MySQL数据库基础

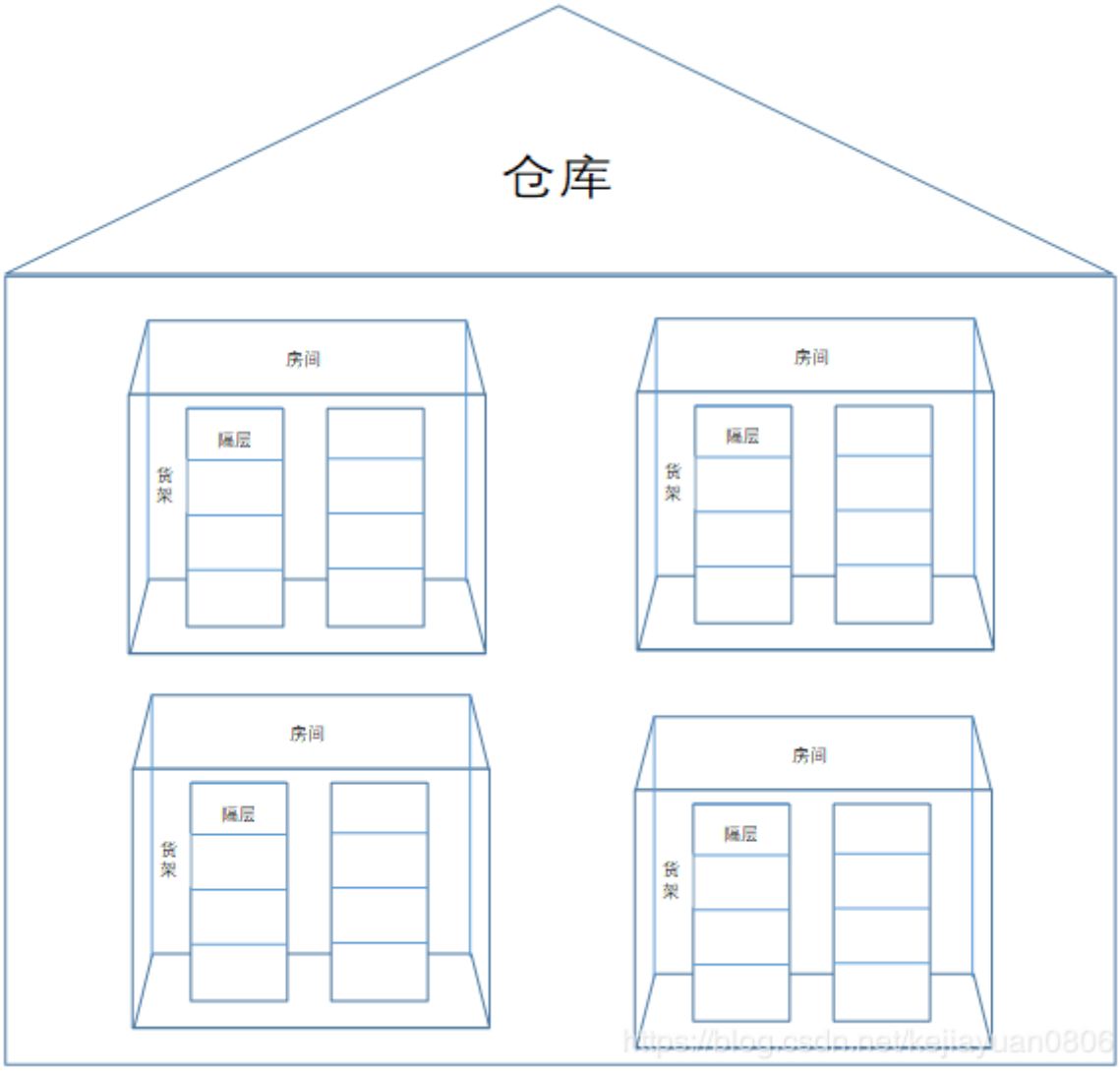
“数据库”起源于20世纪60年代后期，当时美国为了战争的需要，把各种情报收集在一起，存储在计算机内，叫做DataBase(DB)。随着计算机广泛地应用于数据管理，对数据的共享提出了越来越高的要求，传统的文件系统已经不能满足人们的需要，能够统一管理和共享数据的数据库管理系统（DBMS）应运而生。

