02 关系模型

我们已经知道，关系数据库是建立在关系模型上的。而关系模型本质上就是若干个存储数据的二维表，可以把它们看作很多Excel表。

表的每一行称为记录（Record），记录是一个逻辑意义上的数据。

表的每一列称为字段（Column），同一个表的每一行记录都拥有相同的若干字段。

字段定义了数据类型（整型、浮点型、字符串、日期等），以及是否允许为`NULL`。注意`NULL`表示字段数据不存在。一个整型字段如果为`NULL`不表示它的值为`0`，同样的，一个字符串型字段为`NULL`也不表示它的值为空串`''`。

通常情况下，字段应该避免允许为NULL。不允许为NULL可以简化查询条件，加快查询速度，也利于应用程序读取数据后无需判断是否为NULL。

和Excel表有所不同的是，关系数据库的表和表之间需要建立“一对多”，“多对一”和“一对一”的关系，这样才能够按照应用程序的逻辑来组织和存储数据。

例如，一个班级表：

ID	名称	班主任
201	二年级一班	王老师
202	二年级二班	李老师

每一行对应着一个班级，而一个班级对应着多个学生，所以班级表和学生表的关系就是“一对多”：

ID	姓名	班级ID	性别	年龄
1	小明	201	M	9
2	小红	202	F	8
3	小军	202	M	8
4	小白	201	F	9

反过来，如果我们先在学生表中定位了一行记录，例如`ID=1`的小明，要确定他的班级，只需要根据他的“班级ID”对应的值`201`找到班级表中`ID=201`的记录，即二年级一班。所以，学生表和班级表是“多对一”的关系。

如果我们把班级表分拆得细一点，例如，单独创建一个教师表：

ID	名称	年龄
A1	王老师	26
A2	张老师	39
A3	李老师	32
A4	赵老师	27

班级表只存储教师ID：

ID	名称	班主任ID
201	二年级一班	A1
202	二年级二班	A3

这样，一个班级总是对应一个教师，班级表和教师表就是“一对一”关系。

在关系数据库中，关系是通过*主键*和*外键*来维护的。我们在后面会分别深入讲解。

主键

在关系数据库中，一张表中的每一行数据被称为一条记录。一条记录就是由多个字段组成的。例如，`students`表的两行记录：

ID	CLASS_ID	NAME	GENDER	SCORE
1	1	小明	M	90
2	1	小红	F	95

每一条记录都包含若干定义好的字段。同一个表的所有记录都有相同的字段定义。

对于关系表，有个很重要的约束，就是任意两条记录不能重复。不能重复不是指两条记录不完全相同，而是指能够通过某个字段唯一区分出不同的记录，这个字段被称为*主键*。

例如，假设我们把`name`字段作为主键，那么通过名字`小明`或`小红`就能唯一确定一条记录。但是，这么设定，就没法存储同名的同学了，因为插入相同主键的两条记录是不被允许的。

对主键的要求，最关键的一点是：记录一旦插入到表中，主键最好不要再修改，因为主键是用来唯一定位记录的，修改了主键，会造成一系列的影响。

由于主键的作用十分重要，如何选取主键会对业务开发产生重要影响。如果我们以学生的身份证号作为主键，似乎能唯一定位记录。然而，身份证号也是一种业务场景，如果身份证号升位了，或者需要变更，作为主键，不得不修改的时候，就会对业务产生严重影响。

所以，选取主键的一个基本原则是：不使用任何业务相关的字段作为主键。

因此，身份证号、手机号、邮箱地址这些看上去可以唯一的字段，均*不可*用作主键。

作为主键最好是完全业务无关的字段，我们一般把这个字段命名为`id`。常见的可作为`id`字段的类型有：

1. 自增整数类型：数据库会在插入数据时自动为每一条记录分配一个自增整数，这样我们就完全不用担心主键重复，也不用自己预先生成主键；
2. 全局唯一GUID类型：使用一种全局唯一的字符串作为主键，类似`8f55d96b-8acc-4636-8cb8-76bf8abc2f57`。GUID算法通过网卡MAC

地址、时间戳和随机数保证任意计算机在任意时间生成的字符串都是不同的，大部分编程语言都内置了GUID算法，可以自己预算出主键。

对于大部分应用来说，通常自增类型的主键就能满足需求。我们在`students`表中定义的主键也是`BIGINT NOT NULL AUTO_INCREMENT`类型。

如果使用INT自增类型，那么当一张表的记录数超过2147483647（约21亿）时，会达到上限而出错。使用BIGINT自增类型则可以最多约922亿亿条记录。

联合主键

关系数据库实际上还允许通过多个字段唯一标识记录，即两个或更多的字段都设置为主键，这种主键被称为联合主键。

对于联合主键，允许一列有重复，只要不是所有主键列都重复即可：

ID_NUM	ID_TYPE	OTHER COLUMNS...
1	A	...
2	A	...
2	B	...

