# 数据库概念

数据管理技术经历多年的发展，已经发展到**数据库系统阶段**，在该阶段会把数据存储到**数据库（DataBase DB）**中，即**数据库相当于存储数据仓库**。为了便于用户组织和管理数据，其还专门提供了**数据库管理系统（DataBase Management System DBMS），可以有效管理存储在数据库中的数据**。常见的MySQL和Oracle就是两款优秀的数据库管理系统。

## 数据库中的基本概念

在目前阶段，如果要存储 和管理数据，则离不开数据库。当数据存储到数据库后，就会通过数据库管理系统对这些数据进行组织和管理，下面来看一下学习数据库过程中的一些基本概念。

### 数据库管理技术的发展阶段

所谓数据管理，是指对各种数据进行分类、组织、编码、存储、检索和维护。发展到现在，数据库管理技术经历了三个阶段，分别为人工管理阶段、文件系统阶段和数据库系统阶段。

#### 人工管理阶段

20世纪50年代中期以前，由于计算机中硬件还没有像现在这样的磁盘、软件没有专门管理数据的软件，所以计算机只局限于科学技术方面，数据则由计算和处理它的程序自行携带。该时期被称为人工管理阶段。人工管理阶段的特点如下：

* 数据不能长期保存。
* 程序本身管理数据。
* 数据不能共享。
* 数据不具备独立性。

数据的独立性是数据库系统的最基本的特征之一。数据独立性是指应用程序和数据结构之间相互独立、互不影响。在三层模式体系结构中数据独立性是指数据库系统在某一层次模式上的改变不会使它的上一层模型也发生改变得能力。正是三级模式之间的两层映像保证了数据库系统中的数据具有较高的数据独立性。数据独立性包括数据逻辑性和数据物理独立性。

物理独立性是指用户的应用程序与存储在磁盘上的数据库中数据是相互独立的。即数据在磁盘上怎样存储由DBMS管理，用户程序不需要了解，应用程序要处理的只是数据的逻辑结构，这样当数据的物理存储改变了，应用程序也不用改变。

逻辑独立性是指用户的应用程序与数据库的逻辑结构是相互独立的，即当数据的逻辑结构改变时，用户程序也可以不变。逻辑数据独立性比物理数据独立性更难做到，这是因为应用程序对于它们所访问的数据的逻辑结构依赖程度很大。

#### 文件系统阶段

随着技术的发展，在20世纪50年代后期到20世纪60年代中期，计算机不仅应用于科学技术，而且开始用于管理。在该时期由于计算机硬件出现了磁盘，计算机软件出现了高级语言和操作系统，因此程序和数据有了一定独立性，出现了程序文件和数据文件，这就是所谓的文件系统阶段。文件系统阶段的特点如下：

* 数据可以长期保存。
* 数由有文件系统来管理。
* 数据冗（rong）余大，共享性差。冗余即数据文件有相同重复的内容。
* 数据独立性差。

#### 数据库系统阶段

随着网络技术的发展，计算机软/硬件功能的进步，在20世纪60年代后期，计算机可以管理规模巨大的数据，这时如果计算机还使用文件系统来管理数据，则远远不能满足当时各种应用需求，于是出现了数据库技术，特别是关系型数据库技术。该阶段也就是所谓的数据库系统阶段。数据库系统阶段的优点如下：

* 数据实现结构化。
* 数据实现了共享性。
* 数据独立性强。
* 数据粒度变小。

而数据库系统阶段细分又可分为层次型和网状型数据库系统阶段、关系型数据库阶段和非关系型数据库阶段。

##### 3.1 网状数据库系统阶段

网状数据库(network database)是采用网状原理和方法，以网状数据模型为基础建立的数据库。一般是指由网状数据库管理系统产生的网状数据库系统。网状数据模型是以记录类型为结点的[网络结构](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E7%BB%93%E6%9E%84/8259917" \t "https://baike.baidu.com/item/%E7%BD%91%E7%8A%B6%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)，即一个结点可以有一个或多个下级结点，也可以有一个或多个上级结点，两个结点之间甚至可以有多种联系(称之为复合联系)，例如“教师”与“课程”两个记录类型，可以有“任课”和“辅导”两种联系，称之为复合链。两个记录类型之间的值可以是多对多的联系，例如一门课程被多个学生修读，一个学生选修多门课程。得益于复合联系网状数据模型可以更直接地去描述现实世界。而层次结构实际上是网状结构的一个特例。但是，和[层次数据库](https://baike.baidu.com/item/%E5%B1%82%E6%AC%A1%E6%95%B0%E6%8D%AE%E5%BA%93/10915947" \t "https://baike.baidu.com/item/%E7%BD%91%E7%8A%B6%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)一样，网状数据库的操作语言也是过程性的，数据的逻辑独立性仍然不高。

网状数据库系统在20 世纪 70 年代与 80 年代初非常流行，在数据库系统产品中占主导地位，虽然近年来逐渐被关系数据库系统取代，在美国、加拿大等国家 ，由于历史原因 ，网状数据库的用户数仍然很多。如富士通公司M系列机上配制AIM系统，UNIVAC上配制的DMS1100，HONEYWELL公司机器上配制的IDS系统和CINCON的TOTAL等系统都是网状方法。在网状数据库系统中，有许多系统是以美国的DBTG为标准实现的。在DBTG系统中，数据库的记录类型用结点表示，记录类型之间的联系用有向边表示，称为系型，则数据库的全局逻辑结构为一有向图。记录类型由数据项和数据集组成的命名集合。系型定义为记录类型之间命名的联系，系型是一个二级树。一个系型有且仅有一个记录类型定义为首记录类型。其它记录类型为成员记录类型。根据系型的结构特点可划分为三种类型：单成员系、多成员系和奇异系。

层次型[数据库管理系统](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F" \t "https://baike.baidu.com/item/%E5%B1%82%E6%AC%A1%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)是紧随[网状数据库](https://baike.baidu.com/item/%E7%BD%91%E7%8A%B6%E6%95%B0%E6%8D%AE%E5%BA%93/463319" \t "https://baike.baidu.com/item/%E5%B1%82%E6%AC%A1%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)而出现的。现实世界中很多事物是按层次组织起来的。层次数据模型的提出，首先是为了模拟这种按层次组织起来的事物。

[层次数据库](https://baike.baidu.com/item/%E5%B1%82%E6%AC%A1%E6%95%B0%E6%8D%AE%E5%BA%93" \t "https://baike.baidu.com/item/%E5%B1%82%E6%AC%A1%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)也是按记录来存取数据的。层次数据模型中最基本的数据关系是基本层次关系，它代表两个记录型之间一对多的关系，也叫做双亲子女关系（PCR）。数据库中有且仅有一个记录型无双亲，称为根节点。其他记录型有且仅有一个双亲。在[层次模型](https://baike.baidu.com/item/%E5%B1%82%E6%AC%A1%E6%A8%A1%E5%9E%8B" \t "https://baike.baidu.com/item/%E5%B1%82%E6%AC%A1%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)中从一个节点到其双亲的映射是惟一的，所以对每一个记录型（除根节点外）只需要指出它的双亲，就可以表示出层次模型的整体结构。[层次模型](https://baike.baidu.com/item/%E5%B1%82%E6%AC%A1%E6%A8%A1%E5%9E%8B" \t "https://baike.baidu.com/item/%E5%B1%82%E6%AC%A1%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)是树状的。最著名最典型的[层次数据库](https://baike.baidu.com/item/%E5%B1%82%E6%AC%A1%E6%95%B0%E6%8D%AE%E5%BA%93" \t "https://baike.baidu.com/item/%E5%B1%82%E6%AC%A1%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)系统是IBM公司的[IMS](https://baike.baidu.com/item/IMS" \t "https://baike.baidu.com/item/%E5%B1%82%E6%AC%A1%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)（Information Management System），这是IBM公司研制的最早的大型数据库系统程序产品。从60年代末产生起，如今已经发展到IMSV6，提供群集、N路数据共享、[消息队列](https://baike.baidu.com/item/%E6%B6%88%E6%81%AF%E9%98%9F%E5%88%97" \t "https://baike.baidu.com/item/%E5%B1%82%E6%AC%A1%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)共享等先进特性的支持。这个具有30年历史的数据库产品在如今的WWW应用连接、商务智能应用中扮演着新的角色。

##### 3.2 关系型数据库系统阶段

关系数据库，是建立在关系模型基础上的[数据库](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93/103728" \t "https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)，借助于集合代数等数学概念和方法来处理数据库中的[数据](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE/33305" \t "https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)。现实世界中的各种实体以及实体之间的各种联系均用关系模型来表示。关系模型是由埃德加·科德于1970年首先提出的，并配合“科德十二定律”。现如今虽然对此模型有一些批评意见，但它还是数据存储的传统标准。标准数据查询语言SQL就是一种基于关系数据库的语言，这种语言执行对关系数据库中数据的检索和操作。 关系模型由关系数据结构、[关系操作](https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E6%93%8D%E4%BD%9C/9797727" \t "https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)集合、关系完整性约束三部分组成。

[SQL](https://baike.baidu.com/item/SQL" \t "https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)(Structured Query Language)语言是1974年由Boyce和Chamberlin提出的一种介于[关系代数](https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E4%BB%A3%E6%95%B0" \t "https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)与[关系演算](https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E6%BC%94%E7%AE%97" \t "https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)之间的[结构化查询语言](https://baike.baidu.com/item/%E7%BB%93%E6%9E%84%E5%8C%96%E6%9F%A5%E8%AF%A2%E8%AF%AD%E8%A8%80" \t "https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)，是一个通用的、功能极强的关系型[数据库语言](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E8%AF%AD%E8%A8%80" \t "https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)。

简单说，关系型数据库是由多张能互相联接的二维行列表格组成的数据库。当前主流的关系型数据库有Oracle、DB2、PostgreSQL、Microsoft SQL Server、Microsoft Access、MySQL、浪潮K-DB等。

关系模块中常用的操作包括：数据查询 选择 投影 连接 并 交 差 除 数据操作 增加 删除 修改 查询

**完整性约束包括：**

实体完整性：实体完整性这项规则要求每个数据表都必须有主键，而作为主键的所有字段，其属性必须是独一及非空值。

参照完整性：若属性或属性组F是基本关系R的[外键](https://baike.baidu.com/item/%E5%A4%96%E9%94%AE" \t "https://baike.baidu.com/item/_blank)，它与基本关系S的主键Ks相对应（基本关系R和S不一定是不同的关系），则对于R中的每个元组在F上的值必须为：

（1）空值，F的每个属性值均为空值。

（2）S中某个元组中的主键值（主码值）。

即参照的关系中的属性值必须能够在被参照关系找到或者取空值，否则不符合数据库的[语义](https://baike.baidu.com/item/%E8%AF%AD%E4%B9%89" \t "https://baike.baidu.com/item/_blank)。在实际操作时如更新、删除、插入一个表中的数据，通过参照引用相互关联的另一个表中的数据，来检查对表的数据操作是否正确，不正确则拒绝操作。

用户定义完整性：任何[关系数据库系统](https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E6%95%B0%E6%8D%AE%E5%BA%93%E7%B3%BB%E7%BB%9F" \t "https://baike.baidu.com/item/%E7%94%A8%E6%88%B7%E8%87%AA%E5%AE%9A%E4%B9%89%E5%AE%8C%E6%95%B4%E6%80%A7/_blank)都应该支持实体完整性和参照完整性。除此之外，不同的关系数据库系统根据其应用环境的不同，往往还需要一些特殊的约束条件，用户定义的完整性就是针对某一具体关系数据库的唯一约束条件。它反映某一具体应用所涉及的数据必须满足的语义要求。