数据库概述

一、数据库简介

1.什么是数据库

我们可以把它理解为商场里存放货物的仓库，仓库中的货物不是杂乱无章的堆放，而是分门别类来陈列的。一个仓库里有多个存储货物的房间，每个房间有多个货架，每个货架有多个隔层用来存放不同的物品。数据库也是一样，它是**按照一定的数据结构来组织、存储和管理数据的仓库。**



我们使用数据库可以存储千万级以上的数据量，相对于文件管理来说，数据存储量和运行效率大大提升。同时也可以设置约束条件来保证数据的一致性和完整性，并且可以实现多个用户同时访问、共享数据，通过设置不同用户的访问权限可以保证数据的安全性，所以在企业数据管理中都会使用数据库来组织和存储数据。

2.数据库分类

数据库按照不同的数据结构可以分为关系型数据库和非关系型数据库。

2.1关系型数据库

关系型数据库中存放的是结构化数据集，它是由固定列和任意行构成的二维表结构的数据。

主流的关系型数据库：

- Oracle：运行稳定、功能齐全、性能超群，适用于大型企业。

- DB2：速度快、可靠性好、处理海量数据急速高效，适用于大中型企业。
- MySQL：开源、体积小、速度快，适用于中小型企业。
- SQL server：全面高效、界面友好易操作，适用于中小型企业。

2.2非关系型数据库

非关系型数据库存放的是非结构化数据集，例如用户的聊天记录、拍摄的图片、录制的视频等等。

常用的非关系型数据库：NOSQL、mongoDB等。

3.表结构的特点

现在业界用的最多的就是关系型数据库，所以我们这里重点介绍关系型数据库的存储结构，即表结构。表结构中的表是来源于关系型数据库中的table，关系型数据库中的数据都是以表结构进行存储的，它是数据分析工具中最基本的数据存储结构。

字段

记录

员工号	员工姓名	职位	直属领导	雇佣时间	薪水	提成	部门号
7369	smitz	clerk	7902	1980-12-17	800		20
7499	allen	salesman	7698	1981-02-20	1600	300	30
7521	ward	salesmaz	7698	1981-02-22	1250	500	30
7566	jones	manager	7839	1981-04-02	2975		20
7654	martin	salesman	7698	1981-09-28	1250	1400	30
7698	blake	manager	7839	1981-05-01	2850		30
7782	clark	manager	7839	1981-06-09	2450		10
7788	scott	analyst	7566	1987-04-19	3000		20
7839	king	persident		1981-11-17	5000		10
7844	turner	salesman	7698	1981-09-08	1500	0	30
7876	adams	clerk	7788	1987-05-23	1100		20
7900	james	clerk	7698	1981-12-03	950		30
7902	ford	analyst	7566	1981-12-03	3000		20
7934	miller	clerk	7782	1982-01-23	1300		10

- 由固定列和任意行构成的表结构的数据集
- 表中的列称为字段，表中的行称为记录
- 以**字段**为基本的存储单位和计算单位
- 每一个字段必须有字段名，且**同一个表中的字段名不能重复**
- 每个字段的**数据类型必须一致**

