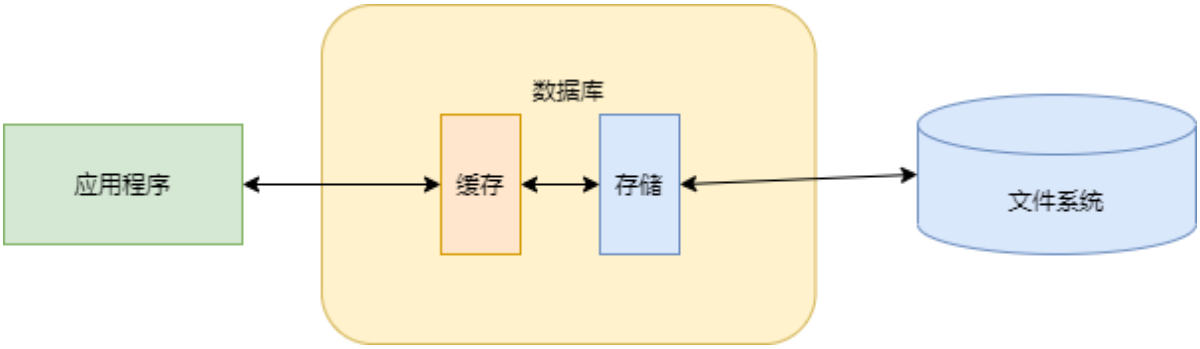
12 数据库和SQL语句

数据库简介

数据库概念

在实际的开发过程中，应用程序会有大量的数据需要持久地存储在磁盘之中，目前我们已知的持久化存储的方式就是采用文件，但是在磁盘物理的结构上，文件其实是一种顺序的字符集合。使用文件可以很方便地输入和输出数据，但是这些数据都是简单的字节流，不包含任何复杂数据结构，所以应用开发者需要花费大量的时间去解析文件的内容，并将其映射到内存的数据结构以供应用程序后续使用。除此之外，由于磁盘的顺序读写的速度要远远快于随机读写，所以为了访问位置靠后的数据，要消耗大量的时间（类似于链表的访问）。另外，文件系统的数据无法保证数据的安全性，应用程序没有办法在存储数据的时候检查其是否正确，而当部分数据损坏时，文件系统本身无法定位问题和设计修复方案。

数据库即数据仓库，它是一系列结构化的数据的集合，其作用是对**结构化的数据进行持久存储**。我们把数据库中的数据、数据库管理系统和其他关联应用统称为数据库。数据库在系统的位置介于磁盘文件系统和应用程序之间，一方面，数据库对上层应用程序提供了持久化地读写数据的接口，使应用程序可以尽量高效地将内存数据结构在磁盘中访问、存储和修改。另一方面，数据库采用了高级的数据结构来组织在磁盘的数据，并在内存当中设计了高效的缓存系统，从而可以充分利用磁盘的各种特点（比如读写速度远逊色于内存，顺序读写的速度远高于随机访问的速度等等），从而最大化上层应用程序的性能。



数据库分类

我们无意于过分深入数据库的历史沿革，而是会聚焦于当前的数据库产品的市场——当前的数据库可以分成两大阵营：关系型数据库和非关系型数据库。

- 关系型数据库使用**关系**来组织数据库当中的各个数据，所谓的关系可以认为就是有若干**行**和**列**构成的**表格**。由于关系型数据库便于理解易于使用，所以长期以来是市场上的主流数据库模式。关系型数据库通常会一个名为**SQL（Structured Query Language，结构化查询语言）**的语言来访问和更新。现在市场占用率比较高的关系型数据库有这些：Oracle、MySQL、PostgreSQL、DB2、Informix、Sybase、SQL Server 2000、Access、SQLite。Oracle是商业数据库，性能优异但是收费昂贵，一般用于传统行业和银行领域；MySQL和PostgreSQL是开源的关系型数据库管理系统，被大多数互联网公司采用。
- 非关系型数据库现在一般统称为NoSQL，由于在市场竞争中始终无法取代关系型数据库，现在一般将其解释为"Not only SQL"，从而表示非关系型数据库是关系型数据的补充之义。NoSQL一般认为不适合作为通用领域的存储部件，但是在一些特定领域，比如高并发领域等等还是有一定的用武之地。常见的NoSQL产品有：Redis、Memcached、MongoDB、Cassandra等等。NoSQL的存储结构各有千秋，不过一般都会支持key-value的键值对存储结构。

本课程选用的数据库管理系统是MySQL，是一种关系型数据库。

关系型数据库和SQL

关系

关系型数据库使用**关系**来组织数据库当中的各个数据。所谓的关系可以认为就是有若干**行**和**列**构成的**表**。表中的一行也叫一个记录，它以**元组**的形式存在，用来描述一个具体对象各个数据。

在这个元组中，会存在多个**域**，每个域代表了某个对象当中的某一个**属性**。所有对象的同一属性构成了一列，每一列有着自己独特的名字。对于关系当中的每一个属性，都有一个允许取值的集合，我们也称之为域。

主键和外键

如何区分一个关系中两个不同的元组？我们需要挑选元组中某个属性或者多个属性的组合作为标识，不同的元组的在这些属性的取值上一定不同，满足这样需求的属性集合称为**超级键**。而这些集合中元素最少的称为**候选键**（其真子集必然不是超级键），数据库管理员可以选择一个候选键当作**主键**。我们尽量不要用描述具体信息的属性充当主键（比如姓名、职位），主键一般的选择是一个无意义的自动自增的整数，这样可以尽量避免修改（比如假设使用公司名作为主键，若发生公司合并的情况）。

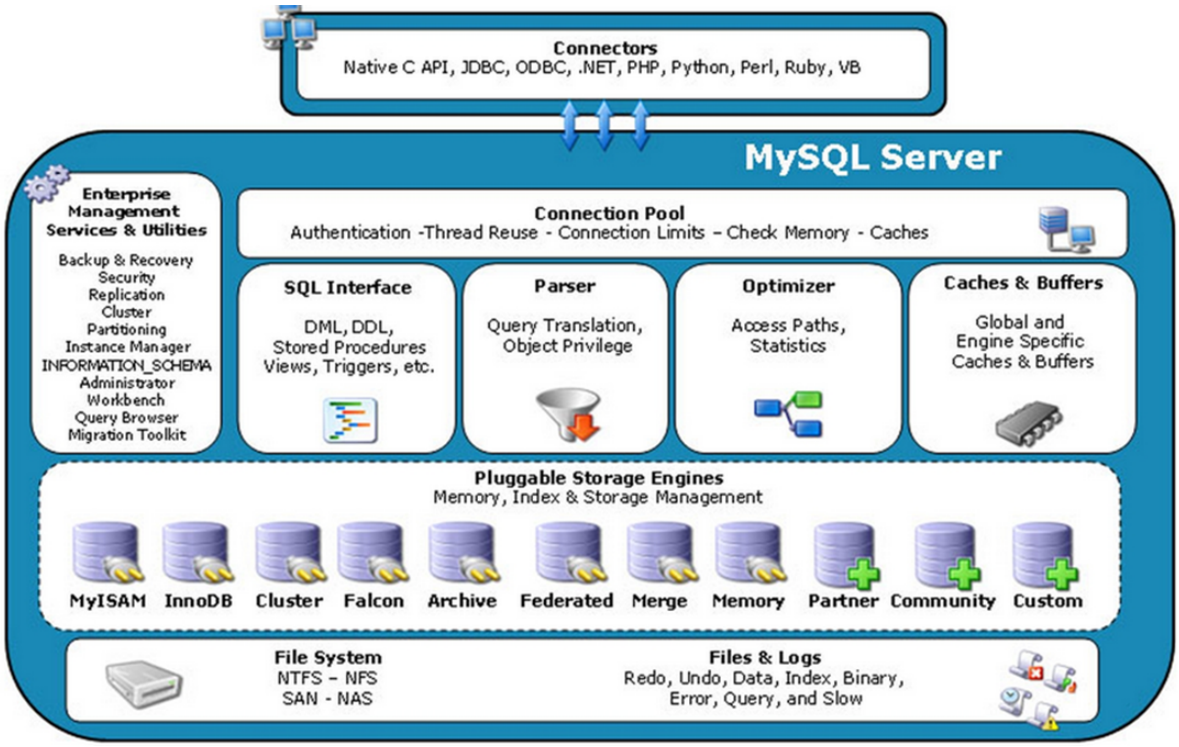
有些情况下，业务的数据内容非常复杂，因此只用一个关系不能将其完整存储，需要设计多个关系来满足业务需求。这些关系之间一般来说是存在联系的，一个关系中r1的某一列可能和另一个关系r2的主键描述的是同一份的数据，在业务中经常可能会出现根据r1表的信息查询r2关系中的属性。对于r1关系而言，这就是**参照**r2关系的**外键**，r1是外键的参照关系，r2是外键的被参照关系。