不同的应用有着不同的具体要求，这些约束条件就是用户根据需要自己定义的。对于这类完整性，关系模型只提供定义和检验这类完整性的机制，以使用户能够满足自己的需求，而关系模型自身并不去定义任何这类完整性规则  。

示例：某个属性必须取唯一值、某个非主属性也不能取空值的，如右图职工关系中必须给出职工的姓名，就可以要求职工姓名不能取空值)、某个属性只能在某范围内取值(如性别的取值只能取自“男”和“女”)。要求“考查”课的分数以60分或40分计，在用户输入“考查”课的成绩时，要进行检查，以确保满足特定的约束要求。再如年龄属性，如果属于某一个学生主体，则可能要求年龄在17岁到25岁之间，而如果年轻属性属于某一个公司员工主体，则可能要求年龄在18岁到40岁之间等。

全关系系统应该完全支持关系模型的所有特征。关系模型的奠基人[埃德加·科德](https://baike.baidu.com/item/%E5%9F%83%E5%BE%B7%E5%8A%A0%C2%B7%E7%A7%91%E5%BE%B7" \t "https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)具体地给出了全关系系统应遵循的基本准则。

* 准则0：一个关系型的[关系数据库管理系统](https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E6%95%B0%E6%8D%AE%E5%BA%93%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F" \t "https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)必须能完全通过它的关系能力来管理数据库。
* 准则1：信息准则 关系数据库管理系统的所有信息都应该在逻辑一级上用表中的值这一种方法显式的表示。
* 准则2：保证访问准则 依靠表名、主码和列名的组合，保证能以逻辑方式访问关系数据库中的每个[数据项](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E9%A1%B9" \t "https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)。
* 准则3：[空值](https://baike.baidu.com/item/%E7%A9%BA%E5%80%BC" \t "https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)的系统化处理 全关系的[关系数据库管理系统](https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E6%95%B0%E6%8D%AE%E5%BA%93%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F" \t "https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)支持空值的概念，并用系统化的方法处理空值。
* 准则4：基于关系模型的动态的联机[数据字典](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%AD%97%E5%85%B8" \t "https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank) 数据库的描述在逻辑级上和普通数据采用同样的表述方式。
* 准则5：统一的数据子语言 一个关系数据库管理系统可以具有几种语言和多种终端访问方式，但必须有一种语言，它的语句可以表示为严格语法规定的字符串，并能全面的支持各种规则。
* 准则6：视图更新准则 所有理论上可更新的视图也应该允许由系统更新。
* 准则7：高级的插入、修改和删除操作 系统应该对各种操作进行查询优化。
* 准则8：数据的物理独立性 无论数据库的数据在存储表示或访问方法上作任何变化，应用程序和终端活动都保持逻辑上的不变性。
* 准则9： 数据逻辑独立性 当对基本关系进行理论上信息不受损害的任何改变时，应用程序和终端活动都保持逻辑上的不变性。
* 准则10：数据完整的独立性 关系数据库的完整性约束条件必须是用[数据库语言](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E8%AF%AD%E8%A8%80" \t "https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)定义并存储在[数据字典](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%AD%97%E5%85%B8" \t "https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)中的。
* 准则11：分布独立性 [关系数据库管理系统](https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E6%95%B0%E6%8D%AE%E5%BA%93%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F" \t "https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)在引入分布数据或数据重新分布时保持逻辑不变。
* 准则12：无破坏准则 如果一个关系数据库[管理系统](https://baike.baidu.com/item/%E7%AE%A1%E7%90%86%E7%B3%BB%E7%BB%9F" \t "https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93/_blank)具有一个低级语言，那么这个低级语言不能违背或绕过完整性准则。

##### 3.3 后关系数据库技术阶段

由于关系数据库在数据模型、性能、扩展伸缩性上存在问题，所以出现了一些后关系数据库的技术，例如，面向对象数据库技术（ORDBMS）和结构化数据库技术（NOSQL）。

对于ORDBMS技术，虽然其尚未完全成熟，但是能很好的支持对数据和对象的管理，能够很好地和面向对象设计技术相融合。而NOSQL技术，则打破了关系数据库与ACID理论相互统一的局面。利用NOSQL技术存储数据时，不需要固定的表结构，因此在大型数据存取上具备关系型数据库无法比拟的性能优势。

##### 3.4 非关系型数据库系统阶段

NoSQL（Not Only SQL），泛指非关系型的数据库。随着互联网[web2.0](https://baike.baidu.com/item/web2.0/97695" \t "https://baike.baidu.com/item/NoSQL/_blank)网站的兴起，传统的关系数据库在应付web2.0网站，特别是超大规模和高并发的[SNS](https://baike.baidu.com/item/SNS/10242" \t "https://baike.baidu.com/item/NoSQL/_blank)类型的web2.0纯[动态网](https://baike.baidu.com/item/%E5%8A%A8%E6%80%81%E7%BD%91" \t "https://baike.baidu.com/item/NoSQL/_blank)站已经显得力不从心，暴露了很多难以克服的问题，而非关系型的数据库则由于其本身的特点得到了非常迅速的发展。NoSQL数据库的产生就是为了解决大规模数据集合多重数据种类带来的挑战，尤其是大数据应用难题。NoSQL数据库的四大分类如下：

键值([Key-Value](https://baike.baidu.com/item/Key-Value" \t "https://baike.baidu.com/item/NoSQL/_blank))存储[数据库](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93" \t "https://baike.baidu.com/item/NoSQL/_blank)：这一类数据库主要会使用到一个[哈希表](https://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E8%A1%A8" \t "https://baike.baidu.com/item/NoSQL/_blank)，这个表中有一个特定的键和一个指针指向特定的数据。Key/value模型对于IT系统来说的优势在于简单、易部署。但是如果[DBA](https://baike.baidu.com/item/DBA/3349" \t "https://baike.baidu.com/item/NoSQL/_blank)只对部分值进行查询或更新的时候，Key/value就显得效率低下了。举例如：Tokyo Cabinet/Tyrant, Redis, Voldemort, Oracle BDB.

列存储数据库：这部分数据库通常是用来应对分布式存储的海量数据。键仍然存在，但是它们的特点是指向了多个列。这些列是由列家族来安排的。如：Cassandra, HBase, Riak.

文档型数据库：文档型数据库的灵感是来自于Lotus Notes办公软件的，而且它同第一种键值存储相类似。该类型的数据模型是版本化的文档，半结构化的文档以特定的格式存储，比如JSON。文档型数据库可 以看作是键值数据库的升级版，允许之间嵌套键值。而且文档型数据库比键值数据库的查询效率更高。如：CouchDB, MongoDb. 国内也有文档型数据库SequoiaDB，已经开源。

图形(Graph)数据库：图形结构的数据库同其他行列以及刚性结构的SQL数据库不同，它是使用灵活的图形模型，并且能够扩展到多个服务器上。NoSQL数据库没有标准的查询语言(SQL)，因此进行数据库查询需要制定数据模型。许多NoSQL数据库都有REST式的数据接口或者查询API。如：Neo4J, InfoGrid, Infinite Graph.

因此，我们总结NoSQL数据库在以下的这几种情况下比较适用：1、数据模型比较简单；2、需要灵活性更强的IT系统；3、对数据库性能要求较高；4、不需要高度的数据一致性；5、对于给定key，比较容易映射复杂值的环境。

### 数据库系统阶段涉及的概念

现阶段对于数据的出来仍然离不开数据库，在该阶段处理数据时，经常会涉及各种概念：数据库、数据库管理系统和数据库系统。同时如果想玩全掌握数据库系统阶段的数据处理技术，也必须掌握这些概念。

数据库(DataBase DB):是指长期保存在计算机的存储设备上,按照一定规则组织起来,可以被各种用户或应用共享的数据集合;

数据库管理系统(DataBase Management System DBMS):是指一种操作和管理数据库的大型软件,用于建立、作用和维护数据库，对数据库进行统一管理和控制，以保证数据库的安全性和完整性。用户通过数据库管理系统访问数据库中的数据。当前比较流行的数据库管理系统有MySQL、Oracle、SQL Server和DB2等。

数据库系统（DataBase System DBS）：是指在计算机系统中引入数据库之后的系统，通常由计算机硬件、软件、数据库管理系统和数据管理员构成。

在通常情况下，经常会用数据库来表示它们使用的数据库软件。这经常会引起混淆，确切的说，数据库软件应该为数据库管理系统，数据库是通过数据库管理系统创建和操作的容器。

数据库存储数据的优势：

* 可存储大量数据；
* 方便检索；
* 保持数据的一致性、完整性；
* 安全，可共享；
* 通过组合分析，可产生新数据。

### 数据库管理系统提供的功能

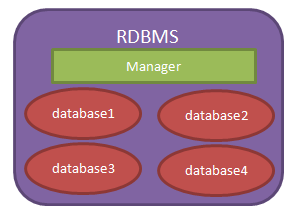
数据库管理系统提供了许多功能，可以通过SQL（结构化查询语言）来定义和操作数据，维护数据的完整性和安全性，以及进行各种数据库的管理等。数据库管理系统主要支持三种SQL功能。

数据定义语言（Data Definition Language DDL）数据库管理系统提供了数据定义语言定义数据库涉及各种对象，定义数据的完整性约束、保密限制等约束。通俗来讲就是对数据库对象的增删改查等操作。

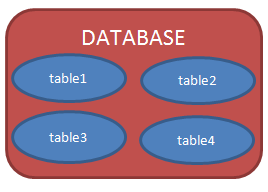
数据操作语言（Data Manipulation Language DML）数据库管理系统

### 1.5　理解数据库

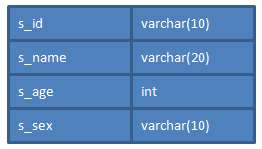
我们现在所说的数据库泛指关“关系型数据库管理系统（RDBMS - Relational database management system）”，即“数据库服务器”。



当我们安装了数据库服务器后，就可以在数据库服务器中创建数据库，每个数据库中还可以包含多张表。



数据库表就是一个多行多列的表格。在创建表时，需要指定表的列数，以及列名称，列类型等信息。而不用指定表格的行数，行数是没有上限的。下面是tab\_student表的结构：



当把表格创建好了之后，就可以向表格中添加数据了。向表格添加数据是以行为单位的！下面是s\_student表的记录：

|  |  |  |  |
| --- | --- | --- | --- |
| **s\_id** | **s\_name** | **s\_age** | **s\_sex** |
| **S\_1001** | zhangSan | 23 | male |
| **S\_1002** | liSi | 32 | female |
| **S\_1003** | wangWu | 44 | male |

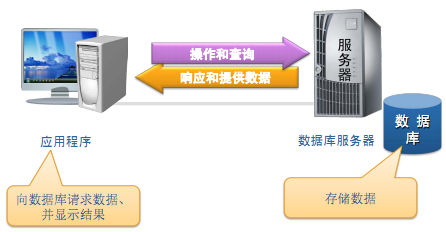
大家要学会区分什么是表结构，什么是表记录。

#### 1.5.1 表结构

  数据表是由表名、表中的字段和表的记录三个部分组成的。设计数据表结构就是定义数据表文件名，确定数据表包含哪些字段，各字段的字段名、字段类型、及宽度，并将这些数据输入到计算机当中。

