
行号        stu_id       che
1          1            92
2          2            92
3          3            NULL
4          4            NULL
```

筛选条件中也可以使用JSON_VALUE函数：

```
SQL> select * from json_t where json_value(score,'$."物理"')>80;
行号        stu_id   score
1          2        {"语文":68,"数学":96,"外语":80,"物理":95,"化学":92}
```

JSON类型字段也可以存储数组：

```
SQL> insert into json_t values(5,'["语文","数学","外语",100]');
影响行数 1
SQL> select * from json_t;
行号        stu_id   score
1          1        {"语文":80,"数学":76,"外语":84,"物理":65,"化学":92}
2          2        {"语文":68,"数学":96,"外语":80,"物理":95,"化学":92}
3          3        {"语文":77,"数学":41,"外语":84,"历史":90,"地理":72}
4          4        {"语文":97,"数学":81,"外语":94,"历史":80,"地理":92}
5          5        ["语文","数学","外语",100]
```

对数组的访问使用下标（从0开始）：

```
SQL> select stu_id,json_value(score,'$[0]') from json_t;
行号        stu_id       json_value (score,'$[0]')
1          1            NULL
2          2            NULL
3          3            NULL
4          4            NULL
5          5            语文
```

查询使用JSON数据类型的字段的系统视图如下。

USER_JSON_COLUMNS，显示当前用户拥有的JSON数据信息。

ALL_JSON_COLUMNS，显示当前用户有权访问的JSON数据信息。