如果我们把上述表的`id_num`和`id_type`这两列作为联合主键，那么上面的3条记录都是允许的，因为没有两列主键组合起来是相同的。

没有必要的情况下，我们尽量不使用联合主键，因为它给关系表带来了复杂度的上升。

小结

- 主键是关系表中记录的唯一标识。主键的选取非常重要：主键不要带有业务含义，而应该使用BIGINT自增或者GUID类型。主键也不应该允许`NULL`。
- 可以使用多个列作为联合主键，但联合主键并不常用。

外键

当我们用主键唯一标识记录时，我们就可以在`students`表中确定任意一个学生的记录：

ID	NAME	OTHER COLUMNS...
1	小明	...
2	小红	...

我们还可以在`classes`表中确定任意一个班级记录：

ID	NAME	OTHER COLUMNS...
1	一班	...
2	二班	...

但是我们如何确定`students`表的一条记录，例如，`id=1`的小明，属于哪个班级呢？

由于一个班级可以有多个学生，在关系模型中，这两个表的关系可以称为“一对多”，即一个`classes`的记录可以对应多个`students`表的记录。

为了表达这种一对多的关系，我们需要在`students`表中加入一列`class_id`，让它的值与`classes`表的某条记录相对应：

ID	CLASS_ID	NAME	OTHER COLUMNS...
1	1	小明	...
2	1	小红	...
5	2	小白	...

这样，我们就可以根据`class_id`这个列直接定位出一个`students`表的记录应该对应到`classes`的哪条记录。

例如：

- 小明的`class_id`是`1`，因此，对应的`classes`表的记录是`id=1`的一班；
- 小红的`class_id`是`1`，因此，对应的`classes`表的记录是`id=1`的一班；
- 小白的`class_id`是`2`，因此，对应的`classes`表的记录是`id=2`的二班。

在`students`表中，通过`class_id`的字段，可以把数据与另一张表关联起来，这种列称为`外键`。

外键并不是通过列名实现的，而是通过定义外键约束实现的：

```
ALTER TABLE students
ADD CONSTRAINT fk_class_id
FOREIGN KEY (class_id)
REFERENCES classes (id);
```

其中，外键约束的名称`fk_class_id`可以任意，`FOREIGN KEY (class_id)`指定了`class_id`作为外键，`REFERENCES classes (id)`指定了这个外键将关联到`classes`表的`id`列（即`classes`表的主键）。

通过定义外键约束，关系数据库可以保证无法插入无效的数据。即如果`classes`表不存在`id=99`的记录，`students`表就无法插入`class_id=99`的记录。

由于外键约束会降低数据库的性能，大部分互联网应用程序为了追求速度，并不设置外键约束，而是仅靠应用程序自身来保证逻辑的正确性。这种情况下，`class_id`仅仅是一个普通的列，只是它起到了外键的作用而已。

要删除一个外键约束，也是通过`ALTER TABLE`实现的：

```
ALTER TABLE students
DROP FOREIGN KEY fk_class_id;
```

注意：删除外键约束并没有删除外键这一列。删除列是通过`DROP COLUMN ...`实现的。

多对多

通过一个表的外键关联到另一个表，我们可以定义出一对多关系。有些时候，还需要定义“多对多”关系。例如，一个老师可以对应多个班级，一个班级也可以对应多个老师，因此，班级表和老师表存在多对多关系。

多对多关系实际上是通过两个一对多关系实现的，即通过一个中间表，关联两个一对多关系，就形成了多对多关系：

`teachers`表：

ID	NAME
1	张老师
2	王老师
3	李老师
4	赵老师

`classes`表：

ID	NAME
1	一班
2	二班

中间表`teacher_class`关联两个一对多关系：

ID	TEACHER_ID	CLASS_ID
1	1	1
2	1	2
3	2	1
4	2	2
5	3	1
6	4	2

通过中间表`teacher_class`可知`teachers`到`classes`的关系：

- `id=1`的张老师对应`id=1,2`的一班和二班；
- `id=2`的王老师对应`id=1,2`的一班和二班；
- `id=3`的李老师对应`id=1`的一班；
- `id=4`的赵老师对应`id=2`的二班。