### 1.6　应用程序与数据库

　　应用程序使用数据库完成对数据的存储！

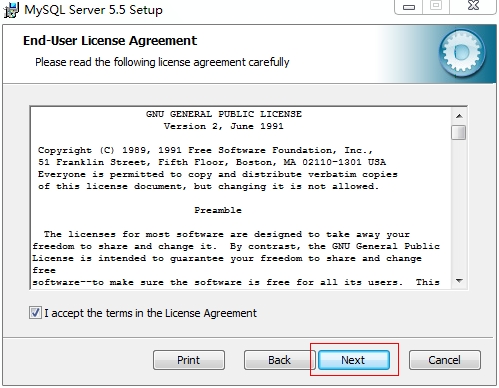
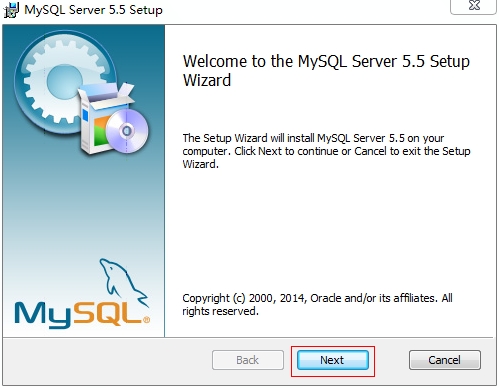


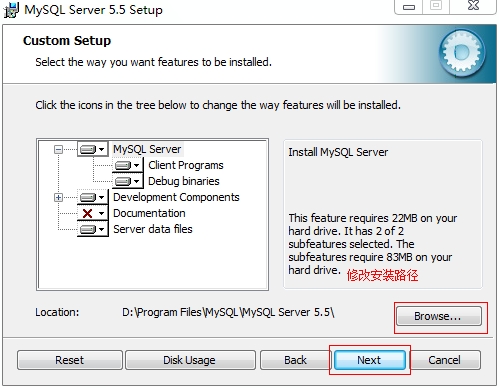
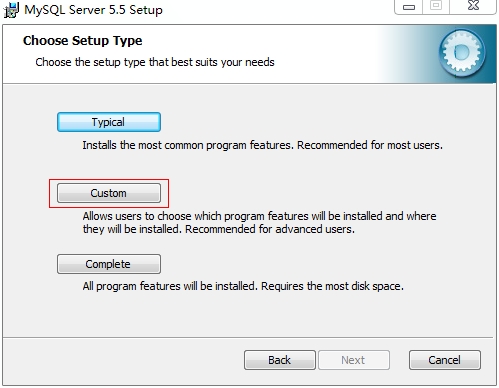
# MySQL

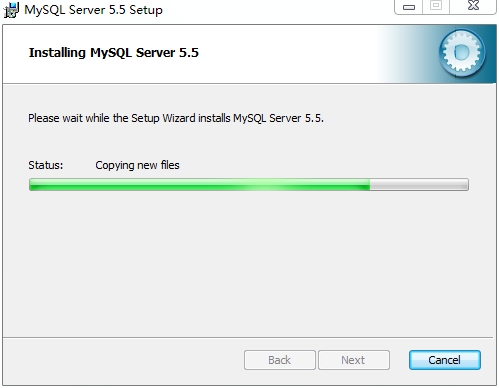
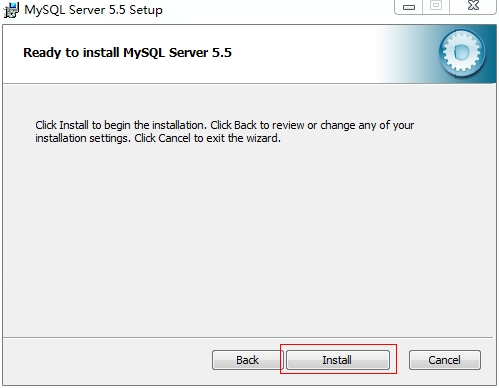
## 1　安装MySQL数据库

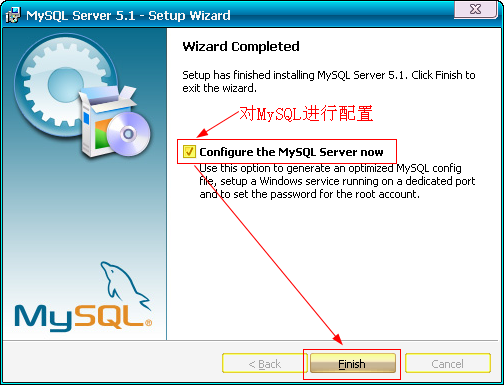
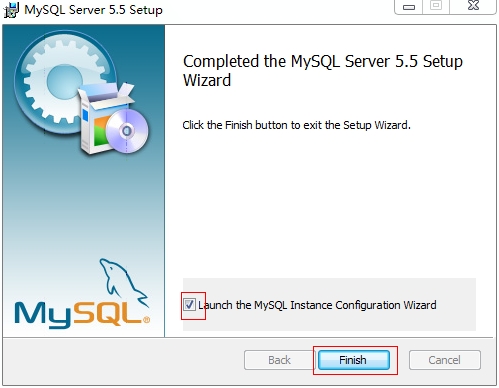
### 1.1　安装MySQL

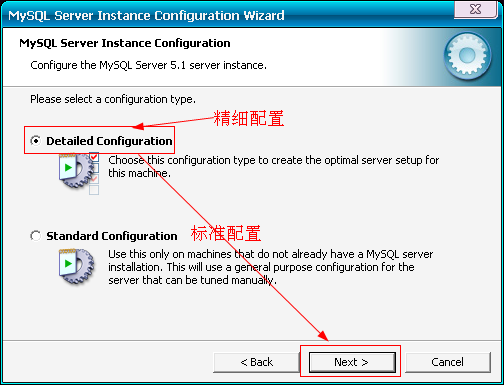
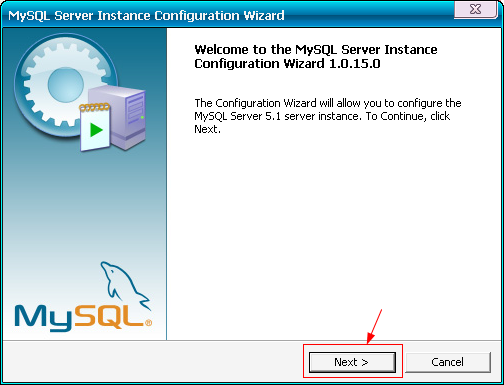
参考：MySQL安装图解.doc

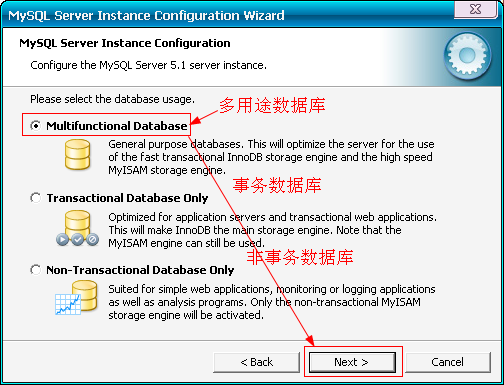
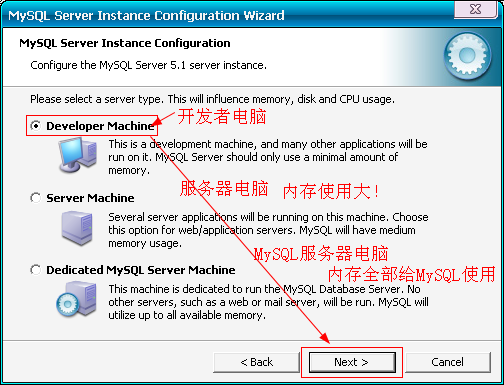


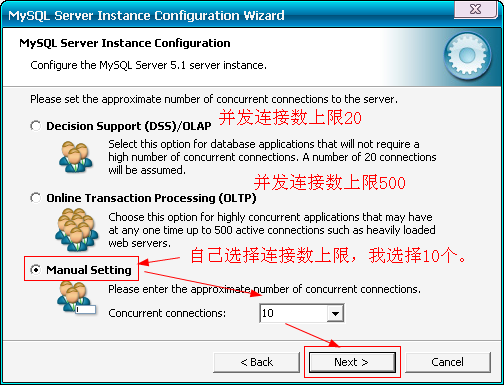
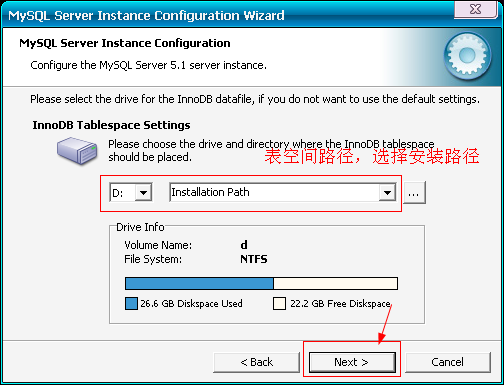


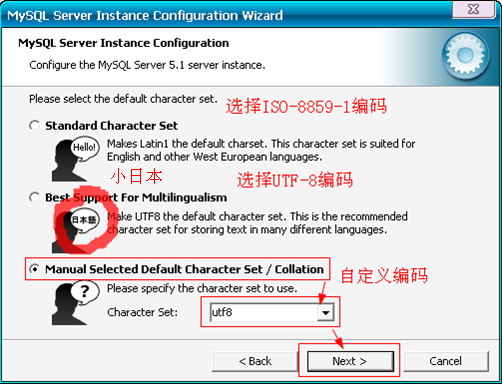
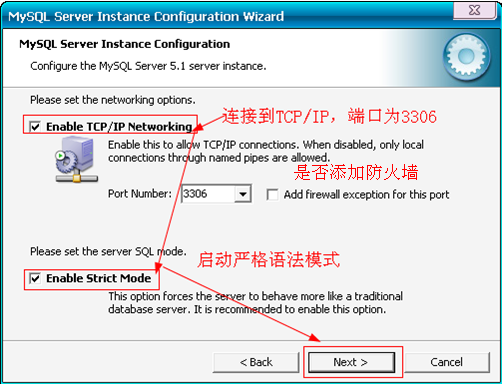


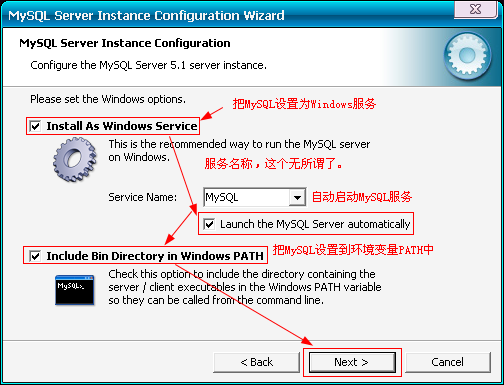


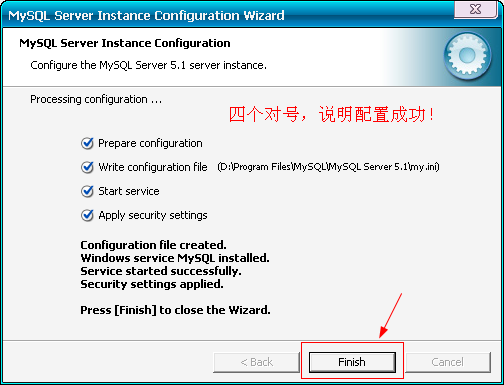
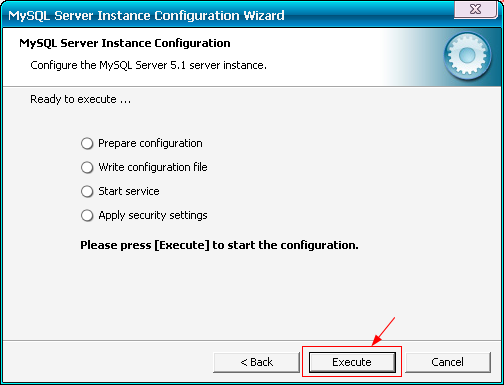