SQL

SQL即结构化查询语言，其最初的数学来源是关系代数和元组关系演算。在逐渐地演变过程，SQL既可以用来管理和维护数据库系统，又可以查询、添加、更新和删除数据库当中的数据。

SQL和之前所学习过的C语言不同：C语言是一种过程式的语言，它被用来描述怎样去完成一个任务各个步骤；而SQL是一种**声明式**的语言，它并不关心如果实际地从数据库中提取的过程，用户也无需了解数据的存储结构，它只是简单地说明了任务的目标，而具体的执行步骤则由数据库系统的语句解析器和优化器生成，并最终被存储引擎执行。

这是某个版本的mysql内部结构图示意（仅供参考）：



虽然SQL已经被ANSI组织定义为标准，但是各个不同的数据库对标准的SQL支持不太一致。并且大部分数据库都在标准的SQL上做了扩展。如果我们只使用标准SQL的核心功能，那么所有数据库通常都可以执行；而不常用的SQL功能，不同的数据库支持的程度都不一样。

总的来说，SQL语言定义了这么几种操作数据库的能力：

- **DDL：Data Definition Language，数据定义语言。**DDL允许用户定义数据，也就是创建表、删除表、修改表结构这些操作。通常，DDL由数据库管理员执行。
- **DML：Data Manipulation Language，数据操作语言。**DML为用户提供查询、添加、删除、更新数据的能力，这些是应用程序对数据库的日常操作。其中查询操作是使用得最频繁的操作。

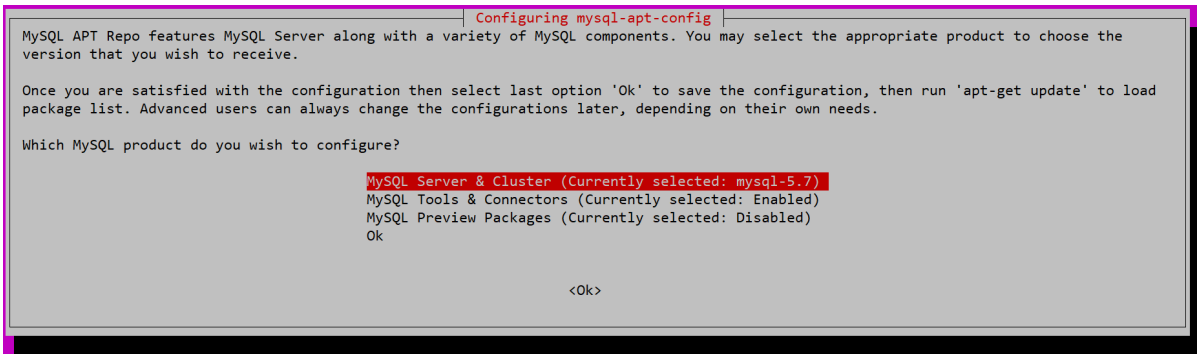
MySQL初步

MySQL 8.0/5.7的安装

> （官方提供的安装过程：MySQL :: A Quick Guide to Using the MySQL APT Repository）

安装MySQL需要首先配置包管理器信息，然后再进行安装。本课程选择的是8.0版本（Ubuntu 20以及以上的版本无需此操作），5.7版本也可以使用。

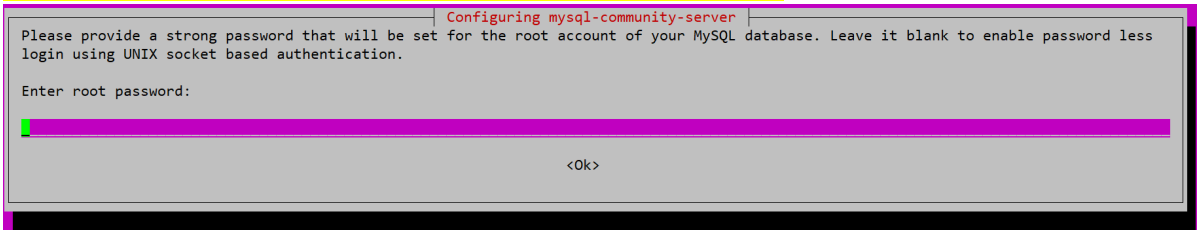
```
# 下面的wget命令用于下载mysql-apt-config
# 也可以从本地服务器当中下载到Linux中
wget https://dev.mysql.com/get/mysql-apt-config_0.8.22-1_all.deb
# 安装mysql-apt-config
sudo dpkg -i mysql-apt-config_0.8.22-1_all.deb
```



进入上述界面以后使用上下方向键移动选项和enter键选定选项，使得MySQL服务端的版本是8.0（或者是5.7），所有选项如上图所示。

```
#安装mysql
sudo apt update # 这里不能少
sudo apt-get install mysql-server #安装服务端和命令行客户端
```

在安装MySQL请务必先设置好一个用户和密码，这样就可以不使用Linux的root用户连接mysqld了。（如果此步骤没有出现，需要执行后续的手动修改登录密码的过程）



如果出现密码验证模式的选择，最好选择"Legacy Authentication Method"。虽然第一项是8.0版本的推荐方式，但是继续使用以前的加密方式比较方便于Navicat去连接，故更加推荐一点。（如果本界面没有弹出，可以无视）

使用下面的命令如果可以登录成功，说明安装成功，否则需要执行手动修改密码的过程：

```
# 首先在命令行当中输入下列命令
mysql -u root -p
# 随后输入刚才设置的密码（如果未设置密码需要手动修改登录密码）
```

手动修改登录密码

首先，将Linux的登录用户切换成root用户。然后直接用特权用户身份登录mysql。

```
su
mysql
```

在登录之后输入如下的命令可以修改root的访问方式(下面方法适用于5.7版本):

```
use mysql;
-- 下面这句命令有点长，请注意。
update mysql.user set
authentication_string=password('你的密码') where
user='root' and Host ='localhost';
update user set plugin="mysql_native_password";
flush privileges;
quit
```

下面方法适用于8.0版本:

```
use mysql;
-- 下面这句命令有点长，请注意。
alter user 'root'@'localhost' identified with mysql_native_password by '你的密码';
flush privileges;
quit
```