二、MySQL简介

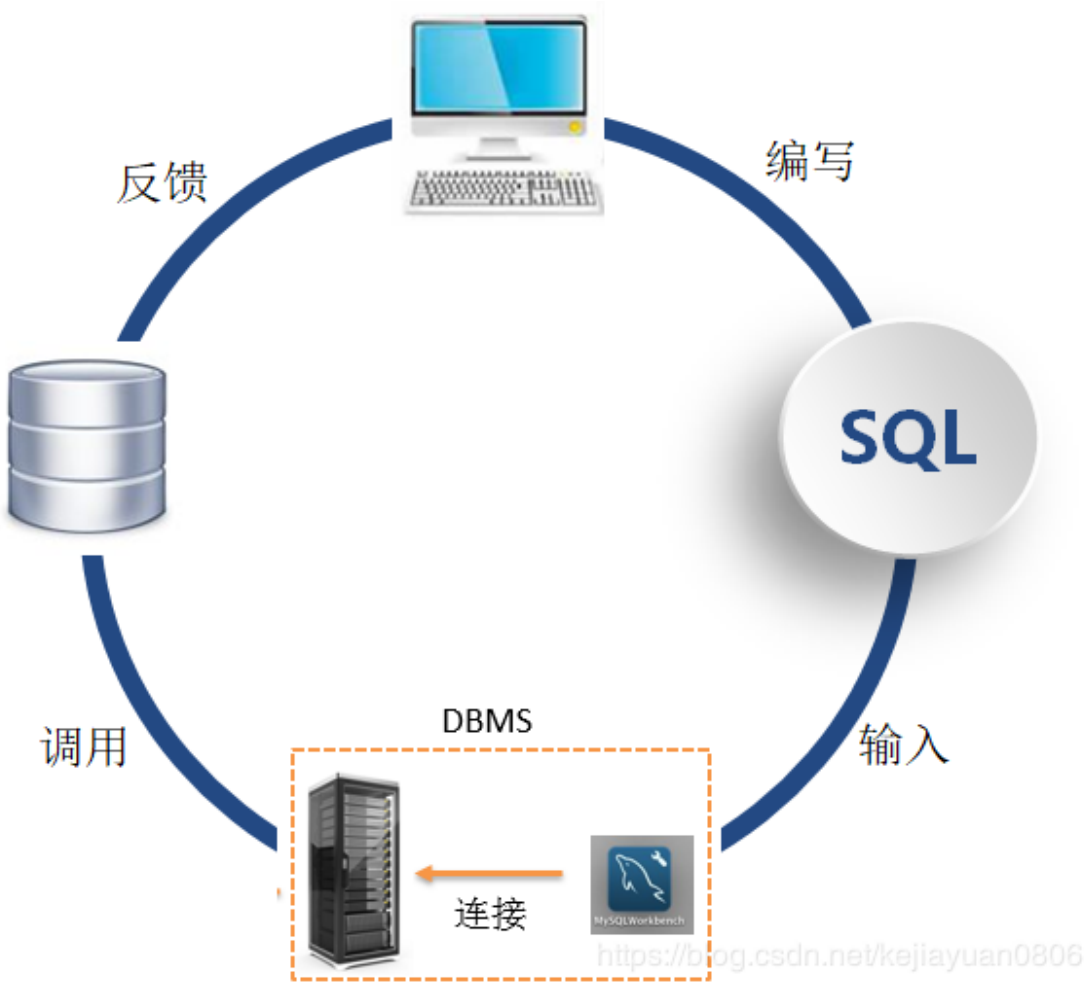
我们在工作中企业喜欢用MySQL，因为它成本低、速度快，并且它所提供的功能也可以满足大部分业务需求，所以基于这些特性MySQL成为企业中比较受欢迎的数据库管理系统，我们本次课程也选择使用MySQL来进行学习。

1.MySQL发展历程

最早是由瑞典MySQL AB公司开发，仅供内部使用。2000年基于GPL协议开放源码，2008年MySQL AB公司被Sun公司收购，2009年Sun公司又被Oracle公司收购。有了Oracle公司的技术支持，MySQL在2010年以后发布了多个版本，在各方面加强了企业级的特性。

2.数据库、数据库管理系统和SQL之间的关系

- 数据库：长期存储在计算机内、有组织的、统一管理的**相关数据的集合。**
- 数据库管理系统：**用于管理数据库的软件**，它对数据库进行统一的管理和控制，以保证数据库的安全性和完整性。
- SQL：一种**结构化查询语言**（Structure Query Language），它是国际标准化组织（ISO）采纳的标准数据库语言。



