第 5 章 Drupal 的数据库层

老葛的 Drupal 培训班 http://zhupou.cn

Drupal 的 正常工作依赖于数据库。在 Drupal 内部，在你的代码与数据库之间存在着一个轻量级的数据库抽象层。在本章中，你将学习这一数据库抽象层是如何工作的， 以及如何使用它。你将看到如何通过模块来修改查询语句。接着,你将看到如何;连接其它的数据库（比如一个遗留数据库）。最后，你将学习 Drupal 的模式 API,以在你模块的.install 文件中包含数据库表的创建和更新语句。

定义数据库参数

在建立数据库连接时，通过查看你站点的 settings.php 文件，Drupal 就会知道需要连接哪个数据库以及所用的用户名和密码。这个文 件一般位于 sites/example.com/settings.php 或者 sites/default/settings.php。定义数据库连接的代码，如下所示：

$db_url = 'mysql://username:password@localhost/databasename';

这个例子中使用的是 MySQL 数据库。使用 PostgreSQL 的用户需要将前缀“mysql”替换为“pgsql”。显然，这里使用的用户名和密码对于 你的数据库来说必须是有效的。它们是数据库的有效证件，但不是 Drupal 的，在你使用数据库工具建立数据库帐号时就可创建它们(用户名和密码)。 Drupal 的安装器会向你询问用户名和密码（如果没有预先设置的话），这样它就会为你构建 settings.php 文件中的$db_url 字符串。

老葛的 Drupal 培训班 http://zhupou.cn

理解数据库抽象层

老葛的 Drupal 培训班 http://zhupou.cn

使 用一个数据库抽象层 API 时，你可能感觉不到它的好，直到有一天你决定不再使用它的时候，你才能发现它的全部优点。你是否曾经遇到过这样的项目，它需要修 改数据库系统的代码，你花费了大量的时间，通过仔细的审查每段代码，来将它们改为特定数据库的函数和查询？有了数据库抽象层，你就不需要再考虑不同数据库 系统之间函数名的细微差别，只要你使用的是符合 ANSI SQL 的语句，那么你就不再需要为不同的数据库编写单独的查询语句了。举例来说，在 Drupal 中没有直接使用 mysql_query()或者 pg_query(),而是使用的 db_query(),这样就将业务层和数据库层隔离了开来。

Drupal 的数据库层是轻量级的，它主要用于两个目的。第一个目的是使你的代码不会绑定在特定的数据库上。第二个目的是清理用户向查询语句中提交的数据，以阻止 SQL 注入攻击。这一层是建立在以下原理之上的：使用 sql 比重新学习一门新的抽象层的语言更方便。

Drupal 还有一个模式 API，它允许你以一种通用的方式向 Drupal 描述你的数据库模式（也就是，你将使用哪些表和字段），然后让 Drupal 将其翻译成所用数据库的特定 sql 语句。当我们学习.install 文件时，将会对其进行详细的讨论。

通过检查你的 settings.php 文件内部的变量$db_url，Drupal 来判定要连接的数据库的类型。例如，如果

$db_url 的开始部分为 “mysql”，那么 Drupal 将包含进来 includes/database.mysql.inc. 如果$db_url 的开始部分为 “pgsql”，那么 Drupal 将包含进来 includes/database.pgsql.inc.图 5-1 给出了这 一机制。

作为一个例子，让我们比较一下 db_fetch_object()在 MySQL 和 PostgreSQL 抽象层中不同之处：

```
// From database.mysqli.inc.
function db_fetch_object($result) {
    if ($result) {
        return mysql_fetch_object($result);
    }
}
// From database.pgsql.inc.
function db_fetch_object($result) {
    if ($result) {
        return pg_fetch_object($result);
    }
}
```

如果你所用的数据库还未被支持，你可以通过为你的数据库实现相应的包装函数来建立你自己的数据库驱动器。更多详细，可参看本章最后部分的”编写你自己的数据库驱动器”。

图 5-1 *通过检查变量$db_url，Drupal 判定需要包含进来哪个数据库文件。*

连接到数据库

老葛的 Drupal 培训班 http://zhupou.cn

作为 Drupal 的正常的引导指令流程的一部分，Drupal 将会自动的建立数据库连接，所以你不需要为此担心。

如果你需要在 Drupal 外部使用数据库连接（比如，你在编写一个单独的 PHP 脚本或者有段处于 Drupal 之外的 PHP 代码，它们需要访问 Drupal 的数据库），那么可以使用下面的方式。

```
// Make Drupal PHP's current directory.
chdir('/full/path/to/your/drupal/installation');

// Bootstrap Drupal up through the database phase.
include_once('./includes/bootstrap.inc');
drupal_bootstrap(DRUPAL_BOOTSTRAP_DATABASE);

// Now you can run queries using db_query().
$result = db_query('SELECT title FROM {node}');
...
```

警告 在 Drupal 中，通常会在 sites 目录下配置多个文件夹，这样站点从测试迁移到线上时就不用修改数据库密码信息了。例如，对于测试数据库服务器，你可以使用 sites/staging.example.com/settings.php 文件来放置数据库的密码信息，而对于在线的数据库服务器，你可以使用 ites/www.example.com/settings.php 文件。在建立本节所示的连接时，因为这里没有涉及到 HTTP 请求，所以 Drupal 总是使用 sites/default/settings.php 文件。

执行简单的查询

老葛的 Drupal 培训班 http://zhupou.cn

Drupal 的函数 db_query()是用来为已建立的数据库连接执行查询语句的。这些查询语句包括 SELECT, INSERT, UPDATE, 和 DELETE。

当你编写 SQL 语句的时候，你需要注意一些特定于 Drupal 的语法。首先，表名应放在花括号之间，这样以来，在需要的情况下，就可以为表名添加前缀了， 从而保证表名的唯一性。虚拟主机托管商常常会限制了用户的数据库个数，而这一约定则可以让用户在已有的数据库上安装 Drupal，通过在他们的 settings.php 文件中声明数据库前缀来避免表名的冲突。下面是一个简单查询的例子，用来取回角色 2 的名字：

```
$result = db_query('SELECT name FROM {role} WHERE rid = %d', 2);
```