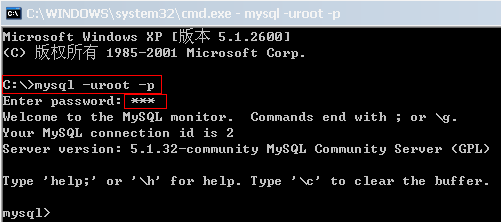








校验



1　安装MySQL

2　校验MySQL

登录MySQL：mysql -uroot -p123

退出MySQL：exit | quit

查看数据库：show databases;

### 1.2　MySQL目录结构

MySQL的数据存储目录为data，data目录通常在C:\Documents and Settings\All Users\Application Data\MySQL\MySQL Server 5.1\data位置。在data下的每个目录都代表一个数据库。

MySQL的安装目录下：

* bin目录中都是可执行文件；
* my.ini文件是MySQL的配置文件；

## 2　基本命令

### 2.1　启动和关闭mysql服务器

* 启动：net start mysql

关闭：net stop mysql（注意这里都不加”;”）

在启动mysql服务后，打开windows任务管理器，会有一个名为mysqld.exe的进程运行，所以mysqld.exe才是MySQL服务器程序。

### 2.2　客户端登录退出mysql

在启动MySQL服务器后，我们需要使用管理员用户登录MySQL服务器，然后来对服务器进行操作。登录MySQL需要使用MySQL的客户端程序：mysql.exe

登录：mysql -u root -p 123 -h localhost（注意不要加”;”）

* -u：后面的root是用户名，这里使用的是超级管理员root；
* -p：后面的123是密码，这是在安装MySQL时就已经指定的密码；
* -h：后面给出的localhost是服务器主机名，它是可以省略的，例如：mysql -u root -p 123；
* 退出：quit或exit；

在登录成功后，打开windows任务管理器，会有一个名为mysql.exe的进程运行，所以mysql.exe是客户端程序。

## 3 SQL语句

### 3.1 SQL概述

#### 3.1.1　什么是SQL

SQL（Structured Query Language）是“结构化查询语言”，它是对关系型数据库的操作语言。它可以应用到所有关系型数据库中，例如：MySQL、Oracle、SQL Server等。SQ标准（ANSI/ISO）有：

* SQL-92：1992年发布的SQL语言标准；
* SQL:1999：1999年发布的SQL语言标签；
* SQL:2003：2003年发布的SQL语言标签；

这些标准就与JDK的版本一样，在新的版本中总要有一些语法的变化。不同时期的数据库对不同标准做了实现。

虽然SQL可以用在所有关系型数据库中，但很多数据库还都有标准之后的一些语法，我们可以称之为“方言”。例如MySQL中的LIMIT语句就是MySQL独有的方言，其它数据库都不支持！当然，Oracle或SQL Server都有自己的方言。

#### 3.1.2　语法要求

* SQL语句可以单行或多行书写，以分号结尾；
* 可以用空格和缩进来来增强语句的可读性；
* 关键字不区别大小写，建议使用大写；

### 3.2 分类

* DDL（Data Definition Language）：数据定义语言，用来定义数据库对象：库、表、列等；
* DML（Data Manipulation Language）：数据操作语言，用来定义数据库记录（数据）；
* DCL（Data Control Language）：数据控制语言，用来定义访问权限和安全级别；
* DQL（Data Query Language）：数据查询语言，用来查询记录（数据）。

### 3.3　DDL

#### 3.3.1　基本操作

* 查看所有数据库名称：SHOW DATABASES；
* 切换数据库：USE mydb1，切换到mydb1数据库；

#### 3.3.2　操作数据库

* 创建数据库：CREATE DATABASE [IF NOT EXISTS] mydb1；

创建数据库，例如：CREATE DATABASE mydb1，创建一个名为mydb1的数据库。如果这个数据已经存在，那么会报错。例如CREATE DATABASE IF NOT EXISTS mydb1，在名为mydb1的数据库不存在时创建该库，这样可以避免报错。

* 删除数据库：DROP DATABASE [IF EXISTS] mydb1；

删除数据库，例如：DROP DATABASE mydb1，删除名为mydb1的数据库。如果这个数据库不存在，那么会报错。DROP DATABASE IF EXISTS mydb1，就算mydb1不存在，也不会的报错。

* 修改数据库编码：ALTER DATABASE mydb1 CHARACTER SET utf8

修改数据库mydb1的编码为utf8。注意，在MySQL中所有的UTF-8编码都不能使用中间的“-”，即UTF-8要书写为UTF8。

#### 3.3.3　数据类型

MySQL与Java一样，也有数据类型。MySQL中数据类型主要应用在列上。

常用类型：

* int：整型
* double：浮点型，例如double(5,2)表示最多5位，其中必须有2位小数，即最大值为999.99；
* decimal：泛型型，在表单钱方面使用该类型，因为不会出现精度缺失问题；
* char：固定长度字符串类型；
* varchar：可变长度字符串类型；
* text：字符串类型；
* blob：字节类型；
* date：日期类型，格式为：yyyy-MM-dd；
* time：时间类型，格式为：hh:mm:ss
* timestamp：时间戳类型；

#### 3.3.4　操作表

* 创建表：

CREATE TABLE 表名(

列名 列类型,

列名 列类型,

......

);

例如：

|  |
| --- |
| **CREATE TABLE stu(**  **sid CHAR(6),**  **sname VARCHAR(20),**  **age INT,**  **gender VARCHAR(10)**  **);** |

再例如：

|  |
| --- |
| **CREATE TABLE emp(**  **eid CHAR(6),**  **ename VARCHAR(50),**  **age INT,**  **gender VARCHAR(6),**  **birthday DATE,**  **hiredate DATE,**  **salary DECIMAL(7,2),**  **resume VARCHAR(1000)**  **);** |

* 查看当前数据库中所有表名称：SHOW TABLES；
* 查看指定表的创建语句：SHOW CREATE TABLE emp，查看emp表的创建语句；
* 查看表结构：DESC emp，查看emp表结构；
* 删除表：DROP TABLE emp，删除emp表；
* 修改表：

1. 修改之添加列：给stu表添加classname列：

ALTER TABLE stu ADD (classname varchar(100));

1. 修改之修改列类型：修改stu表的gender列类型为CHAR(2)：

ALTER TABLE stu MODIFY gender CHAR(2);

1. 修改之修改列名：修改stu表的gender列名为sex：

ALTER TABLE stu change gender sex CHAR(2);

1. 修改之删除列：删除stu表的classname列：

ALTER TABLE stu DROP classname;

1. 修改之修改表名称：修改stu表名称为student：

ALTER TABLE stu RENAME TO student;

### 3.4　DML

#### 3.4.1　插入数据

语法：

INSERT INTO 表名(列名1,列名2, …) VALUES(值1, 值2)

|  |
| --- |
| **INSERT INTO stu(sid, sname,age,gender) VALUES('s\_1001', 'zhangSan', 23, 'male');** |
| 因为没有插入age和gender列的数据，所以该条记录的age和gender值上为NULL  **INSERT INTO stu(sid, sname) VALUES('s\_1001', 'zhangSan');** |

语法：

INSERT INTO 表名 VALUES(值1,值2,…)

因为没有指定要插入的列，表示按创建表时列的顺序插入所有列的值：

|  |
| --- |
| **INSERT INTO stu VALUES('s\_1002', 'liSi', 32, 'female');** |

　　注意：所有字符串数据必须使用单引用！

#### 3.4.2　修改数据

语法：

UPDATE 表名 SET 列名1=值1, … 列名n=值n [WHERE 条件]

|  |
| --- |
| **UPDATE stu SET sname=’zhangSanSan’, age=’32’, gender=’female’ WHERE sid=’s\_1001’;** |
| **UPDATE stu SET sname=’liSi’, age=’20’ WHERE age>50 AND gender=’male’;** |
| **UPDATE stu SET sname=’wangWu’, age=’30’ WHERE age>60 OR gender=’female’;** |
| **UPDATE stu SET gender=’female’ WHERE gender IS NULL**  **UPDATE stu SET age=age+1 WHERE sname=’zhaoLiu’;** |

#### 3.4.3　删除数据

语法：

DELETE FROM 表名 [WHERE 条件]

|  |
| --- |
| **DELETE FROM stu WHERE sid=’s\_1001’003B** |
| **DELETE FROM stu WHERE sname=’chenQi’ OR age > 30;** |
| 删除表所有记录  **DELETE FROM stu;** |

语法：

TRUNCATE TABLE 表名

|  |
| --- |
| 删除stu表，再创建stu表。等同与删除stu表所有记录。  **TRUNCATE TABLE stu;** |

虽然TRUNCATE和DELETE都可以删除表的所有记录，但有原理不同。DELETE的效率没有TRUNCATE高！

TRUNCATE其实属性DDL语句，因为它是先DROP TABLE，再CREATE TABLE。而且TRUNCATE删除的记录是无法回滚的，但DELETE删除的记录是可以回滚的（回滚是事务的知识！）。

### 3.5　DCL

#### 3.5.1　创建用户

语法：

CREATE USER 用户名@地址 IDENTIFIED BY '密码';

|  |
| --- |
| user1用户只能在localhost这个IP登录mysql服务器  **CREATE USER user1@localhost IDENTIFIED BY ‘123’;** |
| user2用户可以在任何电脑上登录mysql服务器  **CREATE USER user2@’%’ IDENTIFIED BY ‘123’;** |

#### 3.5.2　给用户授权

　　语法：

GRANT 权限1, … , 权限n ON 数据库.\* TO 用户名

|  |
| --- |
| **GRANT CREATE,ALTER,DROP,INSERT,UPDATE,DELETE,SELECT ON mydb1.\* TO user1@localhost;** |
| **GRANT ALL ON mydb1.\* TO user2@localhost;** |

#### 3.5.3　撤销授权

　　语法：

　　REVOKE权限1, … , 权限n ON 数据库.\* FORM 用户名

|  |
| --- |
| **REVOKE CREATE,ALTER,DROP ON mydb1.\* FROM user1@localhost;** |

#### 3.5.4　查看用户权限

语法：

SHOW GRANTS FOR 用户名

|  |
| --- |
| **SHOW GRANTS FOR user1@localhost;** |

#### 3.5.5　删除用户

语法：

DROP USER 用户名

|  |
| --- |
| **DROP USER user1@localhost;** |

#### 3.5.6　修改用户密码

语法：

USE mysql;

UPDATE USER SET PASSWORD=PASSWORD(‘密码’) WHERE User=’用户名’ and Host=’IP’;

FLUSH PRIVILEGES;

|  |
| --- |
| **UPDATE USER SET PASSWORD=PASSWORD('1234') WHERE User='user2' and Host=’localhost’;**  **FLUSH PRIVILEGES;** |