三、SQL语言

结构化查询语言(Structured Query Language)简称SQL，是一种数据库查询和程序设计语言，用于存取数据以及查询、更新和管理关系数据库系统。

1.SQL语言分类

SQL语言按照不同的功能可以分为以下四类：

- 数据定义语言DDL：用于创建，修改，删除数据库中的各种对象（数据库、表、视图、索引等），常用命令有CREATE，ALTER，DROP
- 数据操作语言DML：用于操作数据库表中的记录，常用命令有INSERT，UPDATE，DELETE
- 数据查询语言DQL：用于查询数据库表中的记录，基本结构：SELECT <字段名> FROM <表或视图名> WHERE <查询条件>
- 数据控制语言DCL：用于定义数据库访问权限和安全级别，常用命令：GRANT，REVOKE

作为一名数据分析师，我们需要重点掌握DQL数据查询语言，熟练使用DDL和DML，而DCL数据控制语言简单了解即可。

2.SQL书写要求

- SQL语句可以单行或多行书写，用分号结尾
- SQL关键字用空格分隔，也可以用缩进来增强语句的可读性
- SQL对大小写不敏感
- 可用#或-- 单行注释，用/* */多行注释，注释语句不可执行

数据定义

数据定义语言是对数据库管理系统中的对象进行增删改查的操作。

一、定义数据库

数据库是专门存储数据对象的容器，这里的数据对象包括表、视图、触发器、存储过程等，其中表是最基本的数据对象。

1.创建数据库

在 MySQL 数据库中存储数据对象之前，先要创建好数据库。

```
create database <数据库名称>;
```

- 数据库名称不能与SQL关键字相同
- 同一个数据库管理系统中的数据库名称不能重复

示例：创建一个名为test的数据库

```
create database test;
```

2.查看数据库

在 MySQL 中，可查看当前用户权限范围以内的数据库。

```
show databases;
```

示例：查看当前用户权限内的所有数据库

```
show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

3.选择数据库

在 MySQL 中，use 语句用来完成一个数据库到另一个数据库的跳转。创建数据库之后，该数据库不会自动成为当前数据库，需要用 USE 来指定当前数据库，才能对该数据库及其存储的数据对象执行操作。

```
use <数据库名称>;
```

示例：将test指定为当前数据库

```
use test;
```

4.删除数据库

在 MySQL 中，删除数据库的同时会删除数据库中存储的所有对象和数据，因此需谨慎使用。

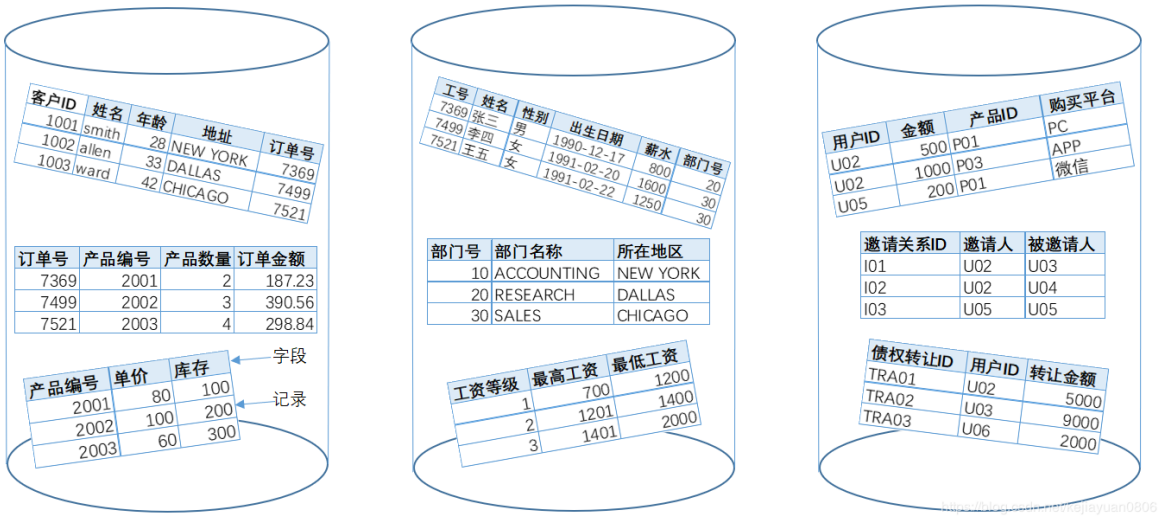
```
drop database <数据库名称>;
```

示例：删除test数据库