注意，占位符%d 的使用。在 Drupal 中，查询语句通常会使用占位符，而实际的值则作为参数跟在后面。占位符%d 将被后面参数值（在这里就是 2）自动的替换掉。占位符越多，那么参数就会越多，两者是相

对应的：

db_query('SELECT name FROM {role} WHERE rid > %d AND rid != %d', 1, 7);

在数据库中执行时，前面的一行将转化为如下形式：

SELECT FROM role WHERE rid > 1 and rid != 7

用户提交的数据应该作为单独的参数传入，这样这些值就可以被清理，从而阻止 SQL 注入攻击。Drupal 使用 printf 语法(参看 http://php.net/printf)来实现占位符对查询语句中实际值的替换。根据用户提交信息的数据类型，可以选用不同的占位符。
表 5-1 列出了数据库查询的占位符及其含义。

表 5-1 数据库查询的占位符及其含义

占位符	含义
%s	字符串
%d	整数
%f	浮点数
%b	二进位数据；不要包含在' '
%%	插入一个%符 (比如，SELECT * FROM {users} WHERE name LIKE '%%%s%%')

db_query()的第一个参数总是查询语句本身。剩下的参数都是一些动态值，用来验证和入到查询字符串中。可以将这些值放在一个数组中，或者将每个值都作为一个独立的参数。后者更常用一些。

我们应该注意到，使用这个语法，TRUE, FALSE 和 NULL 将会被自动转换为了等价的数字形式(0 或 1)。一般情况下，都不会因此出现问题。

让我们看一些例子。在这些例子中，我们使用一个名为 joke 的数据库表，它包含了 3 个字段：节点 ID（整数），版本 ID（整数），还有包含笑话妙语的文本字段（关于 joke 模块的更多信息，参看第 7 章）。

让我们从一个简单的查询入手。 从 joke 表中取出所有字段的所有记录，需要满足的条件为----字段 vid 的整数值等于$node->nid 的值：

db_query('SELECT * FROM {joke} WHERE vid = %d', $node->vid);

向 joke 表中插入一行记录。新纪录中包含两个整数和一个字符串值。注意字符串值的占位符位于单引号中；这将帮助阻止 SQL 注入攻击。由于查询语句本身包含了单引号，所以我们在查询的外面使用了双引号：

```
db_query("INSERT INTO {joke} (nid, vid, punchline) VALUES (%d, %d, '%s')",
$node->nid, $node->vid, $node->punchline);
```

修改 joke 表中的所有记录，需要满足的条件为----字段 vid 的整数值等于$node->nid 的值。通过设置字段 puchline 等于$node->punchline 包含的字符串值来修改所有的这些记录：

```
db_query("UPDATE {joke} SET punchline = '%s' WHERE vid = %d", $node->punchline,
```

$node->vid);

从 joke 表中删除所有记录，需要满足的条件为----字段 vid 的整数值等于$node->nid 的值：
db_query('DELETE FROM {joke} WHERE nid = %d', $node->nid);

取回查询结果

有多种方式用于取回查询结果，这依赖于你的需求，你是需要单独的一行还是需要整个结果集，或者你打算获得一定范围内的结果集，是为了内部使用还是想将其分页显示。

获得单个值

如果你需要的仅仅是来自数据库的单个值，那么你可以使用 db_result()来取回该值。下面是一个例子，用来取回未被管理员禁用的注册用户总数（不包含匿名用户）：

```
$count = db_result(db_query('SELECT COUNT(uid) FROM {users} WHERE status = 1
AND uid != 0'));
```

获得多行

在大多数情况下，你需要从数据库中返回的都是多个字段。下面是一个典型的迭代模式，用于遍历整个结果集：

```
$type = 'blog';
$status = 1; // In the node table, a status of 1 means published.
$sql = "SELECT * FROM {node} WHERE type = '%s' AND status = %d";
$result = db_query(db_rewrite_sql($sql), $type, $status);
while ($data = db_fetch_object($result)) {
   $node = node_load($data->nid);
   print node_view($node, TRUE);
}
```

前面的代码片段将输出类型为 blog 的所有已发布节点（表 node 中的字段 status 的值，为 0 时意味着未发布，为 1 时意味着已发布）。我们接下来就会 讲解 db_rewrite_sql()。函数 db_fetch_object()从结果集中取出一行作为一个对象。如果想将取出的结果作为一个数组的话， 那么可以使用 db_fetch_array()。前者更为常用，因为与数组相比，大多数开发者都绝前者的语法更简明一些。

老葛的 Drupal 培训班 http://zhupou.cn

获得限制范围内的结果

你可能会想，对于一个有很多日志条目的站点，比如说有 10，000 个，那么运行前面的代码将会非常危险。我们将对这个语句的结果进行限制，仅仅取回 10 个最新发布的日志：

```
$type = 'blog';
$status = 1; // In the node table, a status of 1 means published.
$sql = "SELECT * FROM {node} n WHERE type = '%s' AND status = %d ORDER BY
n.created DESC";
```

```
$result = db_query_range(db_rewrite_sql($sql), $type, $status, 0, 10);
while ($data = db_fetch_object($result)) {
   $node = node_load($data->nid);
   print node_view($node, TRUE);
}
```

我们没有将语句传递给 db_query()并使用 LIMIT 条件语句，在这里我们使用了函数 db_query_range()。为什么呢？因为并非所有的数据库都支持 LIMIT 语法，所以我们需要使用 db_query_range()作为包装函数。

注意，我们将这些用来填充占位符的变量放在了范围的前面（也就是将 type 和 status 放在了 0，10 之前，如前面的例子所示）。

将结果分页显示

老葛的 Drupal 培训班 http://zhupou.cn

我们可以使用一个更好的方式来显示这些日志：分页显示。我们可以使用 Drupal 的分页器来实现这一点（如图 5-2）。让我们再次取回所有的日志条目，只是这次我们将其进行分页显示，在页面的底部，包含了指向其它结果页面的链接和“first”和“last”的链接。

```
$type = 'blog';
$status = 1;
$sql = "SELECT * FROM {node} n WHERE type = '%s' AND status = %d ORDER BY
n.created DESC";
$pager_num = 0; // This is the first pager on this page. We number it 0.
$result = pager_query(db_rewrite_sql($sql), 10, $pager_num, NULL, $type,
   $status);
while ($data = db_fetch_object($result)) {
   $node = node_load($data->nid);
   print node_view($node, TRUE);
}
// Add links to remaining pages of results.
print theme('pager', NULL, 10, $pager_num);
```