### 3.6 数据查询语法（DQL)

　　DQL就是数据查询语言，数据库执行DQL语句不会对数据进行改变，而是让数据库发送结果集给客户端。

语法：

SELECT selection\_list /\*要查询的列名称\*/

FROM table\_list /\*要查询的表名称\*/

WHERE condition /\*行条件\*/

GROUP BY grouping\_columns /\*对结果分组\*/

HAVING condition /\*分组后的行条件\*/

ORDER BY sorting\_columns /\*对结果分组\*/

LIMIT offset\_start, row\_count /\*结果限定\*/

创建名：

* 学生表：stu

|  |  |  |
| --- | --- | --- |
| **字段名称** | **字段类型** | **说明** |
| **sid** | char(6) | 学生学号 |
| **sname** | varchar(50) | 学生姓名 |
| **age** | int | 学生年龄 |
| **gender** | varchar(50) | 学生性别 |

|  |
| --- |
| CREATE TABLE stu (  sid CHAR(6),  sname VARCHAR(50),  age INT,  gender VARCHAR(50)  ); |
| INSERT INTO stu VALUES('S\_1001', 'liuYi', 35, 'male');  INSERT INTO stu VALUES('S\_1002', 'chenEr', 15, 'female');  INSERT INTO stu VALUES('S\_1003', 'zhangSan', 95, 'male');  INSERT INTO stu VALUES('S\_1004', 'liSi', 65, 'female');  INSERT INTO stu VALUES('S\_1005', 'wangWu', 55, 'male');  INSERT INTO stu VALUES('S\_1006', 'zhaoLiu', 75, 'female');  INSERT INTO stu VALUES('S\_1007', 'sunQi', 25, 'male');  INSERT INTO stu VALUES('S\_1008', 'zhouBa', 45, 'female');  INSERT INTO stu VALUES('S\_1009', 'wuJiu', 85, 'male');  INSERT INTO stu VALUES('S\_1010', 'zhengShi', 5, 'female');  INSERT INTO stu VALUES('S\_1011', 'xxx', NULL, NULL); |

* 雇员表：emp

|  |  |  |
| --- | --- | --- |
| **字段名称** | **字段类型** | **说明** |
| **empno** | int | 员工编号 |
| **ename** | varchar(50) | 员工姓名 |
| **job** | varchar(50) | 员工工作 |
| **mgr** | int | 领导编号 |
| **hiredate** | date | 入职日期 |
| **sal** | decimal(7,2) | 月薪 |
| **comm** | decimal(7,2) | 奖金 |
| **deptno** | int | 部分编号 |

|  |
| --- |
| CREATE TABLE emp(  empno INT,  ename VARCHAR(50),  job VARCHAR(50),  mgr INT,  hiredate DATE,  sal DECIMAL(7,2),  comm decimal(7,2),  deptno INT  ) ; |
| INSERT INTO emp values(7369,'SMITH','CLERK',7902,'1980-12-17',800,NULL,20);  INSERT INTO emp values(7499,'ALLEN','SALESMAN',7698,'1981-02-20',1600,300,30);  INSERT INTO emp values(7521,'WARD','SALESMAN',7698,'1981-02-22',1250,500,30);  INSERT INTO emp values(7566,'JONES','MANAGER',7839,'1981-04-02',2975,NULL,20);  INSERT INTO emp values(7654,'MARTIN','SALESMAN',7698,'1981-09-28',1250,1400,30);  INSERT INTO emp values(7698,'BLAKE','MANAGER',7839,'1981-05-01',2850,NULL,30);  INSERT INTO emp values(7782,'CLARK','MANAGER',7839,'1981-06-09',2450,NULL,10);  INSERT INTO emp values(7788,'SCOTT','ANALYST',7566,'1987-04-19',3000,NULL,20);  INSERT INTO emp values(7839,'KING','PRESIDENT',NULL,'1981-11-17',5000,NULL,10);  INSERT INTO emp values(7844,'TURNER','SALESMAN',7698,'1981-09-08',1500,0,30);  INSERT INTO emp values(7876,'ADAMS','CLERK',7788,'1987-05-23',1100,NULL,20);  INSERT INTO emp values(7900,'JAMES','CLERK',7698,'1981-12-03',950,NULL,30);  INSERT INTO emp values(7902,'FORD','ANALYST',7566,'1981-12-03',3000,NULL,20);  INSERT INTO emp values(7934,'MILLER','CLERK',7782,'1982-01-23',1300,NULL,10); |

* 部分表：dept

|  |  |  |
| --- | --- | --- |
| **字段名称** | **字段类型** | **说明** |
| **deptno** | int | 部分编码 |
| **dname** | varchar(50) | 部分名称 |
| **loc** | varchar(50) | 部分所在地点 |

|  |
| --- |
| CREATE TABLE dept(  deptno INT,  dname varchar(14),  loc varchar(13)  ); |
| INSERT INTO dept values(10, 'ACCOUNTING', 'NEW YORK');  INSERT INTO dept values(20, 'RESEARCH', 'DALLAS');  INSERT INTO dept values(30, 'SALES', 'CHICAGO');  INSERT INTO dept values(40, 'OPERATIONS', 'BOSTON'); |

#### 3.6.1　基础查询

##### 3.6.1.1　查询所有列

**SELECT \* FROM stu;**

##### 3.6.1.2　查询指定列

**SELECT sid, sname, age FROM stu;**

#### 3.6.2　条件查询

##### 3.6.2.1　条件查询介绍

条件查询就是在查询时给出WHERE子句，在WHERE子句中可以使用如下运算符及关键字：

* =、!=、<>、<、<=、>、>=；
* BETWEEN…AND；
* IN(set)；
* IS NULL；
* AND；
* OR；
* NOT；

##### 3.6.2.2　查询性别为女，并且年龄50的记录

**SELECT \* FROM stu**

**WHERE gender='female' AND age<50;**

##### 3.6.2.3　查询学号为S\_1001，或者姓名为liSi的记录

**SELECT \* FROM stu**

**WHERE sid ='S\_1001' OR sname='liSi';**

##### 3.6.2.4　查询学号为S\_1001，S\_1002，S\_1003的记录

//In:范围中有一个符合即被查询出来

**SELECT \* FROM stu**

**WHERE sid IN ('S\_1001','S\_1002','S\_1003');**

##### 3.6.2.5　查询学号不是S\_1001，S\_1002，S\_1003的记录

**SELECT \* FROM tab\_student**

**WHERE s\_number NOT IN ('S\_1001','S\_1002','S\_1003');**

Not IN:范围中有一个不符合就不被查询出来

##### 3.6.2.6　查询年龄为null的记录

**SELECT \* FROM stu**

**WHERE age IS NULL;**//详见下面SQL中null值使用

##### 3.6.2.7　查询年龄在20到40之间的学生记录

**SELECT \***

**FROM stu**

**WHERE age>=20 AND age<=40;**

或者

**SELECT \***

**FROM stu**

**WHERE age BETWEEN 20 AND 40;**

##### 3.6.2.8　查询性别非男的学生记录

**SELECT \***

**FROM stu**

**WHERE gender!='male';**

或者

**SELECT \***

**FROM stu**

**WHERE gender<>'male';**

或者

**SELECT \***

**FROM stu**

**WHERE NOT gender='male';**

##### 3.6.2.9　查询姓名不为null的学生记录

**SELECT \***

**FROM stu**

**WHERE NOT sname IS NULL;**

或者

**SELECT \***

**FROM stu**

**WHERE sname IS NOT NULL;**

#### 3.6.3　模糊查询

当想查询姓名中包含a字母的学生时就需要使用模糊查询了。模糊查询需要使用关键字LIKE。

##### 3.6.3.1　查询姓名由5个字母构成的学生记录

**SELECT \***

**FROM stu**

**WHERE sname LIKE '\_\_\_\_\_';**

模糊查询必须使用LIKE关键字。其中 “\_”匹配任意一个字母，5个“\_”表示5个任意字母。

##### 3.6.3.2　查询姓名由5个字母构成，并且第5个字母为“i”的学生记录

**SELECT \***

**FROM stu**

**WHERE sname LIKE '\_\_\_\_i';**

##### 3.6.3.3　查询姓名以“z”开头的学生记录

**SELECT \***

**FROM stu**

**WHERE sname LIKE 'z%';**

其中“%”匹配0~n个任何字母。

##### 3.6.3.4　查询姓名中第2个字母为“i”的学生记录

**SELECT \***

**FROM stu**

**WHERE sname LIKE '\_i%';**

##### 3.6.3.5　查询姓名中包含“a”字母的学生记录

**SELECT \***

**FROM stu**

**WHERE sname LIKE '%a%';**

#### 3.6.4　字段控制查询

##### 3.6.4.1　去除重复记录

去除重复记录（两行或两行以上记录中系列的上的数据都相同），例如emp表中sal字段就存在相同的记录。当只查询emp表的sal字段时，那么会出现重复记录，那么想去除重复记录，需要使用DISTINCT：

**SELECT** **DISTINCT sal FROM emp;**

##### 3.6.4.2　查看雇员的月薪与佣金之和

　　因为sal和comm两列的类型都是数值类型，所以可以做加运算。如果sal或comm中有一个字段不是数值类型，那么会出错。

**SELECT \*,sal+comm FROM emp;**

comm列有很多记录的值为NULL，因为任何东西与NULL相加结果还是NULL，所以结算结果可能会出现NULL。下面使用了把NULL转换成数值0的函数IFNULL：

**SELECT \*,sal+IFNULL(comm,0) FROM emp;**

##### 3.6.4.3　给列名添加别名

在上面查询中出现列名为sal+IFNULL(comm,0)，这很不美观，现在我们给这一列给出一个别名，为total：

**SELECT \*, sal+IFNULL(comm,0) AS total FROM emp;**

给列起别名时，是可以省略AS关键字的：

**SELECT \*,sal+IFNULL(comm,0) total FROM emp;**

#### 3.6.5　排序

##### 3.6.5.1　查询所有学生记录，按年龄升序排序

(升序:从小到大ASC(默认排序方式,可以省略)

降序:从大到小DESC)

**SELECT \***

**FROM stu**

**ORDER BY sage ASC;**

或者

**SELECT \***

**FROM stu**

**ORDER BY sage;**

##### 3.6.5.2　查询所有学生记录，按年龄降序排序

**SELECT \***

**FROM stu**

**ORDER BY age DESC;**

##### 3.6.5.3　查询所有雇员，按月薪降序排序，如果月薪相同时，按编号升序排序

**SELECT \* FROM emp**

**ORDER BY sal DESC,empno ASC;**

#### 3.6.6　聚合函数

聚合函数是用来做纵向运算的函数：

* COUNT()：统计指定列不为NULL的记录行数；
* MAX()：计算指定列的最大值，如果指定列是字符串类型，那么使用字符串排序运算；
* MIN()：计算指定列的最小值，如果指定列是字符串类型，那么使用字符串排序运算；
* SUM()：计算指定列的数值和，如果指定列类型不是数值类型，那么计算结果为0；
* AVG()：计算指定列的平均值，如果指定列类型不是数值类型，那么计算结果为0；

##### 3.6.6.1　COUNT

当需要纵向统计时可以使用COUNT()。

* 查询emp表中记录数：

**SELECT COUNT(\*) AS cnt FROM emp;**

查询emp表中有佣金的人数(佣金为0也算是有佣金)：

**SELECT COUNT(comm) cnt FROM emp;**

注意，因为count()函数中给出的是comm列，那么只统计comm列非NULL的行数。

* 查询emp表中月薪大于2500的人数：

**SELECT COUNT(\*) FROM emp**

**WHERE sal > 2500;**

* 统计月薪与佣金之和大于2500元的人数：

**SELECT COUNT(\*) AS cnt FROM emp WHERE sal+IFNULL(comm,0) > 2500;**

* 查询有佣金的人数，以及有领导的人数：

两个查询是独立的互相不干扰

**SELECT COUNT(comm), COUNT(mgr) FROM emp;**

##### 3.6.6.2　SUM和AVG

当需要纵向求和时使用sum()函数。

* 查询所有雇员月薪和：

**SELECT SUM(sal) FROM emp;**

* 查询所有雇员月薪和，以及所有雇员佣金和：

会自动将null值转换为0进行操作

**SELECT SUM(sal), SUM(comm) FROM emp;**

* 查询所有雇员月薪+佣金和：

**SELECT SUM(sal+IFNULL(comm,0)) FROM emp;**

* 统计所有员工平均工资：

**SELECT SUM(sal)/COUNT(sal) FROM emp;**

或者

**SELECT AVG(sal) FROM emp;**

* 统计所有佣金的人的平均佣金：

当AVG对一列进行操作时会对这一列中不为null的项进行操作,其实底层操作就是Select sum(comm)/count(comm) from emp;