删除MySQL的方法

请不要自己随意卸载mysql，很容易导致无法再次安装。如需卸载，请参考下面的链接:

Ubuntu18.04彻底删除MySQL数据库*青蛙组长的博客-CSDN博客*ubuntu 删除mysql

适用于安装过程出现错误的同学。

连接到MySQL服务端

mysql正常安装并启动以后，会以守护进程的方式运行服务端进程，其名字为mysqld。用户如果需要使用数据库的功能，需要使用客户端连接服务端。最常见的一种客户端就是命令行客户端，其英文缩写为mysql-cli。

使用命令行客户端连接mysql服务端的命令如下:

```
mysql -u root -p
# 随后在提示符之后输入密码即可
# 登录成功会显示命令提示符
# mysql>
```

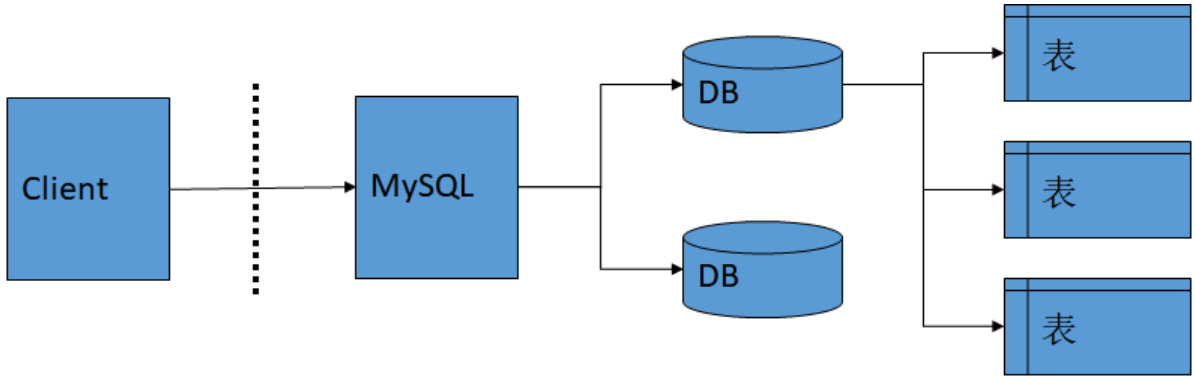
在命令提示行输入quit即可退出

```
quit
# 注意这里没有分号
# 按下ctrl+d也是可以的
```

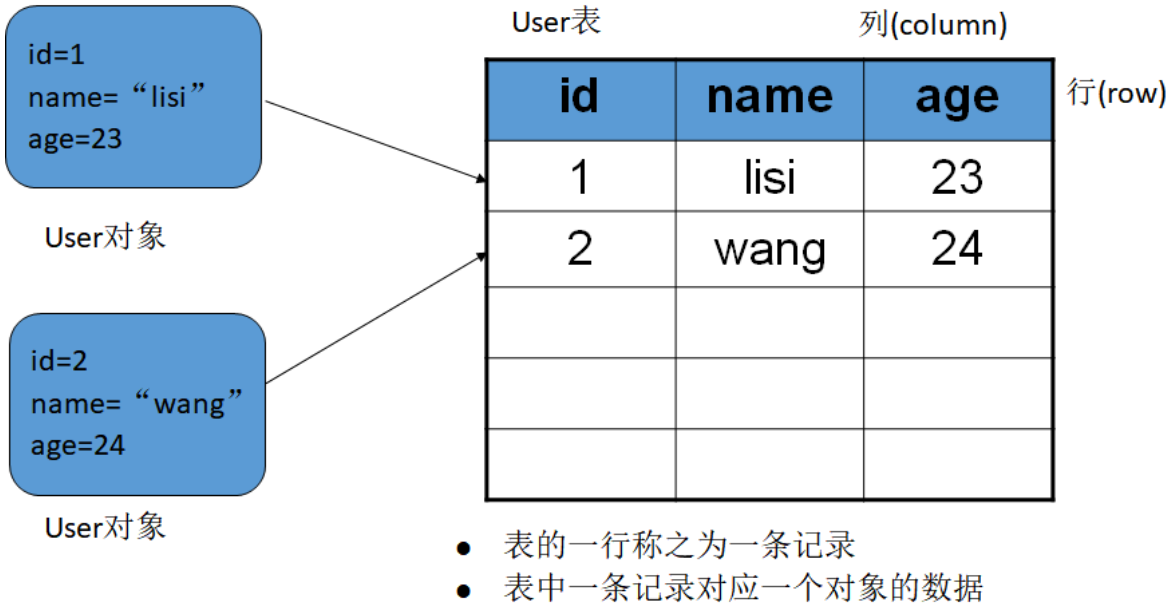
关于MySQL的基本结构

在操作系统当中安装MySQL之后，会启动一个mysqld服务端进程，mysqld服务端会监听一个端口（默认是3306），本地进程可以通过本地套接字、远端进程可以通过TCP协议与其建立连接，并且通过服务端来访问修改和更新其管理的数据。

MySQL当中的所有数据都以**数据库**（这个数据库名词不是广义的数据库软件的含义，只是用来MySQL用来管理数据的单元，可以理解为文件夹）为单位进行存储，每个数据库的内部由若干张**关系**（也叫**表**）组成。



不同的表中存储了不同对象的若干不同属性的数据的值。



使用命令和SQL操作MySQL

在连接上mysql服务端以后，就可以键入查询命令。我们首先先来介绍操作mysql属性相关的一些命令。

显示数据库的版本

在命令行当中输入以下内容：`select version(), current_date;`，然后按下回车就会出现类似下面的内容。

```
mysql> select version(), current_date;
+-----------+--------------+
| version() | current_date |
+-----------+--------------+
| 8.0.31    | 2023-02-15   |
+-----------+--------------+
1 row in set (0.04 sec)
```

这个命令的例子就是一个典型的查询命令的使用方式，有着这样的特点：

- 一个查询命令总是以分号结尾的（非查询命令quit与之不同），换行并不意味着一个命令输入完成；
- 一个查询命令的返回结果总是以表的形式出现的，其中第一行会显示每一列的名字；
- 一个查询命令的返回结果还会告知用户结果的行数和执行时间（按真实时钟统计），以便粗略地了解性能。
- 查询命令当中的关键字是无视大小写的，所以下面的命令是等价的：

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

创建一个数据库

首先，使用show语句可以展示当前服务端已经存在的数据库：

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| connecttest        |
| information_schema |
| mycloud            |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
6 rows in set (1.48 sec)
#mysql数据库的内容是用来管理各个用户的权限的
```

如果想要选择使用某个数据库，可以使用use语句（use和quit一样，不用加分号结尾）：

```
mysql> use mysql
Database changed
```

使用create语句可以创建一个新的数据库：

```
mysql> create database test;
Query OK, 1 row affected (0.03 sec)
```

值得注意的是，在Linux系统当中，这个数据库的名字test是大小写敏感的。后续使用的表的名字也是一样大小写敏感。

创建完数据库并不意味着已经选择该数据库，还需要继续使用use语句进行选择：

```
mysql> use test
Database changed
```

创建数据库的命令是持久的，所以本次连接创建的数据库下一次也会存在，但是use语句需要在每次连接以后都执行一次。使用select语句可以知道当前选择的数据库：

```
mysql> select database();
+------------+
| database() |
+------------+
| test       |
+------------+
1 row in set (0.00 sec)
```

使用drop database语句可以删除一个数据库：