虽然 pager_query() 实际上不属于数据库抽象层，但是当你需要创建一个带有导航的分页显示时，它还是很有用的。最后一行调用的是 theme('pager')，它用来显示指向 其它页面的导航链接，你不需要向 theme('pager')中传递结果的总数，因为总数在调用 pager_query()时已被记录下来了。

图 5-2. *Drupal 的分页器，为结果集包含了内置的导航链接*

模式(Schema)API

(译者注：Schema 被翻译成了模式， Schema module 翻译成了模式模块， schema definition 翻译成了模式定义，这里面的句子有点绕口^_^)

通过数据库抽象层, Drupal 可以支持多个数据库(MySQL, PostreSQL,等等) 。对于那些需要创建自己的数据库表的模块，可以使用模式定义来向 Drupal 描述表结构。接着，Drupal 将定义翻译成适合数据库的语法。

使用模块的.install 文件

我们在第 2 章中已经看到，当我们编写的模块需要创建一个或者多个数据库表来存储信息时，创建和维护表结构的指令都放在了模块的.install 文件中。

创建数据库表

老葛的 Drupal 培训班 http://zhupou.cn

安装钩子函数一般将数据库表的安装委托给 drupal_install_schema()； 而 drupal_install_schema()负责从模块的模式钩子中获取模式定义，并修改数据库，如图 5-3 所示。接着，安装钩子函数再做一些其它的必需的安装工作。下面是来自 modules/book/book.install 文件的例子，这里将数据库表的安装委托给了 drupal_install_schema()。由于书籍模块需要处理书籍节点类型，所以在安装完数据库表后它还创建了这一节点类型。

```
/**
* Implementation of hook_install().
*/
function book_install() {
  // Create tables.
  drupal_install_schema('book');

  // Add the node type.
  _book_install_type_create();
}
```

模式一般这样定义：

```
$schema['tablename'] = array(
  // Table description.
  'description' => t('Description of what the table is used for.'),
    'fields' => array(
      // Field definition.
      'field1' => array(
```

```
            'type' => 'int',
            'unsigned' => TRUE,
            'not null' => TRUE,
            'default' => 0,
            'description' => t('Description of what this field is used for.'),
          ),
      ),
      // Index declarations.
      'primary key' => array('field1'),
  );
```

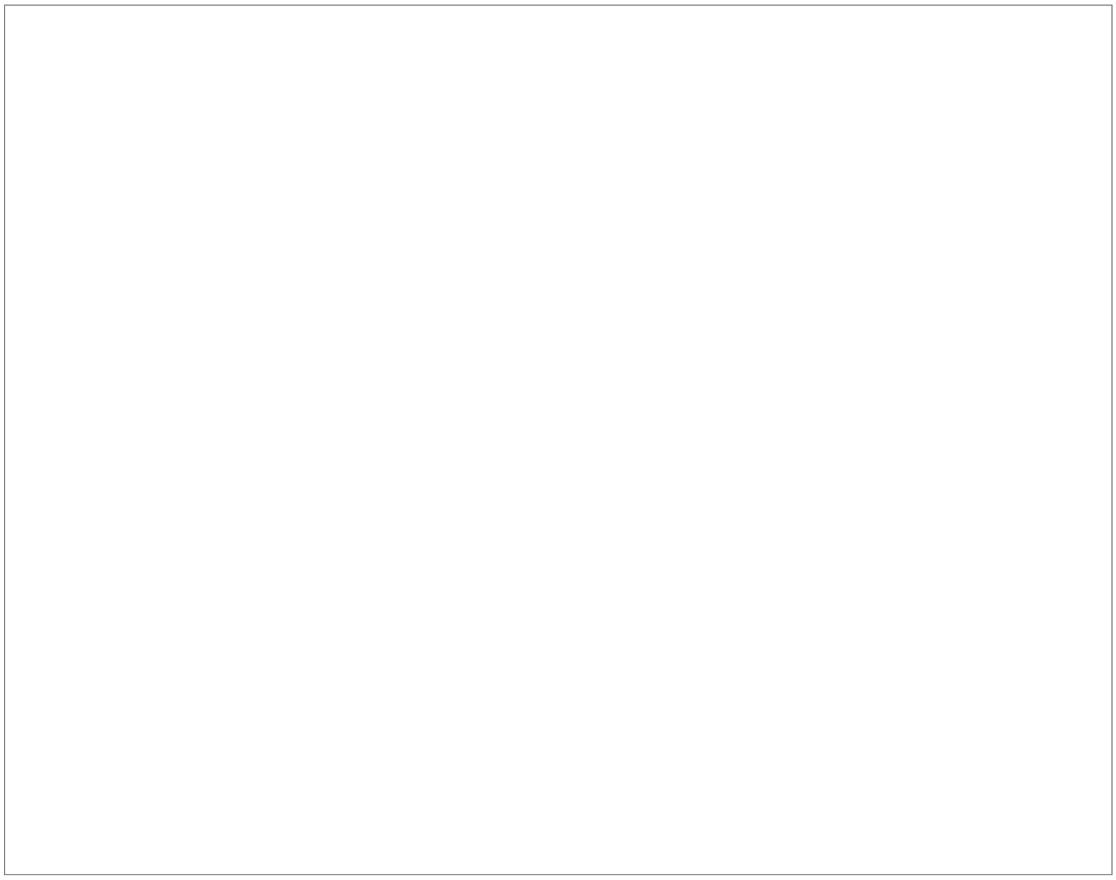

图 5-3.*使用模式定义创建数据库表*

创建数据库表(1)