**SELECT AVG(comm) FROM emp;**

##### 3.6.6.3　MAX和MIN

* 查询最高工资和最低工资：

**SELECT MAX(sal), MIN(sal) FROM emp;**

#### 3.6.7　分组查询

当需要分组查询时需要使用GROUP BY子句，例如查询每个部门的工资和，这说明要使用部分来分组。

##### 3.6.7.1　分组查询

* 查询每个部门的部门编号和每个部门的工资和：

**SELECT deptno, SUM(sal)**

**FROM emp**

**GROUP BY deptno;**

* 查询每个部门的部门编号以及每个部门的人数：

**SELECT deptno,COUNT(\*)**

**FROM emp**

**GROUP BY deptno;**

* 查询每个部门的部门编号以及每个部门工资大于1500的人数：

**SELECT deptno,COUNT(\*)**

**FROM emp**

**WHERE sal>1500**

**GROUP BY deptno;**

##### 3.6.7.2　HAVING子句

* 查询工资总和大于9000的部门编号以及工资和：

**SELECT deptno, SUM(sal)**

**FROM emp**

**GROUP BY deptno**

**HAVING SUM(sal) > 9000;**

注意，WHERE是对分组前记录的条件，如果某行记录没有满足WHERE子句的条件，那么这行记录不会参加分组；而HAVING是对分组后数据的约束。<WHERE是对分组前记录的条件,而HAVING是对分组后数据的约束>

#### 3.6.8　LIMIT(MySQL的方言)

LIMIT用来限定查询结果的起始行，以及总行数。

##### 3.6.8.1　查询5行记录，起始行从0开始

**SELECT \* FROM emp LIMIT 0, 5;**

注意，起始行从0+1开始，即第一行开始！

##### 3.8.8.2　查询10行记录，起始行从3开始

**SELECT \* FROM emp LIMIT 3, 10;**

##### 3.8.8.3　分页查询

如果一页记录为10条，希望查看第3页记录应该怎么查呢？

* 第一页记录起始行为0，一共查询10行；
* 第二页记录起始行为10，一共查询10行；
* 第三页记录起始行为20，一共查询10行；

## 4 完整性约束

　　完整性约束是为了表的数据的正确性！如果数据不正确，那么一开始就不能添加到表中。

### 4.1　主键

当某一列添加了主键约束后，那么这一列的数据就不能重复出现。这样每行记录中其主键列的值就是这一行的唯一标识。例如学生的学号可以用来做唯一标识，而学生的姓名是不能做唯一标识的，因为学生有可能同名。

主键约束:非空,唯一,被引用(外键)

主键列的值不能为NULL，也不能重复！

指定主键约束使用PRIMARY KEY关键字

* 创建表：定义列时指定主键：

**CREATE TABLE stu(**

**sid CHAR(6) PRIMARY KEY,**

**sname VARCHAR(20),**

**age INT,**

**gender VARCHAR(10)**

**);**

* 创建表：定义列之后独立指定主键：

**CREATE TABLE stu(**

**sid CHAR(6),**

**sname VARCHAR(20),**

**age INT,**

**gender VARCHAR(10),**

**PRIMARY KEY(sid)**

**);**

* 修改表时指定主键：

**ALTER TABLE stu**

**ADD PRIMARY KEY(sid);**

* 删除主键（只是删除主键约束，而不会删除主键列）：

**ALTER TABLE stu DROP PRIMARY KEY;**

### 4.2　主键自增长

MySQL提供了主键自动增长的功能！这样用户就不用再为是否有主键是否重复而烦恼了。当主键设置为自动增长后，在没有给出主键值时，主键的值会自动生成，而且是最大主键值+1，也就不会出现重复主键的可能了。

* 创建表时设置主键自增长（主键必须是整型才可以自增长）：

**CREATE TABLE stu(**

**sid INT PRIMARY KEY AUTO\_INCREMENT,**

**sname VARCHAR(20),**

**age INT,**

**gender VARCHAR(10)**

**);**

* 修改表时设置主键自增长：

**ALTER TABLE stu CHANGE sid sid INT AUTO\_INCREMENT;**

* 修改表时删除主键自增长：

**ALTER TABLE stu CHANGE sid sid INT;**

### 4.3　非空

指定非空约束的列不能没有值，也就是说在插入记录时，对添加了非空约束的列一定要给值；在修改记录时，不能把非空列的值设置为NULL。

* 指定非空约束：

**CREATE TABLE stu(**

**sid INT PRIMARY KEY AUTO\_INCREMENT,**

**sname VARCHAR(10) NOT NULL,**

**age INT,**

**gender VARCHAR(10)**

**);**

当为sname字段指定为非空后，在向stu表中插入记录时，必须给sname字段指定值，否则会报错：

**INSERT INTO stu(sid) VALUES(1);**

　　插入的记录中sname没有指定值，所以会报错！

### 4.4　唯一

还可以为字段指定唯一约束！当为字段指定唯一约束后，那么字段的值必须是唯一的。这一点与主键相似！例如给stu表的sname字段指定唯一约束：

**CREATE TABLE tab\_ab(**

**sid INT PRIMARY KEY AUTO\_INCREMENT,**

**sname VARCHAR(10) UNIQUE**

**);**

**INSERT INTO sname(sid, sname) VALUES(1001, 'zs');**

**INSERT INTO sname(sid, sname) VALUES(1002, 'zs');**

　　当两次插入相同的名字时，MySQL会报错！

### 4.5 概念模型

当我们要完成一个软件系统时，需要把系统中的实体抽取出来，形成概念模型。

例如部门、员工都是系统中的实体。概念模型中的实体最终会成为Java中的类、数据库中表。

实体之间还存在着关系，关系有三种：

\* 1对多：例如每个员工都从属一个部门，而一个部门可以有多个员工，其中员工是多方，而部门是一方。

\* 1对1：例如老公和老婆就是一对一的关系，一个老公只能有一个老婆，而一个老婆只能有一个老公。

\* 多对多：老师与学生的关系就是多对多，一个老师可以有多个学生，一个学生可以有多个老师。

概念模型在Java中成为实体类（javaBean）

类就使用成员变量来完成关系，一般都是双向关联！

多对一双向中关联，即员工关联部门，部门也关联员工

class Employee {//多方关联一方

...

private Department department;

}

class Department {//一方关联多方

...

private List<Employee> employees;

}

class Husband {

...

private Wife wife;

}

class Wife {

...

private Husband

}

class Student {

...

private List<Teacher> teachers

}

class Teacher {

...

private List<Student> students;

}

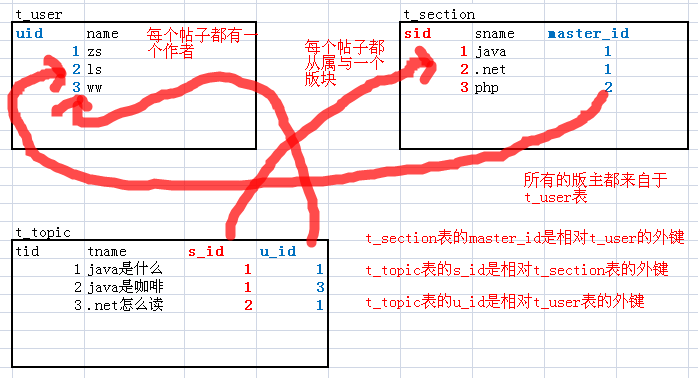
### 4.6　外键约束

主外键是构成表与表关联的唯一途径！

外键是另一张表的主键！例如员工表与部门表之间就存在关联关系，其中员工表中的部门编号字段就是外键，是相对部门表的外键。

外键可以重复,可以为空;

我们再来看BBS系统中：用户表（t\_user）、分类表（t\_section）、帖子表（t\_topic）三者之间的关系。



例如在t\_section表中sid为1的记录说明有一个分类叫java，版主是t\_user表中uid为1的用户，即zs！

例如在t\_topic表中tid为2的记录是名字为“Java是咖啡”的帖子，它是java版块的帖子，它的作者是ww。

外键就是用来约束这一列的值必须是另一张表的主键值！！！

* 创建t\_user表，指定uid为主键列：

**CREATE TABLE t\_user(**

**uid INT PRIMARY KEY AUTO\_INCREMENT,**

**uname VARCHAR(20) UNIQUE NOT NULL**

**);**

* 创建t\_section表，指定sid为主键列，u\_id为相对t\_user表的uid列的外键：

**CREATE TABLE t\_section(**

**sid INT PRIMARY KEY AUTO\_INCREMENT,**

**sname VARCHAR(30),**

**u\_id INT,**

**CONSTRAINT fk\_t\_user FOREIGN KEY(u\_id)** **REFERENCES t\_user(uid)**

**);**

* 修改t\_section表，指定u\_id为相对t\_user表的uid列的外键：

**ALTER TABLE t\_section**

**ADD CONSTRAINT fk\_t\_user**

**FOREIGN KEY(u\_id)**

**REFERENCES t\_user(uid);**

* 修改t\_section表，删除u\_id的外键约束：

**ALTER TABLE t\_section**

**DROP FOREIGN KEY fk\_t\_user;**

#### 概念模型在数据库中成为表

数据库表中的多对一关系，只需要在多方使用一个独立的列来引用1方的主键即可

/\*员工表\*/

create talbe emp (

empno int primary key,/\*员工编号\*/

...

deptno int/\*所属部门的编号\*/

);

/\*部门表\*/

create table dept (

deptno int primary key,/\*部门编号\*/

...

);

emp表中的deptno列的值表示当前员工所从属的部门编号。也就是说emp.deptno必须在dept表中是真实存在！

但是我们必须要去对它进行约束，不然可能会出现员工所属的部门编号是不存在的。这种约束就是外键约束。

我们需要给emp.deptno添加外键约束，约束它的值必须在dept.deptno中存在。外键必须是另一个表的主键！

语法：CONSTRAINT 约束名称 FOREIGN KEY(外键列名) REFERENCES 关联表(关联表的主键)

创建表时指定外键约束

create talbe emp (

empno int primary key,

...

deptno int,

CONSTRAINT fk\_emp FOREIGN KEY(mgr) REFERENCES emp(empno)

);

### 4.6　表与表之间的关系

* 一对一：例如t\_person表和t\_card表，即人和身份证。这种情况需要找出主从关系，即谁是主表，谁是从表。人可以没有身份证，但身份证必须要有人才行，所以人是主表，而身份证是从表。设计从表可以有两种方案：
* 在t\_card表中添加外键列（相对t\_user表），并且给外键添加唯一约束；
* 给t\_card表的主键添加外键约束（相对t\_user表），即t\_card表的主键也是外键。

**数据库一对一关系**

在表中建立一对一关系比较特殊，需要让其中一张表的主键，即是主键又是外键。

create table husband(

hid int PRIMARY KEY,

...