```
 drop database test;
```

创建一张表

首先，我们可以使用show语句展示当前选择的数据库当中的所有表（当然，目前是空的）：

```
mysql> show tables;
Empty set (0.00 sec)
```

下面，我们需要设计一张表pet来存储关于宠物的信息。比如，宠物的昵称、主人的信息、种族、性别等等。需要注意的是，年龄这种随着时间会动态改变的数据不适合存储在数据库当中，更好的方式是存储出生日期，而查询年龄信息时通过计算得到。

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
    ->  species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
Query OK, 0 rows affected (0.11 sec)
```

使用create table语句会在选择的数据库当中创建一张表。在语句中，首先需要指明表的名字，然后再详细描述每一列的名字和类型，以及可选的额外属性。

- VARCHAR(20)是一种长度上限为20的变长字符串；
- CHAR(1)是一种固定长度为1的定长字符串；
- DATE是日期。

关于常见的数据类型

以下是数据经常使用的数据类型：

分类	数据类型	说明
数值类型	BIT(M)	位类型。M指定位数，默认值1，范围1-64。
	BOOL，BOOLEAN	使用0或1表示假或真
	TINYINT [UNSIGNED] [ZEROFILL]	带符号的范围是-128到127。无符号0到255。
	SMALLINT [UNSIGNED] [ZEROFILL]	2的16次方
	INT [UNSIGNED] [ZEROFILL]	2的32次方
	BIGINT [UNSIGNED] [ZEROFILL]	2的64次方
	FLOAT[(*M*,*D*)] [UNSIGNED] [ZEROFILL]	单精度浮点数，M指定显示长度，D指定小数位数
	DOUBLE[(*M*,*D*)] [UNSIGNED] [ZEROFILL]	双精度浮点数
文本、二进制类型	CHAR(size)	定长字符串，size是字符数
	VARCHAR(size)	变长字符串，size是上限
	BLOB	二进制数据 比如图片、音乐等
	LONGBLOB	比较大的二进制数据
	TEXT(clob)	文本数据
	LONGTEXT(longclob)	比较大的文本数据
时间和日期	DATE	日期
	TIME	（一天之中的）时间
	DATETIME	时间和日期的组合
	TIMESTAMP	时间和日期的组合 时间戳（其显示受时区的影响）

在创建完数据库以后，可以再次使用show语句查看表的信息。

```
mysql> show tables;
+----------------+
| Tables_in_test |
+----------------+
| pet            |
+----------------+
1 row in set (0.00 sec)
```

也可以使用describe语句查看表的详细信息。

```
mysql> describe pet; # 用desc也可以
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

增加表内容：插入数据

我们可以使用insert into语句来插入一行数据，values后面应该说明各列的内容，其中字符和日期型数据应包含在单引号中；而NULL用来说明该列内容为空。

```
mysql> INSERT INTO pet VALUES ('Puffball','Diane','hamster','f','1999-03-
30',NULL);
Query OK, 1 row affected (0.03 sec)
```

在插入数据的时候，可以只指定部分列的数据，这样新的一行中未指定的列的内容就是默认值：

```
mysql> INSERT INTO pet (name,owner,species,birth) VALUES
('Whistler','Gwen','bird','1997-12-09');
Query OK, 1 row affected (0.01 sec)
```

接下来，我们使用一个txt文档来描述一个表格。表格当中的内容是我们接下来准备插入的数据，其中，列与列使用tab制表符隔开，\N表示空的属性。

```
Fluffy  Harold  cat f   1993-02-04  \N
Claws   Gwen    cat m   1994-03-17  \N
Buffy   Harold  dog f   1989-05-13  \N
Fang    Benny   dog m   1990-08-27  \N
Bowser  Diane   dog m   1979-08-31  1995-07-29
Chirpy  Gwen    bird    f   1998-09-11  \N
Whistler    Gwen    bird    \N  1997-12-09  \N
Slim    Benny   snake   m   1996-04-29  \N
```

相关命令如下：

```
INSERT INTO pet VALUES('Fluffy','Harold','cat','f','1993-02-04' ,NULL);
INSERT INTO pet VALUES('Claws','Gwen','cat','m','1994-03-17'    ,NULL);
INSERT INTO pet VALUES('Buffy','Harold','dog','f','1989-05-13'  ,NULL);
INSERT INTO pet VALUES('Fang','Benny','dog','m','1990-08-27'    ,NULL);
INSERT INTO pet VALUES('Bowser','Diane','dog','m','1979-08-31','1995-07-29');
INSERT INTO pet VALUES('Chirpy','Gwen','bird','f','1998-09-11' ,NULL);
INSERT INTO pet VALUES('Whistler','Gwen','bird',NULL,'1997-12-09' ,NULL);
INSERT INTO pet VALUES('Slim','Benny','snake','m','1996-04-29',NULL);
```

查询表内容：展示表中的数据

展示的所有数据

使用select语句可以从表当中提取信息。其中最简单的使用方法就是列出所有行和所有列：

```
mysql> select * from pet;
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Puffball | Diane  | hamster | f    | 1999-03-30 | NULL       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       |
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       |
| Bowser   | Diane  | dog     | m    | 1979-08-31 | 1995-07-29 |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       |
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       |
+----------+--------+---------+------+------------+------------+
10 rows in set (0.00 sec)
```

在本次查询当中，*表示要访问的是所有列，而from pet说明了从表pet当中提取信息。

展示部分行的信息

在select语句当中增加where子句可以约束提取数据行的范围。where子句的参数是一个条件表达式。

- = 用来表示相等关系（<>表示不相等）：

```
mysql> SELECT * FROM pet WHERE name = 'Bowser';
+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1979-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
1 row in set (0.00 sec)
```

- <、>、<=、>= 可以用来描述数值类型或者日期类型的大小关系：

```
mysql> SELECT * FROM pet WHERE birth >= '1998-1-1';
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Puffball | Diane | hamster | f    | 1999-03-30 | NULL  |
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
+----------+-------+---------+------+------------+-------+
2 rows in set (0.00 sec)
```

- 使用AND和OR可以组合成复合条件表达式

```
mysql>  SELECT * FROM pet WHERE species = 'dog' AND sex = 'f';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
1 row in set (0.00 sec)

mysql> SELECT * FROM pet WHERE species = 'snake' OR species = 'bird';
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL  |
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL  |
| Slim     | Benny | snake   | m    | 1996-04-29 | NULL  |
+----------+-------+---------+------+------------+-------+
4 rows in set (0.00 sec)
```

- AND的优先级高于OR，也可以使用括号来指定优先级：

```
mysql>  SELECT * FROM pet WHERE (species = 'cat' AND sex = 'm')
    ->  OR (species = 'dog' AND sex = 'f');
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
2 rows in set (0.00 sec)
```

展示部分列的信息

select语句的中select的参数可以指定提取数据列的名字和顺序：

```
mysql>  SELECT name, birth FROM pet;
+----------+------------+
| name     | birth      |
+----------+------------+
| Puffball | 1999-03-30 |
| Whistler | 1997-12-09 |
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Buffy    | 1989-05-13 |
| Fang     | 1990-08-27 |
| Bowser   | 1979-08-31 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
+----------+------------+
```

```
10 rows in set (0.01 sec)
```

如果结果当中存在重复的元组，可以在select参数前增加distinct修饰从而对结果进行去重。下面是例子：