老葛的 Drupal 培训班 http://zhupou.cn

让我们看一下 Drupal 的书籍模块中的模式定义，位于 modules/book/book.install 文件中：

```
/**
* Implementation of hook_schema().
*/
function book_schema() {
```

```
  $schema['book'] = array(
    'description' => t('Stores book outline information. Uniquely connects each node in the outline to a link in {menu_links}'),
    'fields' => array(
      'mlid' => array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'default' => 0,
        'description' => t("The book page's {menu_links}.mlid."),
      ),
      'nid' => array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'default' => 0,
        'description' => t("The book page's {node}.nid."),
      ),
      'bid' => array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'default' => 0,
        'description' => t("The book ID is the {book}.nid of the top-level page."),
      ),
    ),

    'primary key' => array('mlid'),
    'unique keys' => array(
      'nid' => array('nid'),
    ),
    'indexes' => array(
      'bid' => array('bid'),
    ),
  );

  return $schema;
}
```

这个模式定义描述了 book 表，它包含 3 个 int 类型的字段。它还有一个主键，一个唯一索引（这意味着该字段中的所有条目都是唯一的）和一个普通索引。注 意，在字段描述中，引用另一个表中的字段时，需要为其使用花括号。这样模式模块（参看下一节）可以为表的描述构建方便的超链接。

使用模式模块

老葛的 Drupal 培训班 http://zhupou.cn

现在你可能会想，“花这么大的功夫，创建一个这么复杂的数组，来向 Drupal 描述我的表结构，是不是有点得不偿失啊？”不要着急。你用用模式模块就知道了，你可以从 http://drupal.org/project/schema 下载到该模块，接着将其启用。导航到“管理➤站点构建 ➤模式”，点击 Inspect（检查）标签，你就能够看到所有数据库表的模式定义了。如果你为你的数据库表准备好了 SQL 脚本，那么使用模式模块就可以帮你自动生成模式定义，接着将模式定义复制粘贴到你的.install 文件中就可以了。

提示 你一般很少需要从头编写一个模式定义。一般，你可以使用已有的表，使用模式模块的 Inspect（检查）标签，让它帮你构建模式定义。

模式模块还允许你查看任意模块的模式。如图 5-4 所示，在模式模块中显示了书籍模块的模式。注意，表和字段描述中，花括号中的表的名字被转化为了有用的链接。

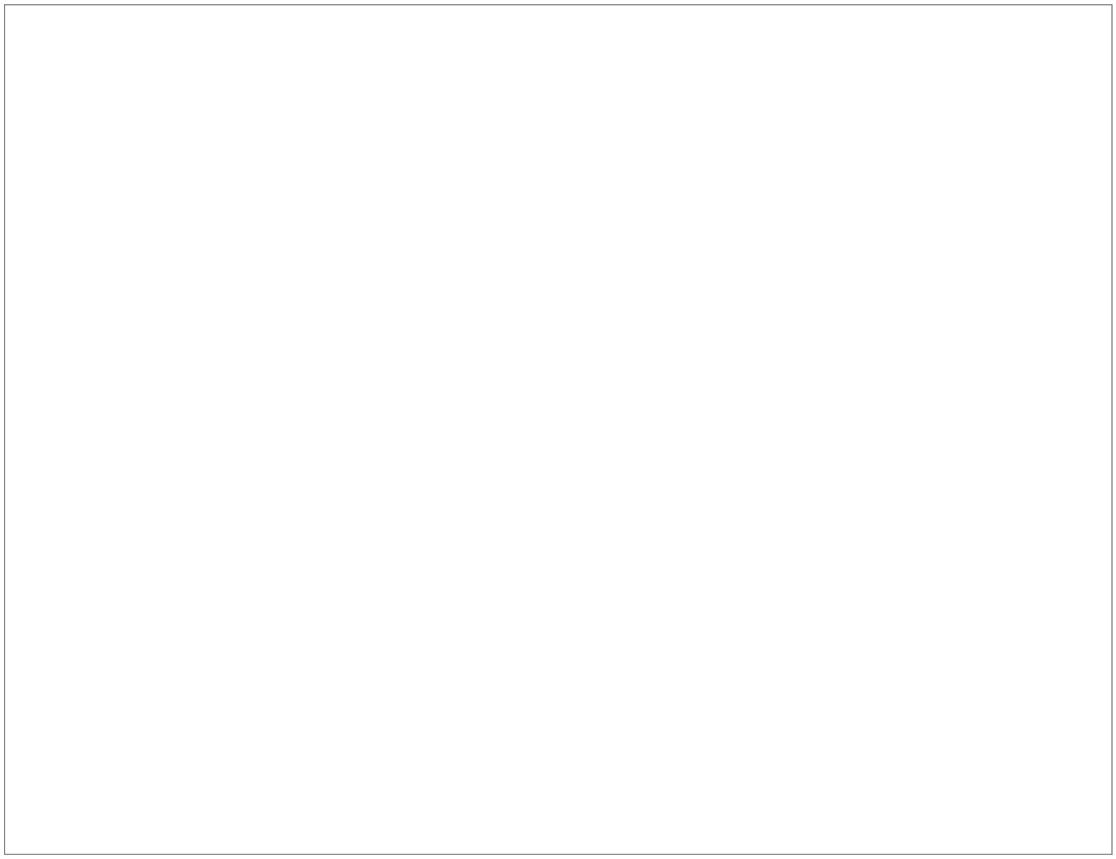

图 5-4.*模式模块显示了书籍模块的模式。*

从模式向数据库的字段类型映射

在模式定义中声明的字段类型，将会映射成数据库中的本地字段类型。例如，一个 size 为 tiny 的整数字段将映射为 MySQL 中的 TINYINT 字段，或者 PostgreSQL 中的 smallint 字段。实际的映射可查看数据库驱动文件中的 db_type_map()函数，比如 includes/database.pgsql.php(参看表 5-2, 本章后面讲到)。

文本型

文本型字段是用来包含文本的。

Varchar

Varchar，也就是变长字符字段；对于长度小于 256 字符的文本，通常使用这一字段类型。最大的字符长度，可以使用 length 键定义。MySQL 中 varchar 字段的长度为 0–255 字符（MySQL 5.0.2 及更早版本）和 0–65,535 字符（MySQL 5.0.3 及以后版本）；而 PostgreSQL 中 varchar 字段的长度则可以更大一些。

```
$field['fieldname'] = array(
   'type' => 'varchar', // Required.
   'length' => 255, // Required.
   'not null' => TRUE, // Defaults to FALSE.
   'default' => 'chocolate', // See below.
   'description' => t('Always state the purpose of your field.'),
);
```