```
drop database test;
```

5.数据库基本结构

因为一个数据库管理系统可以管理多个数据库，一个数据库中可以有多张数据表，一个表中可以有多个字段，因为数据库是以字段为单位进行存储和计算的，所以同一个表中的字段名不可以重复，每个字段的数据类型必须一致，否则是不允许存储的。



那么我们在存储数据之前需要先创建表，创建表需要指定表名、字段名以及每个字段的数据类型和约束条件，其中表名、字段名和数据类型是必须指定的，约束条件可以不指定。

二、数据类型

数据库中每个字段都有适当的数据类型，用于限制或允许该字段中存储的数据。
MySQL中支持三种数据类型：数值型、字符串型、日期和时间型。
不同的数据类型提供不同的取值范围，可以存储的值范围越大，所需的存储空间也会越大。因此应根据实际需要选择最合适的类型，这样有利于提高查询的效率和节省存储空间。

1.数值型

数值型分为整数型和小数型。

类型	大小	范围（有符号）	范围（无符号）	用途
TINYINT	1 字节	(-128，127)	(0，255)	小整数值
SMALLINT	2 字节	(-32 768，32 767)	(0，65 535)	大整数值
MEDIUMINT	3 字节	(-8 388 608，8 388 607)	(0，16 777 215)	大整数值
INT或INTEGER	4 字节	(-2 147 483 648，2 147 483 647)	(0，4 294 967 295)	大整数值
BIGINT	8 字节	(-9,223,372,036,854,775,808，9 223 372 036 854 775 807)	(0，18 446 744 073 709 551 615)	极大整数值
FLOAT	4 字节	(-3.402 823 466 E+38，-1.175 494 351 E-38)，0，(1.175 494 351 E-38，3.402 823 466 351 E+38)	0，(1.175 494 351 E-38，3.402 823 466 E+38)	单精度浮点数值
DOUBLE	8 字节	(-1.797 693 134 862 315 7 E+308，-2.225 073 858 507 201 4 E-308)，0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	双精度浮点数值
DECIMAL	对DECIMAL(M,D)，如果M>D，为M+2否则为D+2	依赖于M和D的值	依赖于M和D的值	小数值

- int整数型，包括正数和负数，我们可以指定它的最大长度，如果不指定它默认最多存储11位数。
- float小数型，默认最多存储10位数，其中8位整数2位小数。
- decimal十进制小数型，适合金额、价格等对精度要求较高的数据存储。默认decimal(10,0)，表示最多10位数字，其中0位小数。

2.字符串型

字符串型主要用来存储不能进行数学运算的文本数据，还可以存储图片和声音的二进制数据。

类型	大小	用途
CHAR	0-255字节	定长字符串
VARCHAR	0-65535 字节	变长字符串
TINYBLOB	0-255字节	不超过 255 个字符的二进制字符串
TINYTEXT	0-255字节	短文本字符串
BLOB	0-65 535字节	二进制形式的长文本数据
TEXT	0-65 535字节	长文本数据
MEDIUMBLOB	0-16 777 215字节	二进制形式的中等长度文本数据
MEDIUMTEXT	0-16 777 215字节	中等长度文本数据
LONGBLOB	0-4 294 967 295字节	二进制形式的极大文本数据
LONGTEXT	0-4 294 967 295字节	极大文本数据

- char存储定长字符串，可以指定字符长度，如char(10)，不论存储数据是否达到10个字节，都会占用10个字节的空间（自动用空格填充），如'abc　　'，提取数据时需要用trim()去掉多余的空格。因为长度固定，方便程序的存储和查找，存取效率比较高。
- varchar存储可变长度字符串，可以指定字符长度，如varchar(10)，占用实际字符长度的存储空间，如'abc'。
- text长文本字符串型，最大长度65535，不能指定长度。

3.日期时间型

日期时间型用于存储日期和时间格式的数据。

类型名称	日期格式	日期范围	存储需求
YEAR	YYYY	1901 ~ 2155	1 个字节
TIME	HH:MM:SS	-838:59:59 ~ 838:59:59	3 个字节
DATE	YYYY-MM-DD	1000-01-01 ~ 9999-12-3	3 个字节
DATETIME	YYYY-MM-DD HH:MM:SS	1000-01-01 00:00:00 ~ 9999-12-31 23:59:59	8 个字节
TIMESTAMP	YYYY-MM-DD HH:MM:SS	1980-01-01 00:00:01 UTC ~ 2040-01-19 03:14:07 UTC	4 个字节

- date为'yyyy-MM-dd'格式的日期类型。
- time为'hh:mm:ss'格式的时间类型。
- datetime为'yyyy-MM-dd hh:mm:ss'格式的日期时间型。
- timestamp为时间戳，表示从1970-01-01 00:00:00到指定日期一共过了多少秒。

在数据库中**字符串和日期时间型**的数据都需要用引号括起来。

三、约束条件

约束条件是在表和字段上强制执行的数据检验规则，它是为了防止不规范的数据进入数据库，在我们对数据执行插入、修改、删除等一系列操作的时候，数据库管理系统会自动按照指定的约束条件对数据进行监测，它主要是对空值和重复值的检测，来保证数据存储的完整性和准确性。

按照约束的不同功能，MySQL中常用的约束条件有以下六种。

1.主键约束

主键是表中非空不重复的字段，它可以唯一的标识表中的一条记录。作为主键的字段取值不能为空，也不可以重复,并且一张表中只能有一个主键，但是构成主键的字段可以是一个也可以有多个。也就是说主键只能有一个，但是一个主键可以由多个字段构成，当多个字段的取值完全一样的情况下才会违反主键约束。

添加主键约束的方法1：