同理可知`classes`到`teachers`的关系：

- `id=1`的一班对应`id=1,2,3`的张老师、王老师和李老师；
- `id=2`的二班对应`id=1,2,4`的张老师、王老师和赵老师；

因此，通过中间表，我们就定义了一个“多对多”关系。

一对一

一对一关系是指，一个表的记录对应到另一个表的唯一一个记录。

例如，`students`表的每个学生可以有自己的联系方式，如果把联系方式存入另一个表`contacts`，我们就可以得到一个“一对一”关系：

ID	STUDENT_ID	MOBILE
1	1	135xxxx6300
2	2	138xxxx2209
3	5	139xxxx8086

有细心的童鞋会问，既然是一对一关系，那为啥不给`students`表增加一个`mobile`列，这样就能合二为一了？

如果业务允许，完全可以把两个表合为一个表。但是，有些时候，如果某个学生没有手机号，那么，`contacts`表就不存在对应的记录。实际上，一对一关系准确地说，是`contacts`表一对一对应`students`表。

还有一些应用会把一个大表拆成两个一对一的表，目的是把经常读取和不经常读取的字段分开，以获得更高的性能。例如，把一个大的用户表分拆为用户基本信息表`user_info`和用户详细信息表`user_profiles`，大部分时候，只需要查询`user_info`表，并不需要查询`user_profiles`表，这样就提高了查询速度。

小结

- 关系数据库通过外键可以实现一对多、多对多和一对一的关系。外键既可以通过数据库来约束，也可以不设置约束，仅依靠应用程序的逻辑来保证。

索引

在关系数据库中，如果有上万甚至上亿条记录，在查找记录的时候，想要获得非常快的速度，就需要使用索引。

索引是关系数据库中对某一列或多个列的值进行预排序的数据结构。通过使用索引，可以让数据库系统不必扫描整个表，而是直接定位到符合条件的记录，这样就大大加快了查询速度。

例如，对于`students`表：

ID	CLASS_ID	NAME	GENDER	SCORE
1	1	小明	M	90
2	1	小红	F	95
3	1	小军	M	88

如果要经常根据`score`列进行查询，就可以对`score`列创建索引：

```
ALTER TABLE students
ADD INDEX idx_score (score);
```

使用`ADD INDEX idx_score (score)`就创建了一个名称为`idx_score`，使用列`score`的索引。索引名称是任意的，索引如果有多列，可以在括号里依次写上，例如：

```
ALTER TABLE students
ADD INDEX idx_name_score (name, score);
```

索引的效率取决于索引列的值是否散列，即该列的值如果越互不相同，那么索引效率越高。反过来，如果记录的列存在大量相同的值，例如`gender`列，大约一半的记录值是`M`，另一半是`F`，因此，对该列创建索引就没有意义。

可以对一张表创建多个索引。索引的优点是提高了查询效率，缺点是在插入、更新和删除记录时，需要同时修改索引，因此，索引越多，插入、更新和删除记录的速度就越慢。

对于主键，关系数据库会自动对其创建主键索引。使用主键索引的效率是最高的，因为主键会保证绝对唯一。

唯一索引

在设计关系数据表的时候，看上去唯一的列，例如身份证号、邮箱地址等，因为他们具有业务含义，因此不宜作为主键。

但是，这些列根据业务要求，又具有唯一性约束：即不能出现两条记录存储了同一个身份证号。这个时候，就可以给该列添加一个唯一索引。例如，我们假设`students`表的`name`不能重复：

```
ALTER TABLE students
ADD UNIQUE INDEX uni_name (name);
```

通过`UNIQUE`关键字我们就添加了一个唯一索引。

也可以只对某一列添加一个唯一约束而不创建唯一索引：

```
ALTER TABLE students
ADD CONSTRAINT uni_name UNIQUE (name);
```

这种情况下，`name`列没有索引，但仍然具有唯一性保证。

无论是否创建索引，对于用户和应用程序来说，使用关系数据库不会有任何区别。这里的意思是说，当我们在数据库中查询时，如果有相应的索引可用，数据库系统就会自动使用索引来提高查询效率，如果没有索引，查询也能正常执行，只是速度会变慢。因此，索引可以在使用数据库的过程中逐步优化。

小结

- 通过对数据库表创建索引，可以提高查询速度。
- 通过创建唯一索引，可以保证某一列的值具有唯一性。
- 数据库索引对于用户和应用程序来说都是透明的。