如果 default 键未被设置，并且 not null 键被设置为了 FALSE，那么默认值将被设置为 NULL。

Char

Char 字段是定长字符字段，该字段的字符长度，可以使用 length 键定义。MySQL 中 char 字段的长度为 0–255 字符。

```
$field['fieldname'] = array(
   'type' => 'char', // Required.
   'length' => 64, // Required.
   'not null' => TRUE, // Defaults to FALSE.
   'default' => 'strawberry', // See below.
   'description' => t('Always state the purpose of your field.'),
);
```

如果 default 键未被设置，并且 not null 键被设置为了 FALSE，那么默认值将被设置为 NULL。

Text

Text 字段用于大块的文本。例如，node_revisions 表（存储节点正文的）中的 body 字段就是这种类型。Text 字段可以不使用默认值。

```
$field['fieldname'] = array(
    'type' => 'text', // Required.
    'size' => 'small', // tiny | small | normal | medium | big
    'not null' => TRUE, // Defaults to FALSE.
    'description' => t('Always state the purpose of your field.'),
);
```


数字型

数字型数据类型是用来存储数字的，它包括 integer（整数），serial（序列数），float（浮点数），和 numeric（数字）类型。

Integer

这个字段是用来存储整数的，比如节点 ID。如果 unsigned 键为 TRUE 的话，那么将不允许使用负整数。

```
$field['fieldname'] = array(
    'type' => 'int', // Required.
    'unsigned' => TRUE, // Defaults to FALSE.
    'size' => 'small', // tiny | small | medium | normal | big
    'not null' => TRUE, // Defaults to FALSE.
    'description' => t('Always state the purpose of your field.'),
);
```

Serial

一个序列字段是用来保存自增数字的。例如，当添加一个节点时，node 表中的 nid 字段将会自增。通过插入一行记录和调用 db_last_insert_id()来实现自增。如果在插入记录和取回最后 ID 之间，另一线程也插入了一条记录，此时会不会出错呢？由于它是基于单连 接追踪的，所以还会返回正确的 ID。一个序列字段必须被索引；通常会把它作为主键进行索引。

```
$field['fieldname'] = array(
    'type' => 'serial', // Required.
    'unsigned' => TRUE, // Defaults to FALSE. Serial numbers are usually positive.
    'size' => 'small', // tiny | small | medium | normal | big
    'not null' => TRUE, // Defaults to FALSE. Typically TRUE for serial fields.
    'description' => t('Always state the purpose of your field.'),
);
```

Float

浮点数字是用来存储浮点数据类型的。对于浮点数字来说，tiny, small, medium, 和 normal 型浮点一般是没有区别的；另外，big 型浮点用来声明双精度字段。

```
$field['fieldname'] = array(
```

```
    'type' => 'float', // Required.
    'unsigned' => TRUE, // Defaults to FALSE.
    'size' => 'normal', // tiny | small | medium | normal | big
    'not null' => TRUE, // Defaults to FALSE.
    'description' => t('Always state the purpose of your field.'),
);
```

Numeric

数字数据类型允许你声明数字的精度和小数位数。精度指的是数字的有效数字位数。小数位数指的是小数点右边的数字位数。例如，123.45 的精度为 5，小数位数为 2。这里不使用 size 键。到目前为止，Drupal 核心中还没有用到该字段。

```
$field['fieldname'] = array(
    'type' => 'numeric', // Required.
    'unsigned' => TRUE, // Defaults to FALSE.
    'precision' => 5, // Significant digits.
    'scale' => 2, // Digits to the right of the decimal.
    'not null' => TRUE, // Defaults to FALSE.
    'description' => t('Always state the purpose of your field.'),
);
```

日期和时间: Datetime

老葛的 Drupal 培训班 http://zhupou.cn

Drupal 核心没有使用这一数据类型,它使用的是存放在整数字段中的 Unix 时间戳。Datetime 格式是一个包含了日期和时间的混合格式。

```
$field['fieldname'] = array(
    'type' => 'datetime', // Required.
    'not null' => TRUE, // Defaults to FALSE.
    'description' => t('Always state the purpose of your field.'),
);
```

二进位：Blob

二进位大型对象数据类型用于存储二进制数据（例如，Drupal 的 cache 表用来存储缓存数据）。二进位数据包括音乐，图片，或者视频。有两个尺寸可用，normal 和 big。

```
$field['fieldname'] = array(
    'type' => 'blob', // Required.
    'size' => 'normal' // normal | big
    'not null' => TRUE, // Defaults to FALSE.
    'description' => t('Always state the purpose of your field.'),
);
```

使用 mysql_type 声明特定字段类型

老葛的 Drupal 培训班 http://zhupou.cn

如果你知道你的数据库引擎的准确字段类型, 那么你可以在你的模式定义中使用 mysql_type (或者 pgsql_type)键.这将覆写该数据库引擎的 type 和 size 键。例如，MySQL 有一个名为 TINYBLOB 的字段类型，专门用于小一点的二进 位大对象。如果对于 MySQL，我们为其使用 TINYBLOB 类型，而对于其它的数据库引擎，我们则为其使用普通的 BLOB 类型，那么在 Drupal 中该 如何声明呢？答案如下所示：

```
$field['fieldname'] = array(
    'mysql_type' > 'TINYBLOB', // MySQL will use this.
    'type' => 'blob', // Other databases will use this.
    'size' => 'normal', // Other databases will use this.
    'not null' => TRUE,
    'description' => t('Wee little blobs.')
);
```

MySQL 和 PostgreSQL 的本地类型，如表 5-2 所示。

表 5-2.*如何将模式定义中的 Type 和 Size 键映射到本地的数据库类型*

模式定义		本地数据库字段类型	
类型	**尺寸**	**MySQL**	**PostgreSQL**
varchar	normal	VARCHAR	varchar
char	normal	CHAR	character
text	tiny	TINYTEXT	text
text	small	TINYTEXT	text
text	medium	MEDIUMTEXT	text
text	big	LONGTEXT	text
text	normal	TEXT	text
serial	tiny	TINYINT	serial
serial	small	SMALLINT	serial
serial	medium	MEDIUMINT	serial
serial	big	BIGINT	bigserial
serial	normal	INT	serial
int	tiny	TINYINT	smallint
int	small	SMALLINT	smallint
int	medium	MEDIUMINT	int
int	big	BIGINT	bigint
int	normal	INT	int
float	tiny	FLOAT	real
float	small	FLOAT	real
float	medium	FLOAT	real
float	big	DOUBLE	double precision

float normal FLOAT real
numeric normal DECIMAL numeric
blob big LONGBLOB bytea
blob normal BLOB bytea
datetime normal DATETIME timestamp