```
mysql> SELECT owner FROM pet;
+--------+
| owner  |
+--------+
| Diane  |
| Gwen   |
| Harold |
| Gwen   |
| Harold |
| Benny  |
| Diane  |
| Gwen   |
| Gwen   |
| Benny  |
+--------+
10 rows in set (0.00 sec)

mysql>  SELECT DISTINCT owner FROM pet;
+--------+
| owner  |
+--------+
| Diane  |
| Gwen   |
| Harold |
| Benny  |
+--------+
4 rows in set (0.00 sec)
```

select的参数和where子句可以配合使用：

```
mysql> SELECT name, species, birth FROM pet
    ->  WHERE species = 'dog' OR species = 'cat';
+--------+---------+------------+
| name   | species | birth      |
+--------+---------+------------+
| Fluffy | cat     | 1993-02-04 |
| Claws  | cat     | 1994-03-17 |
| Buffy  | dog     | 1989-05-13 |
| Fang   | dog     | 1990-08-27 |
| Bowser | dog     | 1979-08-31 |
+--------+---------+------------+
5 rows in set (0.00 sec)
```

对结果按列排序

使用select语句的结果默认是没有明确顺序，有些情况下，用户需要对结果进行排序以更快地得到想要的内容，此时，就可以使用order by子句：

```
mysql>  SELECT name, birth FROM pet ORDER BY birth;
+----------+------------+
| name     | birth      |
+----------+------------+
| Bowser   | 1979-08-31 |
| Buffy    | 1989-05-13 |
| Fang     | 1990-08-27 |
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Slim     | 1996-04-29 |
| Whistler | 1997-12-09 |
| Whistler | 1997-12-09 |
| Chirpy   | 1998-09-11 |
| Puffball | 1999-03-30 |
+----------+------------+
10 rows in set (0.00 sec)
```

增加desc修饰可以按降序排序：

```
mysql>  SELECT name, birth FROM pet ORDER BY birth DESC;
+----------+------------+
| name     | birth      |
+----------+------------+
| Puffball | 1999-03-30 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Claws    | 1994-03-17 |
| Fluffy   | 1993-02-04 |
| Fang     | 1990-08-27 |
| Buffy    | 1989-05-13 |
| Bowser   | 1979-08-31 |
+----------+------------+
10 rows in set (0.00 sec)
```

也可以根据多列情况依次排序：

```
mysql> SELECT name, species, birth FROM pet
    ->  ORDER BY species, birth DESC;
+----------+---------+------------+
| name     | species | birth      |
+----------+---------+------------+
| Chirpy   | bird    | 1998-09-11 |
| Whistler | bird    | 1997-12-09 |
| Whistler | bird    | 1997-12-09 |
| Claws    | cat     | 1994-03-17 |
| Fluffy   | cat     | 1993-02-04 |
| Fang     | dog     | 1990-08-27 |
```

```
| Buffy    | dog     | 1989-05-13 |
| Bowser   | dog     | 1979-08-31 |
| Puffball | hamster | 1999-03-30 |
| Slim     | snake   | 1996-04-29 |
+----------+---------+------------+
10 rows in set (0.00 sec)
```

空值NULL相关的操作

空值NULL是一种特殊的值，一般情况下它描述了未确定的值的含义，因此对空值的处理有别于其他类型的值。

使用is null和is not null运算符可以判断某个值是否为NULL：

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----------+---------------+
| 1 IS NULL | 1 IS NOT NULL |
+-----------+---------------+
|         0 |             1 |
+-----------+---------------+
1 row in set (0.00 sec)
```

不能对NULL使用比较运算符，比如<、<>、=之类的：

```
mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
+----------+-----------+----------+----------+
| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
+----------+-----------+----------+----------+
|     NULL |      NULL |     NULL |     NULL |
+----------+-----------+----------+----------+
1 row in set (0.00 sec)
```

简单的模式匹配

使用like和not like可以启用通配符匹配：

- _匹配任意单个字符；
- %匹配任意字符串；

下面是3个示例：

```
mysql> SELECT * FROM pet WHERE name LIKE 'b%';
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1979-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM pet WHERE name LIKE '%fy';
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
```

```
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM pet WHERE name LIKE '%w%';
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1979-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
4 rows in set (0.00 sec)
```

聚集函数和统计行数

有些情况下，用户并不需要得到整个表的某个子集的内容，而是希望能够得到一些统计信息，比如总行数、最大值之类的。此时可以选择使用聚集函数。其中count函数可以用来统计行数。

聚集函数最简单的用法就是对表整体进行统计：

```
mysql>  SELECT COUNT(*) FROM pet;
+----------+
| COUNT(*) |
+----------+
|       10 |
+----------+
1 row in set (0.07 sec)
```

使用group by子句以后可以进行分组统计：

```
mysql>  SELECT owner, COUNT(*) FROM pet GROUP BY owner;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Diane  |        2 |
| Gwen   |        4 |
| Harold |        2 |
| Benny  |        2 |
+--------+----------+
4 rows in set (0.00 sec)

mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
+---------+----------+
| species | COUNT(*) |
+---------+----------+
| hamster |        1 |
| bird    |        3 |
| cat     |        2 |
| dog     |        3 |
| snake   |        1 |
```

```
+---------+----------+
5 rows in set (0.01 sec)
```

在有些情况下，用户需要在得到聚集的结果之后再对对结果进行筛选。但是在这种情况下，where子句无能为力——因为where子句的生效时机在group by聚集以前。想要解决前述问题就需要引入having子句，having子句可以在聚集之后再对结果进行筛选：

```
mysql> SELECT species,COUNT(species) FROM pet GROUP BY species HAving
COUNT(species) >= 2;
+---------+----------------+
| species | count(species) |
+---------+----------------+
| bird    |              3 |
| cat     |              2 |
| dog     |              3 |
+---------+----------------+
3 rows in set (0.00 sec)
```

多表连接操作

首先，我们需要额外创建一个表，并填入数据：

```
CREATE TABLE event (name VARCHAR(20), date DATE,type VARCHAR(15), remark
VARCHAR(255));
INSERT INTO event VALUES('Fluffy','1995-05-15','litter','4 kittens, 3 female, 1
male');
INSERT INTO event VALUES('Buffy','1993-06-23','litter','5 puppies, 2 female, 3
male');
INSERT INTO event VALUES('Buffy','1994-06-19','litter','3 puppies, 3 female');
INSERT INTO event VALUES('Chirpy','1999-03-21','vet','needed beak straightened');
INSERT INTO event VALUES('Slim','1997-08-03','vet','broken rib');
INSERT INTO event VALUES('Bowser','1991-10-12','kennel',NULL);
INSERT INTO event VALUES('Fang','1991-10-12','kennel',NULL);
INSERT INTO event VALUES('Fang','1998-08-28','birthday','Gave him a new chew
toy');
INSERT INTO event VALUES('Claws','1998-03-17','birthday','Gave him a new flea
collar');
INSERT INTO event VALUES('Whistler','1998-12-09','birthday','First birthday');
```

下面，我们来执行一个多表内连接的例子：

```
mysql> SELECT pet.name,remark
    ->  FROM pet INNER JOIN event
    ->  ON pet.name = event.name
    ->  WHERE event.type = 'litter';
+--------+----------------------------+
| name   | remark                     |
+--------+----------------------------+
| Fluffy | 4 kittens, 3 female, 1 male |
| Buffy  | 3 puppies, 3 female        |
| Buffy  | 5 puppies, 2 female, 3 male |
+--------+----------------------------+
```

在from语句当中可以对多表进行连接操作以生成一个新表。

inner join表示内连接：pet表和event表的内连接会创建一个新表：新表当中的每一行，都是从由pet的某一行和event中某一行组合而成，并且必须两行都满足ON子句的条件。