);

create table wife(

wid int PRIMARY KEY,

...

ADD CONSTRAINT fk\_wife\_wid FOREIGN KEY(wid) REFERENCES husband(hid)

);

其中wife表的wid即是主键，又是相对husband表的外键！

husband.hid是主键，不能重复！

wife.wid是主键，不能重复，又是外键，必须来自husband.hid。

所以如果在wife表中有一条记录的wid为1，那么wife表中的其他记录的wid就不能再是1了，因为它是主键。

同时在husband.hid中必须存在1这个值，因为wid是外键。这就完成了一对一关系。

* 一对多（多对一）：最为常见的就是一对多！一对多和多对一，这是从哪个角度去看得出来的。t\_user和t\_section的关系，从t\_user来看就是一对多，而从t\_section的角度来看就是多对一！这种情况都是在多方创建外键！
* 多对多：例如t\_stu和t\_teacher表，即一个学生可以有多个老师，而一个老师也可以有多个学生。这种情况通常需要创建中间表来处理多对多关系。例如再创建一张表t\_stu\_tea表，给出两个外键，一个相对t\_stu表的外键，另一个相对t\_teacher表的外键。

**数据库多对多关系**

在表中建立多对多关系需要使用中间表，即需要三张表，在中间表中使用两个外键，分别引用其他两个表的主键。

create table student(

sid int PRIMARY KEY,

...

);

create table teacher(

tid int PRIMARY KEY,

...

);

create table stu\_tea(

sid int,

tid int,

ADD CONSTRAINT fk\_stu\_tea\_sid FOREIGN KEY(sid) REFERENCES student(sid),

ADD CONSTRAINT fk\_stu\_tea\_tid FOREIGN KEY(tid) REFERENCES teacher(tid)

);

这时在stu\_tea这个中间表中的每条记录都是来说明student和teacher表的关系

例如在stu\_tea表中的记录：sid为1001，tid为2001，这说明编号为1001的学生有一个编号为2001的老师

sid tid

101 201 /\*编号为101的学生有一个编号为201的老师\*/

101 202 /\*编号为101的学生有一个编号为202的老师\*/

101 203 /\*编号为101的学生有一个编号为203的老师\*/

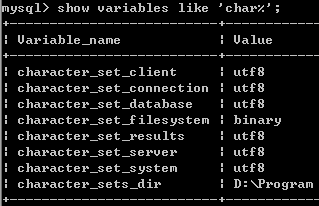
102 201 /\*编号为102的学生有一个编号为201的老师\*/

102 204 /\*编号为102的学生有一个编号为204的老师\*/

## 5 编码

### 5.1　查看MySQL编码

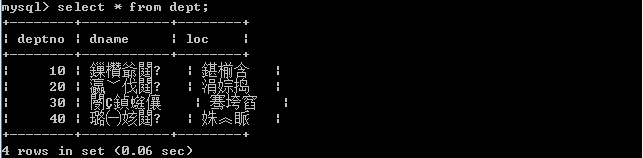
**SHOW VARIABLES LIKE 'char%';**



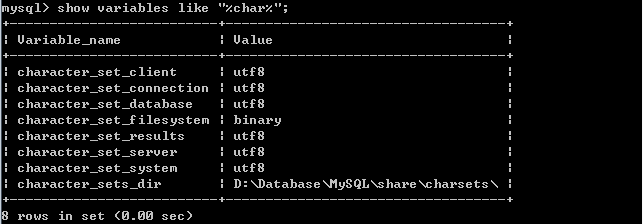
因为当初安装时指定了字符集为UTF8，所以所有的编码都是UTF8。

* character\_set\_client：你发送的数据必须与client指定的编码一致！！！服务器会使用该编码来解读客户端发送过来的数据；
* character\_set\_connection：通过该编码与client一致！该编码不会导致乱码！当执行的是查询语句时，客户端发送过来的数据会先转换成connection指定的编码。但只要客户端发送过来的数据与client指定的编码一致，那么转换就不会出现问题；
* character\_set\_database：数据库默认编码，在创建数据库时，如果没有指定编码，那么默认使用database编码；
* character\_set\_server：MySQL服务器默认编码；
* character\_set\_results：响应的编码，即查询结果返回给客户端的编码。这说明客户端必须使用result指定的编码来解码；

DOS命令行查询中文乱码问题解决方案:



数据库默认编码为utf8,客户端解析方式为gbk,造成乱码







### 5.2　控制台编码

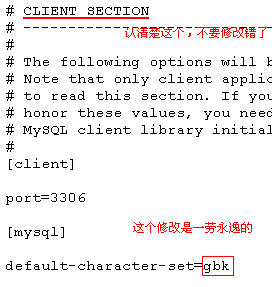
修改character\_set\_client、character\_set\_results、character\_set\_connection为GBK，就不会出现乱码了。但其实只需要修改character\_set\_client和character\_set\_results。

控制台的编码只能是GBK，而不能修改为UTF8，这就出现一个问题。客户端发送的数据是GBK，而character\_set\_client为UTF8，这就说明客户端数据到了服务器端后一定会出现乱码。既然不能修改控制台的编码，那么只能修改character\_set\_client为GBK了。

服务器发送给客户端的数据编码为character\_set\_result，它如果是UTF8，那么控制台使用GBK解码也一定会出现乱码。因为无法修改控制台编码，所以只能把character\_set\_result修改为GBK。

* 修改character\_set\_client变量：**set character\_set\_client=gbk;**
* 修改character\_set\_results变量：**set character\_set\_results=gbk;**

设置编码只对当前连接有效，这说明每次登录MySQL提示符后都要去修改这两个编码，但可以通过修改配置文件来处理这一问题：配置文件路径：D:\Program Files\MySQL\MySQL Server 5.1\ my.ini



**指定默认编码**

我们在安装MySQL时已经指定了默认编码为UTF8，所以我们在创建数据库、创建表时，都无需再次指定编码。

为了一劳永逸，可以在my.ini中设置：

设置character-set-server=utf8即可。

### 5.3　MySQL工具

使用MySQL工具是不会出现乱码的，因为它们会每次连接时都修改character\_set\_client、character\_set\_results、character\_set\_connection的编码。这样对my.ini上的配置覆盖了，也就不会出现乱码了。

## 6 MySQL数据库备份与还原

### 6.1　生成SQL脚本

在控制台使用mysqldump命令可以用来生成指定数据库的脚本文本，但要注意，脚本文本中只包含数据库的内容，而不会存在创建数据库的语句！所以在恢复数据时，还需要自已手动创建一个数据库之后再去恢复数据。注意不能有分=分号”;”.

|  |
| --- |
| mysqldump –u用户名 –p密码 数据库名>生成的脚本文件路径 |



现在可以在C盘下找到mydb1.sql文件了！

注意，mysqldump命令是在Windows控制台下执行，无需登录mysql！！！

### 6.2　执行SQL脚本

执行SQL脚本需要登录mysql，然后进入指定数据库，才可以执行SQL脚本！！！

执行SQL脚本不只是用来恢复数据库，也可以在平时编写SQL脚本，然后使用执行SQL 脚本来操作数据库！大家都知道，在黑屏下编写SQL语句时，就算发现了错误，可能也不能修改了。所以我建议大家使用脚本文件来编写SQL代码，然后执行之！

|  |
| --- |
| SOURCE C:\mydb1.sql |



　　注意，在执行脚本时需要先行核查当前数据库中的表是否与脚本文件中的语句有冲突！例如在脚本文件中存在create table a的语句，而当前数据库中已经存在了a表，那么就会出错！

还可以通过下面的方式来执行脚本文件：

mysql -uroot -p123 mydb1<c:\mydb1.sql

|  |
| --- |
| mysql –u用户名 –p密码 数据库<要执行脚本文件路径 |



这种方式无需登录mysql！

## 7 多表查询

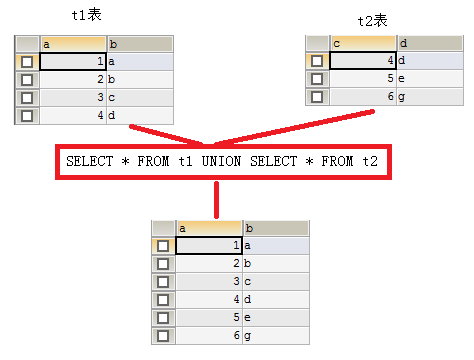
多表查询有如下几种：

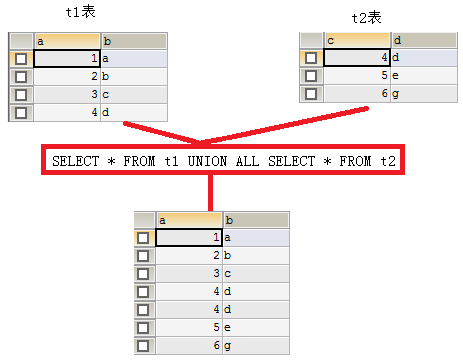
* 合并结果集；
* 连接查询
* 内连接
* 外连接
* 左外连接
* 右外连接
* 全外连接（MySQL不支持）
* 自然连接
* 子查询

### 7.1　合并结果集

1. 作用：合并结果集就是把两个select语句的查询结果合并到一起！
2. 合并结果集有两种方式：

* UNION：去除重复记录，例如：SELECT \* FROM t1 UNION SELECT \* FROM t2；
* UNION ALL：不去除重复记录，例如：SELECT \* FROM t1 UNION ALL SELECT \* FROM t2。

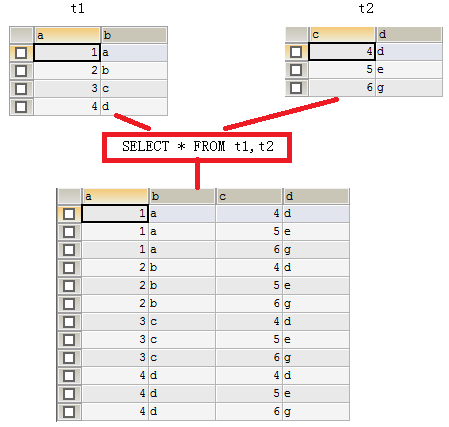




1. **要求：被合并的两个结果：列数、列类型必须相同。**

### 7.2　连接查询

连接查询就是求出多个表的乘积，例如t1连接t2，那么查询出的结果就是t1\*t2。

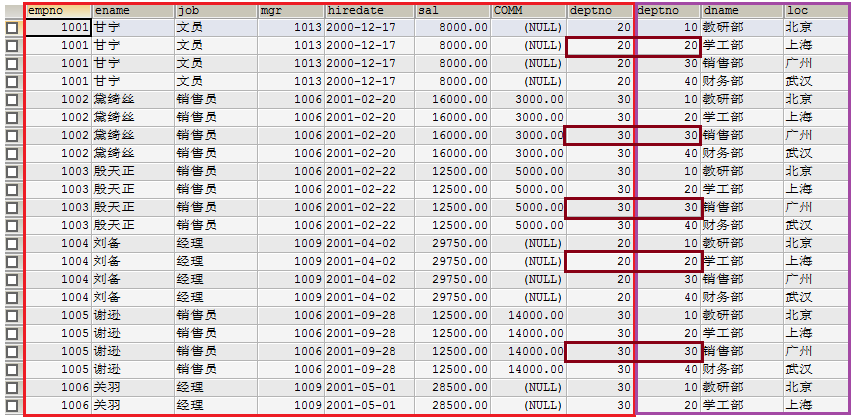


连接查询会产生笛卡尔积(所谓笛卡尔积,通俗点讲就是指包含在两个集合中任意取出两个元素构成的组合的集合.假设R集合中有元素a个,S集合中有元素b个,那么R和S构成的笛卡尔积的元素个数为a\*b)，假设集合A={a,b}，集合B={0,1,2}，则两个集合的笛卡尔积为{(a,0),(a,1),(a,2),(b,0),(b,1),(b,2)}。可以扩展到多个集合的情况。