维护数据库表

老葛的 Drupal 培训班 http://zhupou.cn

当你为一个模块创建新版本时，你可能需要修改数据库模式。可能，你添加了一列，或者为某一列添加了索引。由于该表已经包含了数据，所以你不能简单地删除并重建该表。下面给出了如何保证平稳的修改数据库表：

1. 更新你的.install 文件中 hook_schema()的实现，这样模块的新用户安装时，用的就是新模式了。你的.install 文件中的模式定义，应该总是最新的，以反映你模块的表和字段的当前结构。
2. 通过写一个更新函数，让已有用户对现有模块进行更新。更新函数按照顺序进行命名，起始数字一般是基于 Drupal 版本的。例如，Drupal6 的第一个更新函数可以为 modulename_update_6000()，那么第二个更新函数就为 modulename_update_6001()。下面是来自 modules/comment/comment.install 中的例子，这里向评论表 中的父 ID（pid）列添加了一个索引：

```
/**
 * Add index to parent ID field.
 */
function comment_update_6003() {
   $ret = array(); // Query results will be collected here.
   // $ret will be modified by reference.
   db_add_index($ret, 'comments', 'pid', array('pid'));
   return $ret;
}
```

在更新了模块以后，用户运行 http://example.com/update.php 时，就会调用这个函数。

警告 因为，你每次添加一个表，字段，或者索引时，都会修改 hook_schema()实现中的模式定义，所以你的更新函数千万不要使用这里的模式定义。你可以把 hook_schema()实现中的模式定义看成是当前的，而把更新函数中的模式看成是过去的。参看 http://drupal.org/node/150220。

用来处理模式的函数的完整列表，可参看 http://api.drupal.org/api/group/schemaapi/6。

提示 Drupal 会追踪一个模块当前所用的模式版本。这一信息存储在 system 表中。在运行完本节所示的更新以后，评论模块对应记录中的 schema_version 的 值就变成了 6003。为了让 Drupal 忘记该项，可以使用 devel 模块中的“Reinstall Modules”（重装模块）选项，或者从 system 表中删除该模块的记录。

在 Uninstall 中删除数据库表

老葛的 Drupal 培训班 http://zhupou.cn

当一个模块被禁用时，该模块存储在数据库中的数据还被保留着，如果哪天管理员改变了主意，还可以重新安装该模块。在“管理➤站点构建 ➤模块”页面，有一个卸载标签，是用来从数据库中删除数据的。对于你模块的数据库表的删除，也放在这里进行，只需要在模块的.install 文件中实现 uninstall（卸载）钩子就可以了。同时，你可能还想删除你在模块中定义的各种变量。下面是第 2 章里 annotation 模块中的例子：

```
/**
 * Implementation of hook_uninstall().
 */
function annotate_uninstall() {
   // Use schema API to delete database table.
   drupal_uninstall_schema('annotate');
   // Clean up our entry in the variables table.
   variable_del('annotate_nodetypes');
}
```

使用 hook_schema_alter()修改已有模式

老葛的 Drupal 培训班 http://zhupou.cn

一般来说，模块会创建和使用它们自己的数据库表。但是，如果你的模块想修改一个已有的表时，那该怎么办呢？假定你的模块需要向 node 表中添加一列。最简单的方式是直接访问你的数据库，在里面添加一列。但是这样以来，反映实际数据库表的模式定义就会出现不兼容的问题。这里有一个更好的方式，那就是使用 hook_schema_alter()。

警告 hook_schema_alter()是 Drupal 中的新钩子，对于如何使用这个钩子，什么才是最佳的用法，还存在争论。更多详细，参看 http://api.drupal.org/api/group/hooks/6。

假定你有一个模块，想按照某种方式来标记节点，一般来说，你可以创建一个数据库表，并使用节点 ID 将其与 node 表关联起来，但是你没有这样做，出于性能 的考虑，你决定完全使用 node 表。这样一来，你的模块需要做两件事情：在你模块的安装过程中修改 node 表，并且修改模式定义以如实地反映数据库中的表 结构。前者可以使用 hook_install()，后者可以使用 hook_schema_alter()。假定你的模块为 markednode.module，那么在你的 markednode.install 文件中应该包含以下函数：

```
/**
 * Implementation of hook_install().
```

```
 */
function markednode_install() {
    $field = array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'default' => 0,
        'initial' => 0, // Sets initial value for preexisting nodes.
        'description' => t('Whether the node has been marked by the
            markednode module.'),
    );

    // Create a regular index called 'marked' on the field named 'marked'.
    $keys['indexes'] = array(
        'marked' => array('marked')
    );

    $ret = array(); // Results of the SQL calls will be stored here.
    db_add_field($ret, 'node', 'marked', $field, $keys);
}

/**
 * Implementation of hook_schema_alter(). We alter $schema by reference.
 *
 * @param $schema
 * The system-wide schema collected by drupal_get_schema().
 */
function markednode_schema_alter(&$schema) {
    // Add field to existing schema.
    $schema['node']['fields']['marked'] = array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'default' => 0,
        'description' => t('Whether the node has been marked by the
            markednode module.'),
    );
}
```

使用 drupal_write_record()进行插入和更新

程序员常遇到的一个问题，就是处理数据库中新纪录的插入和已有记录的更新。代码一般都会先检查当前的操作是一个插入操作还是一个更新操作，接着再执行合适的操作。

因为 Drupal 所用的每个表都使用模式来描述，所 Drupal 知道一个表中都包含哪些字段以及每个字段的默认值。通过向 drupal_write_record()传递一个包含了字段和数值的数组，那么你就可以让 Drupal 为你生成和执行 SQL 了，这样你就不需要自己手 写了。