需要注意的是，如果两个表当中存在同名字的列，需要使用.运算符来指明该列所归属的表。

```
mysql> SELECT * FROM pet INNER JOIN event ON pet.name = event.name;
+---------+--------+---------+------+------------+------------+----------+------
------+----------+----------------------------+
| name     | owner  | species | sex  | birth      | death      | name     | date
      | type     | remark                     |
+---------+--------+---------+------+------------+------------+----------+------
------+----------+----------------------------+
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       | Fluffy   | 1995-
05-15 | litter   | 4 kittens, 3 female, 1 male |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       | Buffy    | 1993-
06-23 | litter   | 5 puppies, 2 female, 3 male |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       | Buffy    | 1994-
06-19 | litter   | 3 puppies, 3 female         |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       | Chirpy   | 1999-
03-21 | vet      | needed beak straightened    |
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       | Slim     | 1997-
08-03 | vet      | broken rib                  |
| Bowser   | Diane  | dog     | m    | 1979-08-31 | 1995-07-29 | Bowser   | 1991-
10-12 | kennel   | NULL                        |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       | Fang     | 1991-
10-12 | kennel   | NULL                        |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       | Fang     | 1998-
08-28 | birthday | Gave him a new chew toy     |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       | Claws    | 1998-
03-17 | birthday | Gave him a new flea collar  |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       | Whistler | 1998-
12-09 | birthday | First birthday              |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       | Whistler | 1998-
12-09 | birthday | First birthday              |
+---------+--------+---------+------+------------+------------+----------+------
------+----------+----------------------------+
11 rows in set (0.00 sec)
```

通过给表起别名可以实现自己和自己内连接，下面是示例：

```
mysql>  SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species
    ->  FROM pet AS p1 INNER JOIN pet AS p2
    ->  ON p1.species = p2.species
    ->  AND p1.sex = 'f' AND p1.death IS NULL
    ->  AND p2.sex = 'm' AND p2.death IS NULL;
+--------+------+-------+------+---------+
| name   | sex  | name  | sex  | species |
+--------+------+-------+------+---------+
| Fluffy | f    | Claws | m    | cat     |
| Buffy  | f    | Fang  | m    | dog     |
+--------+------+-------+------+---------+
2 rows in set (0.01 sec)
```

使用内连接得到的新表存在一个特点，那就是左表的行和右表的行都必须满足条件才会组合成新行出现在结果当中。在有些情况下，用户除了需要知道左表和右表当中有哪里公共对象以外，还需要知道哪些左表中的对象从来没有在右表当中出现过。这种情况下就需要使用左外连接：

```
mysql> SELECT * FROM pet LEFT OUTER JOIN event ON pet.name = event.name;
+----------+--------+---------+------+------------+------------+----------+------
------+----------+----------------------------+
| name     | owner  | species | sex  | birth      | death      | name     | date
      | type     | remark                     |
+----------+--------+---------+------+------------+------------+----------+------
------+----------+----------------------------+
| Puffball | Diane  | hamster | f    | 1999-03-30 | NULL       | NULL     | NULL
      | NULL     | NULL                       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       | Whistler | 1998-
12-09 | birthday | First birthday             |
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       | Fluffy   | 1995-
05-15 | litter   | 4 kittens, 3 female, 1 male |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       | Claws    | 1998-
03-17 | birthday | Gave him a new flea collar  |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       | Buffy    | 1994-
06-19 | litter   | 3 puppies, 3 female        |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       | Buffy    | 1993-
06-23 | litter   | 5 puppies, 2 female, 3 male |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       | Fang     | 1998-
08-28 | birthday | Gave him a new chew toy     |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       | Fang     | 1991-
10-12 | kennel   | NULL                       |
| Bowser   | Diane  | dog     | m    | 1979-08-31 | 1995-07-29 | Bowser   | 1991-
10-12 | kennel   | NULL                       |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       | Chirpy   | 1999-
03-21 | vet      | needed beak straightened   |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       | Whistler | 1998-
12-09 | birthday | First birthday             |
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       | Slim     | 1997-
08-03 | vet      | broken rib                 |
+----------+--------+---------+------+------------+------------+----------+------
------+----------+----------------------------+
12 rows in set (0.00 sec)
```

右外连接和左外连接类型，不同之处在于会把右表当中存在，而在左表中未出现的对象也放入新表当中。

```
delete from pet where name = 'Chirpy'; # 先删除左表当中的一行
mysql> SELECT * FROM pet LEFT OUTER JOIN event ON pet.name = event.name;
+----------+--------+---------+------+------------+------------+----------+------
------+----------+----------------------------+
| name     | owner  | species | sex  | birth      | death      | name     | date
      | type     | remark                     |
+----------+--------+---------+------+------------+------------+----------+------
------+----------+----------------------------+
| Puffball | Diane  | hamster | f    | 1999-03-30 | NULL       | NULL     | NULL
      | NULL     | NULL                       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       | Whistler | 1998-
12-09 | birthday | First birthday             |
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       | Fluffy   | 1995-
05-15 | litter   | 4 kittens, 3 female, 1 male |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       | Claws    | 1998-
03-17 | birthday | Gave him a new flea collar  |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       | Buffy    | 1994-
06-19 | litter   | 3 puppies, 3 female        |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       | Buffy    | 1993-
06-23 | litter   | 5 puppies, 2 female, 3 male |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       | Fang     | 1998-
08-28 | birthday | Gave him a new chew toy    |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       | Fang     | 1991-
10-12 | kennel   | NULL                       |
| Bowser   | Diane  | dog     | m    | 1979-08-31 | 1995-07-29 | Bowser   | 1991-
10-12 | kennel   | NULL                       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       | Whistler | 1998-
12-09 | birthday | First birthday             |
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       | Slim     | 1997-
08-03 | vet      | broken rib                 |
+----------+--------+---------+------+------------+------------+----------+------
------+----------+----------------------------+
11 rows in set (0.00 sec)
```

全外连接是左外连接和右外连接的并集，但是Mysql不支持。

嵌套子查询

在select-from-where结构当中，可以将另一个查询的结果（称为**子查询**）嵌套在where子句当中。常用的运算符包括in和not in，这样可以用来限制where子句的查询范围。

```
# 查询同时在pet表和event表的当中的name
mysql> SELECT name FROM pet WHERE name IN (SELECT DISTINCT name from event);
+----------+
| name     |
+----------+
| Whistler |
| Fluffy   |
| Claws    |
| Buffy    |
| Fang     |
```

```
| Bowser   |
| Whistler |
| Slim     |
+----------+
8 rows in set (0.00 sec)
# 查询在pet表中，但是不在event表的当中的name
mysql> SELECT name FROM pet WHERE name NOT IN (SELECT DISTINCT name from event);
+----------+
| name     |
+----------+
| Puffball |
+----------+
1 row in set (0.00 sec)
```

修改表内容：更新和修改

更新表的内容

使用update语句可以修改表内的内容：

```
mysql> UPDATE event SET remark = 'test' WHERE remark IS NULL;
Query OK, 2 rows affected (0.04 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> select * from event;
+----------+------------+----------+-----------------------------+
| name     | date       | type     | remark                      |
+----------+------------+----------+-----------------------------+
| Fluffy   | 1995-05-15 | litter   | 4 kittens, 3 female, 1 male |
| Buffy    | 1993-06-23 | litter   | 5 puppies, 2 female, 3 male |
| Buffy    | 1994-06-19 | litter   | 3 puppies, 3 female         |
| Chirpy   | 1999-03-21 | vet      | needed beak straightened    |
| Slim     | 1997-08-03 | vet      | broken rib                  |
| Bowser   | 1991-10-12 | kennel   | test                        |
| Fang     | 1991-10-12 | kennel   | test                        |
| Fang     | 1998-08-28 | birthday | Gave him a new chew toy     |
| Claws    | 1998-03-17 | birthday | Gave him a new flea collar  |
| Whistler | 1998-12-09 | birthday | First birthday              |
+----------+------------+----------+-----------------------------+
10 rows in set (0.00 sec)
```