那么多表查询产生这样的结果并不是我们想要的，那么怎么去除重复的，不想要的记录呢，当然是通过条件过滤。通常要查询的多个表之间都存在关联关系，那么就通过关联关系去除笛卡尔积。

你能想像到emp和dept表连接查询的结果么？emp一共14行记录，dept表一共4行记录，那么连接后查询出的结果是56行记录。

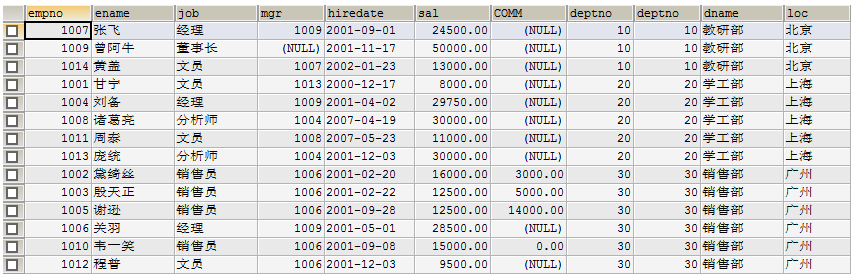
也就你只是想在查询emp表的同时，把每个员工的所在部门信息显示出来，那么就需要使用主外键来去除无用信息了。



#### **使用主外键关系做为条件来去除无用信息**

|  |
| --- |
| SELECT \* FROM emp,dept WHERE emp.deptno=dept.deptno; |

在多表查询中，在使用列时必须指定列所从属的表，例如emp.deptno表示emp表的deptno列。



上面查询结果会把两张表的所有列都查询出来，也许你不需要那么多列，这时就可以指定要查询的列了。

|  |
| --- |
| SELECT emp.ename,emp.sal,emp.comm,dept.dname  FROM emp,dept  WHERE emp.deptno=dept.deptno; |



还可以为表指定别名，然后在引用列时使用别名即可。

|  |
| --- |
| SELECT e.ename,e.sal,e.comm,d.dname  FROM emp AS e,dept AS d  WHERE e.deptno=d.deptno; |
| 其中AS是可以省略的 |

#### 7.2.1　内连接

上面的连接语句就是内连接，但它不是SQL标准中的查询方式，可以理解为方言！SQL标准的内连接为：

|  |
| --- |
| SELECT \*  FROM emp e  INNER JOIN dept d  ON e.deptno=d.deptno; |

INNER可以省略，MySQL默认的连接方式就是内连接

不使用WHERE，而是使用ON

内连接的特点：查询结果必须满足条件。例如我们向emp表中插入一条记录：



　　其中deptno为50，而在dept表中只有10、20、30、40部门，那么上面的查询结果中就不会出现“张三”这条记录，因为它不能满足e.deptno=d.deptno这个条件。

#### 7.2.2　外连接（左连接、右连接）

外连接的特点：查询出的结果存在不满足条件的可能。

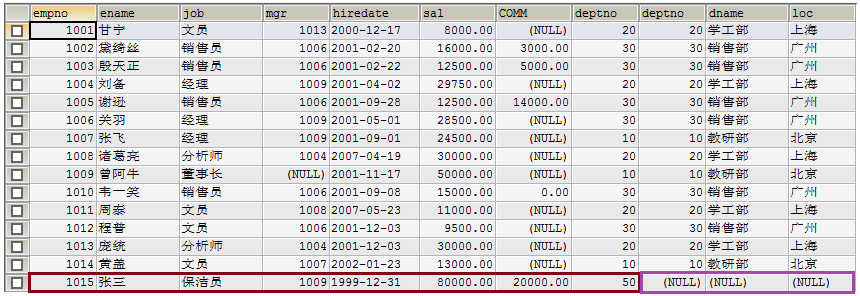
左连接：

|  |
| --- |
| SELECT \* FROM emp e  LEFT OUTER JOIN dept d  ON e.deptno=d.deptno; |

OUTER可以省略

左连接是先查询出左表（即以左表为主），然后查询右表，右表中满足条件的显示出来，不满足条件的显示NULL。

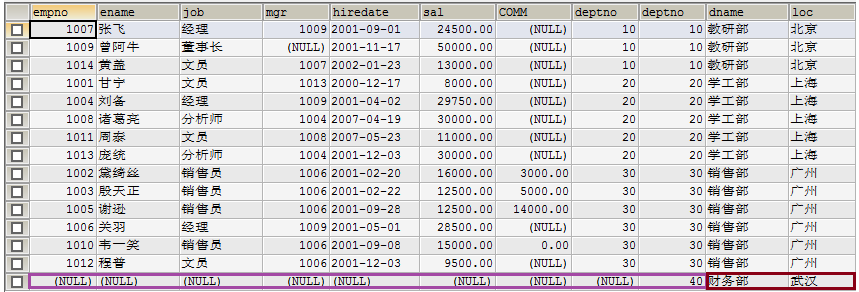
这么说你可能不太明白，我们还是用上面的例子来说明。其中emp表中“张三”这条记录中，部门编号为50，而dept表中不存在部门编号为50的记录，所以“张三”这条记录，不能满足e.deptno=d.deptno这条件。但在左连接中，因为emp表是左表，所以左表中的记录都会查询出来，即“张三”这条记录也会查出，但相应的右表部分显示NULL。



#### 7.2.3　右连接

右连接就是先把右表中所有记录都查询出来，然后左表满足条件的显示，不满足显示NULL。例如在dept表中的40部门并不存在员工，但在右连接中，如果dept表为右表，那么还是会查出40部门，但相应的员工信息为NULL。

|  |
| --- |
| SELECT \* FROM emp e  RIGHT OUTER JOIN dept d  ON e.deptno=d.deptno; |



**连接查询心得**：

连接不限与两张表，连接查询也可以是三张、四张，甚至N张表的连接查询。通常连接查询不可能需要整个笛卡尔积，而只是需要其中一部分，那么这时就需要使用条件来去除不需要的记录。这个条件大多数情况下都是使用主外键关系去除。

两张表的连接查询一定有一个主外键关系，三张表的连接查询就一定有两个主外键关系，所以在大家不是很熟悉连接查询时，首先要学会去除无用笛卡尔积，那么就是用主外键关系作为条件来处理。如果两张表的查询，那么至少有一个主外键条件，三张表连接至少有两个主外键条件。

### 7.3　自然连接

大家也都知道，连接查询会产生无用笛卡尔积，我们通常使用主外键关系等式来去除它。而自然连接无需你去给出主外键等式，它会自动找到这一等式：

* 两张连接的表中名称和类型完成一致的列作为条件，例如emp和dept表都存在deptno列，并且类型一致，所以会被自然连接找到！

当然自然连接还有其他的查找条件的方式，但其他方式都可能存在问题！

|  |
| --- |
| SELECT \* FROM emp NATURAL JOIN dept;内连接  SELECT \* FROM emp NATURAL LEFT JOIN dept;左连接  SELECT \* FROM emp NATURAL RIGHT JOIN dept;右连接 |

### 7.4　子查询

子查询就是嵌套查询，即SELECT中包含SELECT，如果一条语句中存在两个，或两个以上SELECT，那么就是子查询语句了。

* 子查询出现的位置：
* where后，作为条件的一部分；
* from后，作为被查询的一条表；
* 当子查询出现在where后作为条件时，还可以使用如下关键字：
* any
* all
* 子查询结果集的形式：
* 单行单列（用于条件）
* 单行多列（用于条件）
* 多行单列（用于条件）
* 多行多列（用于表）

练习：

1. 工资高于甘宁的员工。

分析：

查询条件：工资>甘宁工资，其中甘宁工资需要一条子查询。

第一步：查询甘宁的工资

|  |
| --- |
| SELECT sal FROM emp WHERE ename='甘宁' |

第二步：查询高于甘宁工资的员工

|  |
| --- |
| SELECT \* FROM emp WHERE sal > (${第一步}) |

结果：

|  |
| --- |
| SELECT \* FROM emp WHERE sal > (SELECT sal FROM emp WHERE ename='甘宁') |

* 子查询作为条件
* 子查询形式为单行单列

1. 工资高于30部门所有人的员工信息

分析：

查询条件：工资高于30部门所有人工资，其中30部门所有人工资是子查询。高于所有需要使用all关键字。

第一步：查询30部门所有人工资

|  |
| --- |
| SELECT sal FROM emp WHERE deptno=30; |

第二步：查询高于30部门所有人工资的员工信息

|  |
| --- |
| SELECT \* FROM emp WHERE sal > ALL (${第一步}) |

结果：all 大于所有

|  |
| --- |
| SELECT \* FROM emp WHERE sal > **ALL** (SELECT sal FROM emp WHERE deptno=30) |

* 子查询作为条件
* 子查询形式为多行单列（当子查询结果集形式为多行单列时可以使用ALL或ANY关键字）

1. 查询工作和工资与殷天正完全相同的员工信息

分析：

查询条件：工作和工资与殷天正完全相同，这是子查询

第一步：查询出殷天正的工作和工资

|  |
| --- |
| SELECT job,sal FROM emp WHERE ename='殷天正' |

第二步：查询出与殷天正工作和工资相同的人

|  |
| --- |
| SELECT \* FROM emp WHERE (job,sal) IN (${第一步}) |

结果：IN在这里我们要注意一个问题,那就是in前面的值不管是一个还是多个我们要将其看为一个整体

|  |
| --- |
| SELECT \* FROM emp WHERE (job,sal) IN (SELECT job,sal FROM emp WHERE ename='殷天正') |

* 子查询作为条件
* 子查询形式为单行多列

1. 查询员工编号为1006的员工名称、员工工资、部门名称、部门地址

分析：

查询列：员工名称、员工工资、部门名称、部门地址

查询表：emp和dept，分析得出，不需要外连接（外连接的特性：某一行（或某些行）记录上会出现一半有值，一半为NULL值）

条件：员工编号为1006

第一步：去除多表，只查一张表，这里去除部门表，只查员工表

|  |
| --- |
| SELECT ename, sal FROM emp e WHERE empno=1006 |

第二步：让第一步与dept做内连接查询，添加主外键条件去除无用笛卡尔积

|  |
| --- |
| SELECT e.ename, e.sal, d.dname, d.loc  FROM emp e, dept d  WHERE e.deptno=d.deptno AND empno=1006 |

第二步中的dept表表示所有行所有列的一张完整的表，这里可以把dept替换成所有行，但只有dname和loc列的表，这需要子查询。

第三步：查询dept表中dname和loc两列，因为deptno会被作为条件，用来去除无用笛卡尔积，所以需要查询它。

|  |
| --- |
| SELECT dname,loc,deptno FROM dept; |

第四步：替换第二步中的dept<底层代码其实是上面的形式>

|  |
| --- |
| SELECT e.ename, e.sal, d.dname, d.loc  FROM emp e, (SELECT dname,loc,deptno FROM dept) d  WHERE e.deptno=d.deptno AND e.empno=1006  SELECT e.ename, e.sal, d.dname, d.loc  FROM emp e, dept d  WHERE e.deptno=d.deptno AND e.empno=1006 |

* 子查询作为表
* 子查询形式为多行多列