假定你有一个表，用来追踪你收集的小兔子。那么你模块中的用来描述表结构的模式钩子应该是这样的：

```
/**
 * Implementation of hook_schema().
 */
function bunny_schema() {
  $schema['bunnies'] = array(
    'description' => t('Stores information about giant rabbits.'),
    'fields' => array(
      'bid' => array(
        'type' => 'serial',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'description' => t("Primary key: A unique ID for each bunny."),
      ),
      'name' => array(
        'type' => 'varchar',
        'length' => 64,
        'not null' => TRUE,
        'description' => t("Each bunny gets a name."),
      ),
      'tons' => array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'description' => t('The weight of the bunny to the nearest ton.'),
      ),
    ),
    'primary key' => array('bid'),
    'indexes' => array(
      'tons' => array('tons'),
    ),
  );
```

```
    return $schema;
}
```

插入一条新纪录非常方便，更新记录也是如此：

```
$table = 'bunnies';
$record = new stdClass();
$record->name = t('Bortha');
$record->tons = 2;
drupal_write_record($table, $record);

// The new bunny ID, $record->bid, was set by drupal_write_record()
// since $record is passed by reference.
watchdog('bunny', 'Added bunny with id %id.', array('%id' => $record->bid));

// Change our mind about the name.
$record->name = t('Bertha');

// Now update the record in the database.
// For updates we pass in the name of the table's primary key.
drupal_write_record($table, $record, 'bid');
watchdog('bunny', 'Updated bunny with id %id.', array('%id' => $record->bid));
```

这里也支持数组，如果$record 是一个数组的话，那么 drupal_write_record()会在内部将其转化为一个对象。

使用 hook_db_rewrite_sql()将查询暴露给其它模块

老葛的 Drupal 培训班 http://zhupou.cn

这个钩子可以用来修改 Drupal 中任何地方的查询，这样你就不用直接修改相关模块了。如果你将一个查询传递给 db_query()， 而且你相信其他人可能想修改它，那么你就需要把它包装到函数 db_rewrite_sql()里面，这样其他的开发者就可以访问它了。当执行一个这样的查 询时，它首先检查所有实现了 hook_db_rewrite_sql()的模块，并给它们一个修改查询的机会。例如，节点模块修改了节点列表查询，从而将 受节点访问规则保护的节点排除在外。

警告 如果你执行一个节点列表查询（例如，你直接对 node 表查询，来获取所有节点的一些子集），但是你没有使用 db_rewrite_sql()来包装你的查询，那么节点访问规则将被忽略，这是由于节点模块无法修改你的查询，因此无法排除受保护的节点。

如果查询语句不是你的，但是你又想在你的模块中修改这一查询，那么你需要在你的模块中实现

hook_db_rewrite_sql()。

表 5-3 使用 SQL 重写的两种方式的总结

表 5-3. *使用 db_rewrite_sql()函数 VS 使用 hook_db_rewrite_sql()钩子*

名称	什么时候使用
db_rewrite_sql()	当编写节点列表查询或者其它查询时，你想让别人能够对它进行修改它时
hook_db_rewrite_sql()	当你想修改其它模块中的查询时

使用 hook_db_rewrite_sql()

老葛的 Drupal 培训班 http://zhupou.cn

下面是函数签名：
function hook_db_rewrite_sql($query, $primary_table = 'n', $primary_field = 'nid',
$args = array())

参数如下所示：

- $query:可被覆写的 SQL 查询。
- $primary_table: 在该查询中，包含主键字段的表的名字或者别名。例如，对于 node 表它的值为 n，而对于 comment 表它的值为 c (例如, 对于 SELECT nid FROM {node} n, 该值应为 n)。常用的值如表 5-4 所示。
- $primary_field:在该查询中的主字段的名称。它的值可为 nid, tid,vid, cid。（例如，如果你的查询要得到一列节点 ID，那么主字段就为 nid）。
- $args:一个包含了参数的数组，用来传递给每个模块中 hook_db_rewrite_sql()的实现。

表 5-4. *$primary_table 别名的常用值*

表	别名
blocks	b
comments	c
forum	f
node	n
menu	m
term_data	t
vocabulary	v

修改其它模块的查询

让我们看一个 hook_db_rewrite_sql()的具体实现。下面的例子利用了 node 表中 moderate 列来覆写节点查询。在我们修改了查询以后，那些不具有“管理内容”权限的用户，就会看不到处于待审核状态的节点（也就是，moderate 列为 1）。

```
/**
 * Implementation of hook_db_rewrite_sql().
 */
function moderate_db_rewrite_sql($query, $primary_table, $primary_field, $args) {
  switch ($primary_field) {
    case 'nid':
      // Run only if the user does not already have full access.
      if (!user_access('administer content')) {
        $array = array();
        if ($primary_table == 'n') {
          // Node table is already present;
          // just add a WHERE to hide moderated nodes.
          $array['where'] = "(n.moderate = 0)";
        }
        // Test if node table is present but alias is not 'n'.
        elseif (preg_match('@{node} ([A-Za-z_]+)@', $query, $match)) {
          $node_table_alias = $match[1];

          // Add a JOIN so that the moderate column will be available.
          $array['join'] = "LEFT JOIN {node} n ON $node_table_alias.nid = n.nid";

          // Add a WHERE to hide moderated nodes.
          $array['where'] = "($node_table_alias.moderate = 0)";
        }
      return $array;
    }
  }
}
```

注意，我们检查所有查询，对于主键为 nid 的并且主表为 node 的查询，我们向里面插入一些额外信息。让我们看一下实际效果。

下面是最初的查询，未经 moderate_db_rewrite_sql()处理的：

```
SELECT * FROM {node} n WHERE n.type = 'blog' and n.status = 1
```

下面是 moderate_db_rewrite_sql()处理过后的查询：

```
SELECT * FROM {node} n WHERE n.type = 'blog' and n.status = 1 AND n.moderate = 0
```

moderate_db_rewrite_sql()被调用后，它向输入的查询中追加了 AND n.moderate = 0。这个钩子通常还用于限制对节点、词汇表、术语、或者评论的访问。

db_rewrite_sql()局限于它能够理解的 SQL 语法。当你需要对表进行关联时，使用 JOIN 语法，而不是在 FROM 语句中对表进行关联。

下面的不正确:

SELECT * FROM {node} AS n, {comment} AS c WHERE n.nid = c.nid

这个正确:

SELECT * FROM {node} n INNER JOIN {comment} c ON n.nid = c.nid
老葛的 Drupal 培训班 http://zhupou.cn

在 Drupal 中使用多个数据库连接

老葛的 Drupal 培训班 http://zhupou.cn

数据库抽象层给我们带来了多项好处，比如函数名称更好记了，查询中内置了安全特性，等等。有时候，我们需要连接到第 3 方或者遗留的数据库上，如果 Drupal 的数据库 API 能满足这一需要并同时提供安全特性的话，那该多美啊。不错，我们可以实现这一点！例如，在你的模块中，你可以连接到一个非 Drupal 的数据库，并从中取出数据。

在 settings.php 文件中，$db_url 既可以是一个字符串（通常是这样的），也可以是包含多个数据库连接的字符串数组。下面是默认的语法，声明了一个单独的连接：
$db_url = 'mysql://username:password@localhost/databasename';

当使用一个数组时，它的键就是在激活数据库连接时所引用的简洁名称，而它的值就是连接的字符串本身。下面是一个例子，在这里我们声明了两个连接字符串， default（默认的）和 legacy（遗留的）：

$db_url['default'] = 'mysql://user:password@localhost/drupal6';
$db_url['legacy'] = 'mysql://user:password@localhost/legacydatabase';

注意 Drupal 本身使用的数据库一定要以 default 为键。

当你需要连接到 Drupal 中其它的数据库上时，你首先使用它的键名激活该连接，当你使用完连接时，将它切换回到默认的连接上。

```
// Get some information from a non-Drupal database.
db_set_active('legacy');
$result = db_query("SELECT * FROM ldap_user WHERE uid = %d", $user->uid);

// Switch back to the default connection when finished.
db_set_active('default');
```

注意 切记一定要切换回到默认的连接上，这样 Drupal 可以干净的完成整个请求生命周期并重新回到自己的表中。

由于数据库抽象层设计的是为每个数据库使用相同的函数名，所以不能够同时使用多个数据库后台（比如，同时使用 MySQL 和 PostgreSQL）。然而，在同一个站点还是可以同时使用 MySQL 和 PostgreSQL 连接的，如何实现的更多详细，请参看 http://drupal.org/node/19522。

使用临时表

老葛的 Drupal 培训班 http://zhupou.cn

如果你需要做很多的处理，那么在请求过程中你可能需要创建一个临时表。你可以通过调用函数 db_query_temporary()来完成它，如下所示：
$result = db_query_temporary($sql, $arguments, $temporary_table_name);

接下来你就可以使用临时表的名字来对临时表进行查询。对于临时表的名字，推荐的一种命名方式是：“temp”+你的模块名+具体名字。
$final_result = db_query('SELECT foo FROM temp_mymodule_nids');

注意，对于临时表，不需要使用花括号来对表进行前缀化，这是因为临时表的是暂时存在的，它不能经过表的前缀化处理。与之相对应的，永久的表的名字应该使用花括号，以支持表的前缀化。

注意 在 Drupal 核心中没有使用临时表，而 Drupal 所用的数据库用户也有可能没有权限来创建临时表。因此，模块的作者不应该假定所有的用户都具有该权限。

编写你自己的数据库驱动器

假定你想为一个新生的未来的名为 DNAbase 的数据库编写一个数据库抽象层，该数据库使用分子计算来提升性能。我们不需要从头开始，而是复制一份已有的抽象层，接着修改它。我们将使用 PostgreSQL 的实现，这是因为 MySQL 的驱动器被拆分成了，一个 includes/database.mysql-common.inc 文件，和两个单独的 mysql、mysqli 驱动器文件。

首先，我们复制一份 includes/database.pgsql.inc 并将其重命名为 includes/database.dnabase.inc。接着我们修改每个包装函数的内部逻辑，使用 DNAbase 的功能来代替 PostgreSQL 的功能。当我们完成了所有的这些修改以后，那么在我们的文件中声明了以下函数：

```
_db_query($query, $debug = 0)
db_add_field(&$ret, $table, $field, $spec, $new_keys = array())
db_add_index(&$ret, $table, $name, $fields)
db_add_primary_key(&$ret, $table, $fields)
db_add_unique_key(&$ret, $table, $name, $fields)
db_affected_rows()
db_change_field(&$ret, $table, $field, $field_new, $spec, $new_keys = array())
db_check_setup()
db_column_exists($table, $column)
db_connect($url)
db_create_table_sql($name, $table)
db_decode_blob($data)
db_distinct_field($table, $field, $query)
db_drop_field(&$ret, $table, $field)
db_drop_index(&$ret, $table, $name)
db_drop_primary_key(&$ret, $table)
db_drop_table(&$ret, $table)
db_drop_unique_key(&$ret, $table, $name)
db_encode_blob($data)
db_error()
db_escape_string($text)
```

```
db_fetch_array($result)
db_fetch_object($result)
db_field_set_default(&$ret, $table, $field, $default)
db_field_set_no_default(&$ret, $table, $field)
db_last_insert_id($table, $field)
db_lock_table($table)
db_query_range($query)
db_query_temporary($query)
db_query($query)
db_rename_table(&$ret, $table, $new_name)
db_result($result)
db_status_report()
db_table_exists($table)
db_type_map()
db_unlock_tables()
db_version()
```

通过更新 settings.php 中的$db_url，我们在 Drupal 中连接到 DNAbase 数据库来测试这一系统。它看起来是这样的：

```
$db_url = 'dnabase://john:secret@localhost/mydnadatabase';
```

其中 john 是用户名，secret 是密码，而 mydnadatabase 是我们将要连接的数据库名。你可能还想创建一个测试模块，来直接调用这些函数以确保它们正常工作。

总结

老葛的 Drupal 培训班 http://zhupou.cn

读完本章后，你应该能够

- 理解 Drupal 的数据库抽象层
- 进行基本的查询
- 从数据库中获取单个或者多个结果
- 获取一个限定范围内的结果
- 使用分页器
- 理解 Drupal 的模式 API
- 编写其它开发者可以修改的查询
- 干净的修改其它模块中的查询
- 连接多个数据库，包括遗留的数据库
- 编写一个抽象层驱动器