删除表的内容

使用delete from语句可以删除表当中某些行：

```
mysql> DELETE FROM event WHERE name = 'Fang';
Query OK, 2 rows affected (0.03 sec)

mysql> SELECT * FROM event;
+----------+------------+----------+-----------------------------+
| name     | date       | type     | remark                      |
+----------+------------+----------+-----------------------------+
```

```
| Fluffy   | 1995-05-15 | litter   | 4 kittens, 3 female, 1 male |
| Buffy    | 1993-06-23 | litter   | 5 puppies, 2 female, 3 male |
| Buffy    | 1994-06-19 | litter   | 3 puppies, 3 female         |
| Chirpy   | 1999-03-21 | vet      | needed beak straightened    |
| Slim     | 1997-08-03 | vet      | broken rib                  |
| Bowser   | 1991-10-12 | kennel   | test                        |
| Claws    | 1998-03-17 | birthday | Gave him a new flea collar  |
| Whistler | 1998-12-09 | birthday | First birthday              |
+----------+------------+----------+-----------------------------+
8 rows in set (0.00 sec)
```

操作表结构

获取表的创建信息

使用show create table可以展示表创建的信息。

```
mysql> SHOW CREATE TABLE event;
+-------+-----------------------------------------------------------------
------------------------------------------------------------------------
---------------------------------------------------------------+
| Table | Create Table


                                                               |
+-------+-----------------------------------------------------------------
------------------------------------------------------------------------
---------------------------------------------------------------+
| event | CREATE TABLE `event` (
  `name` varchar(20) DEFAULT NULL,
  `date` date DEFAULT NULL,
  `type` varchar(15) DEFAULT NULL,
  `remark` varchar(255) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+-------+-----------------------------------------------------------------
------------------------------------------------------------------------
---------------------------------------------------------------+
1 row in set (0.05 sec)
```

修改表的结构

使用alter table语句可以修改表的结构。包括可以增加、修改和删除表的属性和相关约束。

下面是增加列、删除列和修改列的名字和类型的例子：

```
mysql> ALTER TABLE event ADD (info varchar(100));
Query OK, 0 rows affected (0.57 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC event;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
```

```
+--------+-------------+------+-----+---------+-------+
| name   | varchar(20)  | YES  |     | NULL    |       |
| date   | date         | YES  |     | NULL    |       |
| type   | varchar(15)  | YES  |     | NULL    |       |
| remark | varchar(255) | YES  |     | NULL    |       |
| info   | varchar(100) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
5 rows in set (0.13 sec)

mysql> ALTER TABLE event DROP info;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC event;
+--------+-------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| name   | varchar(20)  | YES  |     | NULL    |       |
| date   | date         | YES  |     | NULL    |       |
| type   | varchar(15)  | YES  |     | NULL    |       |
| remark | varchar(255) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)


mysql> ALTER TABLE event CHANGE remark remark1 varchar(254);
Query OK, 8 rows affected (0.22 sec)
Records: 8  Duplicates: 0  Warnings: 0

mysql> DESC event;
+---------+-------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20)  | YES  |     | NULL    |       |
| date    | date         | YES  |     | NULL    |       |
| type    | varchar(15)  | YES  |     | NULL    |       |
| remark1 | varchar(254) | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

修改表的名字

使用rename table语句可以更改一个表的名字。

```
mysql> RENAME TABLE event TO events;
Query OK, 0 rows affected (0.01 sec)

mysql> show tables;
+----------------+
| Tables_in_test |
+----------------+
| events         |
| pet            |
+----------------+
2 rows in set (0.00 sec)
```

表的约束

一致性约束

为了保证数据的完整性，数据库当中所有存储数据的值都必须满足一些特定的**一致性约束**。这些一致性约束是为了保证插入到数据库中的数据是正确的，它防止了用户可能的输入错误。主要分为以下三类：

- 实体完整性：规定表的一行（即每一条记录）在表中是唯一的实体。实体完整性通过表的主键来实现。
- 域完整性：指数据库表的列（即字段）必须符合某种特定的数据类型或约束。比如NOT NULL。
- 参照完整性：保证一个表的外键和另一个表的主键对应。参照的完整性要求关系中不允许引用不存在的实体。与实体完整性是关系模型必须满足的完整性约束条件，目的是保证数据的一致性。

本课程只介绍实体完整性。

可以在创建表的时候设置好约束，下面是创建表的时候设置约束的语法。

```
 col_name datatype
[NOT NULL | NULL] [DEFAULT default_value]
[AUTO_INCREMENT] [UNIQUE [KEY]] [[PRIMARY] KEY]
[reference_definition]
```

参照约束的语法如下：

```
REFERENCES tbl_name (key_part,...)
```

使用primary key指明主键约束，即不允许为空也不允许重复，从而可以区分两条记录的唯一性。

```
mysql> create table owner (name varchar(80) primary key,sex char(1));
Query OK, 0 rows affected (0.01 sec)

mysql> desc owner;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| name  | varchar(80) | NO   | PRI | NULL    |       |
| sex   | char(1)     | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

也可以在表创建完成之后，使用alter table语句再增加或者删除主键约束：

```
mysql> alter table owner drop primary key;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc owner;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| name  | varchar(80) | NO   |     | NULL    |       |
| sex   | char(1)     | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.01 sec)

mysql> alter table owner add(id int);
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> alter table owner add constraint primary key(id);
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc owner;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| name  | varchar(80) | NO   |     | NULL    |       |
| sex   | char(1)     | YES  |     | NULL    |       |
| id    | int         | NO   | PRI | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

下面的示例中可以设置自动增长的主键：

```
mysql> alter table owner add id int primary key auto_increment;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> insert into owner (name,sex) VALUES ('Diane','f');
Query OK, 1 row affected (0.01 sec)

mysql> insert into owner (name,sex) VALUES ('Gwen','m');
Query OK, 1 row affected (0.00 sec)

mysql> insert into owner (name,sex) VALUES ('Harold','m');
Query OK, 1 row affected (0.00 sec)

mysql> insert into owner (name,sex) VALUES ('Benny','f');
Query OK, 1 row affected (0.00 sec)

mysql> select * from owner;
+--------+------+----+
| name   | sex  | id |
+--------+------+----+
```

```
| Diane   | f    |  1 |
| Gwen    | m    |  2 |
| Harold | m    |  3 |
| Benny   | f    |  4 |
+--------+------+----+
4 rows in set (0.00 sec)
```

备份和恢复数据库

在shell命令行当中使用`mysqldump`命令可以备份数据库或者从文件中恢复数据库。

```
$ mysqldump -u root -p --databases test > test.sql #备份数据库test到文件test.sql中
## 恢复之前需要创建一个同名数据库
$ mysqldump -u root -p < test.sql
```

MySQL的C API

准备工作

mysql除了可以使用命令客户端以外，还以库文件的形式提供了客户端。这样用户就可以自己写代码作为客户端访问mysql服务端。在使用的时候有以下注意事项：

- 首先。需要安装mysql相关的库函数。

```
$ sudo apt install libmysqlclient-dev
```

- 在调用函数之前，需要包含头文件 <mysql/mysql.h>
- 在生成可执行程序的链接阶段时，需要加入链接选项`-lmysqlclient`

相关函数和数据结构

- 下面是相关的函数：

```
MYSQL * mysql_init(MYSQL *mysql);//为MySQL连接分配资源，参数一般填NULL
//数据结构MYSQL是操作资源的句柄

void mysql_close(MYSQL *mysql);
//关闭MYSQL连接

MYSQL *mysql_real_connect(MYSQL *mysql, const char *host, const char *user,
const char *passwd, const char *db, unsigned int port, const char *unix_socket,
unsigned long client_flag);//连接到MySQL服务端
//一般的参数
// host -> "localhost" user -> "root" passwd-> 密码 db->数据库名
// 其余参数选择默认 port -> 0 unix_socket -> NULL client_flag -> 0
// 如果出现报错，返回值为NULL，使用mysql_error函数可以获取报错原因


int mysql_query(MYSQL *mysql, const char *stmt_str);
// 执行SQL语句，stmt_str -> SQL语句的内容 不需要加分号
```

```
MYSQL_RES *mysql_store_result(MYSQL *mysql);
// 在mysql_query之后调用，假如执行的SQL语句会得到结果，需要使用该函数将结果存入数据结构MYSQL_RES当中

void mysql_free_result(MYSQL_RES *result);
// 释放数据结构MYSQL_RES占据的内存空间

my_ulonglong mysql_num_rows(MYSQL_RES *result);
// SQL语句结果的行数

unsigned int mysql_num_fields(MYSQL_RES *result);
// SQL语句结果的列数

MYSQL_ROW mysql_fetch_row(MYSQL_RES *result);
// 从结果当中取出一行
```

- 下面是相关的数据结构

```
//MYSQL 代表了一个MYSQL连接
//MYSQL_RES SQL指令的执行结果，可以理解成一个二维数组，或者是若干行的集合
//MYSQL_ROW 一行，实际上底层是一个字符串数组，每一个字符串值代表了表中一个具体值，无论其在数据库当中是什么数据类型。
//my_ulonglong 用来描述行数的无符号整数
```

一个update的例子

接下来，我们打算对test数据库更新一下数据。这是一个写类型的指令。

```
update pet set sex = 'f' where name = 'Slim'
```

下面是代码示例：

```
#include<func.h>
#include<mysql/mysql.h>
int main(){
    MYSQL *db = mysql_init(NULL);
    char *host = "localhost";
    char *user = "root";
    char *password = "123";
    char *database = "test";
    MYSQL *ret = mysql_real_connect(db,host,user,password,database,0,NULL,0);
    if(ret == NULL){
        printf("Error: %s\n", mysql_error(db));
        return -1;
    }
    char *sql = "update pet set sex = 'f' where name = 'Slim'";
    int qret = mysql_query(db,sql);
    if(qret != 0){
        printf("Error: %s\n", mysql_error(db));
        return -1;
    }
    mysql_close(db);
```

```
}
```

一个select的例子

接下来，我们希望使用select指令展示test数据库内部的数据。这是一个读类型的指令，我们需要采用合适数据结构来展示结果的信息。

```
select * from pet
```

下面是代码：

```
#include<func.h>
#include<mysql/mysql.h>
int main(){
    MYSQL *db = mysql_init(NULL);
    char *host = "localhost";
    char *user = "root";
    char *password = "123";
    char *database = "test";
    MYSQL *ret = mysql_real_connect(db,host,user,password,database,0,NULL,0);
    if(ret == NULL){
        printf("Error: %s\n", mysql_error(db));
        return -1;
    }
    char *sql = "select * from pet";
    int qret = mysql_query(db,sql);
    if(qret != 0){
        printf("Error: %s\n", mysql_error(db));
        return -1;
    }
    MYSQL_RES *result = mysql_store_result(db);
    printf("total rows: %lu\n",mysql_num_rows(result));
    MYSQL_ROW row;
    while((row = mysql_fetch_row(result)) != NULL){
        for(int i = 0; i < mysql_num_fields(result); ++i){
            printf("%s\t", row[i]);
        }
        printf("\n");
    }
    mysql_close(db);
}
```

杂项

避免中文显示乱码和无法输入中文的问题

mysql有六处使用了字符集，分别为：client 、connection、database、results、server 、system。

- client是客户端使用的字符集。
- connection是连接数据库的字符集设置类型，如果程序没有指明连接数据库使用的字符集类型就按照服务器端默认的字符集设置。

- database是数据库服务器中某个库使用的字符集设定，如果建库时没有指明，将使用服务器安装时指定的字符集设置。
- results是数据库给客户端返回时使用的字符集设定，如果没有指明，使用服务器默认的字符集。
- server是服务器安装时指定的默认字符集设定。
- system是数据库系统使用的字符集设定。

下面是一个修改字符集为gbk或utf8的例子:

```
show variables like '%character%';
+--------------------------+----------------------------+
| Variable_name            | Value                      |
+--------------------------+----------------------------+
| character_set_client     | utf8                       |
| character_set_connection | utf8                       |
| character_set_database   | latin1                     |
| character_set_filesystem | binary                     |
| character_set_results    | utf8                       |
| character_set_server     | latin1                     |
| character_set_system     | utf8                       |
| character_sets_dir       | /usr/share/mysql/charsets/ |
+--------------------------+----------------------------+

set character_set_server = utf8;
set character_set_database = utf8;
之后新建的所有数据库就可以插入中文数据了。
```

修改mysql配置使其支持远程连接

- 8.0设置支持远程连接

输入命令mysql -u root -p进入mysql-cli，然后输入：

```
use mysql;

select Host,User from user; #看一下原有的配置

update user set Host='%' where User='root';

flush privileges;
```

- Mysql5.7设置远程连接

进入mysql-cli之后，输入：

```
GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY 'password' WITH GRANT
OPTION;

FLUSH PRIVILEGES;
```

升级mysql之后出现问题

从5.7升级到8.0之后有可能会出现部分命令由于没有权限而无法正常使用（比如 show databases;），下面是解决方案：

```
delete from mysql.user where User = 'mysql.infoschema';
create user 'mysql.infoschema'@'localhost' identified by '123';
update mysql.user set Select_priv = 'Y' where 'mysql.infoschema';
flush privileges;
```

重置mysql的登录信息

停止 MySQL 服务端：

```
sudo service mysql stop
```

启动 MySQL 服务器以生成新的密码文件(在命令行中使用 `mysqld` 命令启动 MySQL 服务器，并使用 `--initialize-insecure` 参数生成新的密码文件。这将生成一个空密码的 root 用户，并且密码文件将被重置)

```
sudo mysqld --initialize-insecure
# 有些操作系统上会提示生成的新密码，如果没有提示那么密码就是空字符串
# 假如操作失败，需要执行下列指令
sudo rm -r /var/lib/mysql # 该操作请务必小心！
sudo mkdir -p /var/lib/mysql
sudo chown mysql:mysql /var/lib/mysql
```

启动 MySQL 服务器：** 使用 `mysqld` 命令启动 MySQL 服务器。在启动时，新的密码文件将被加载。例如，在 Ubuntu 上，可以执行以下命令：

```
sudo service mysql start
```

连接到 MySQL 服务器并更改密码： 现在就可以使用生成的临时密码(活着是空密码)连接到 MySQL 服务器，并更改 root 用户的密码。