```
creat table <表名> (
<字段名1> <字段类型1> primary key,
<字段名2> <字段类型2>,
……
<字段名n> <字段类型n>
);
```

示例：

```
create table employee(
e_id int primary key,
e_name varchar(5),
e_sex varchar(5),
e_age int,
d_id int
);
```

添加主键约束的方法2:

```
create table <表名>(
<字段名1> <字段类型1>,
<字段名2> <字段类型2>,
......
<字段名n> <字段类型n>,
[constraint 主键约束名] primary key(字段名1[,字段名2,...字段名n])
);
```

示例:

```
create table employee(
e_name varchar(5),
e_sex varchar(5),
e_age int,
d_id int,
primary key(e_name,e_sex)
);
```

删除主键约束的方法:

```
alter table <表名> drop primary key;
```

示例:

```
alter table employee drop primary key;
```

2.唯一约束

唯一约束要求指定字段的数据取值不能重复，可以是空值，但是空值也只能出现一次。

添加唯一约束的方法1:

```
creat table <表名> (
<字段名1> <字段类型1> unique,
<字段名2> <字段类型2>,
……
<字段名n> <字段类型n>
);
```

示例:

```
create table employee(
e_id int,
e_name varchar(5) unique,
e_sex varchar(5),
e_age int,
d_id int
);
```

添加唯一约束的方法2:

```
creat table <表名> (
<字段名1> <字段类型1>,
<字段名2> <字段类型2>,
……
<字段名n> <字段类型n>,
[constraint 唯一约束名] unique (字段名1[,字段名2...字段名n])
);
```

示例:

```
create table employee(
e_id int,
e_name varchar(5),
e_sex varchar(5),
e_age int,
d_id int,
unique (e_name,e_sex)
);
```

删除唯一约束的方法:

```
alter table <表名> drop index <唯一约束名>;
```

如果单个字段没有指定唯一约束名，则默认的唯一约束名为字段名。如果是多个字段组合为唯一约束时候，默认的唯一约束名为第一个字段的名称。如果指定了约束名，则删除的时候写约束名。

示例:

```
alter table employee drop index e_name;
```

3.自动增长约束

自动增长约束是要求指定字段的数据取值自动增长，默认是从1开始，每增加一条记录，这个字段的取值就会加1，所以它只适用数值型和日期时间型字段，并且要配合主键一起使用。

添加字段增长约束的方法:

```
creat table <表名> (
<字段名1> <字段类型1> primary key auto_increment,
<字段名2> <字段类型2>,
……
<字段名n> <字段类型n>
);
```

示例:

```
create table employee(
e_id int primary key auto_increment,
e_name varchar(5),
e_sex varchar(5),
e_age int,
d_id int
);
```

删除自动增长约束的方法:

```
alter table <表名> modify <字段名> <字段类型>;
```

示例:

```
alter table employee modify e_id int;
```

4.非空约束

非空约束是要求指定字段的取值不能为空值。

非空约束的方法:

```
creat table <表名> (
<字段名1> <字段类型1> not null,
<字段名2> <字段类型2>,
……
<字段名n> <字段类型n>
);
```

示例:

```
create table employee(
e_id int,
e_name varchar(5) not null,
e_sex varchar(5),
e_age int,
d_id int
);
```

删除非空约束的方法:

```
alter table <表名> modify <字段名> <字段类型> [null];
```

示例:

```
alter table employee modify e_name int;
```

5.默认约束

默认约束是指在插入新记录的时候，如果没有为指定字段赋值，数据库管理系统会自动为这个字段赋值为默认约束设定的值。

添加默认约束的方法:

```
creat table <表名> (
<字段名1> <字段类型1> default value,
<字段名2> <字段类型2>,
……
<字段名n> <字段类型n>
);
```

示例:

```
create table employee(
e_id int,
e_name varchar(5),
e_sex varchar(5),
e_age int default 0,
d_id int
);
```

删除默认约束的方法:

```
alter table <表名> modify <字段名> <字段类型>;
```

示例:

```
alter table employee modify e_age int;
```

6.外键约束

外键约束是指一个表中的字段取值依赖于另一个表中字段的值。主键所在的表叫主表，外键所在的表叫从表。每一个外键值必须与主表中的主键值相对应。

添加外键约束的方法:

```
creat table <表名> (
<字段名1> <字段类型1>,
<字段名2> <字段类型2>,
……
<字段名n> <字段类型n>,
[constraint 外键约束名] foreign key(字段名) references <主表>(主键字段)
);
```

示例:

```
-- 创建一个主表
create table department(
d_id int primary key,
d_name varchar(5),
d_num int);
-- 创建从表的同时添加外键
create table employee(
e_id int primary key,
e_name varchar(5),
e_sex varchar(5),
```

```
e_age int,
d_id int,
foreign key(d_id) references department(d_id)
);
```

删除外键约束的方法：

```
alter table <表名> drop foreign key <外键约束名>;
```

- 先删除从表再删除主表。
- 先删除外键约束，再删除表。

示例：

```
alter table employee drop foreign key fk_d_id;
```

四、定义数据表

数据库中的数据是存储在基本表中的，所以需要先创建数据表再添加数据。

1.创建表

创建表的同时指定表名、字段名以及每个字段的数据类型和约束条件。

```
create table <表名>(
<字段名1> <数据类型1>[ <约束条件1>,
<字段名2> <数据类型2> <约束条件2>,
...
<字段名n> <数据类型n> <约束条件n>]
);
```

- <表名>：表名不可与SQL关键字相同，同一个数据库中的表名不可重复。
- <字段名>：字段名不可与SQL关键字相同，同一个表中的字段名不可重复。
- <约束条件>：约束条件可以不指定。

  示例：

  ```
  create table dept(
      deptno int primary key,
      dname varchar(15),
      loc varchar(10)
  );

  create table employee(
      empid int primary key,
      ename varchar(15) unique,
      job varchar(10) not null,
      mgr int,
      hiredate date,
      sal float default 0,
      comm float,
      deptno int,
      foreign key(deptno) references dept(deptno)
  );
  ```

2.查看表

在当前数据库中可以查看创建好的数据表。

```
show tables [like '表名'];
```

示例：查看当前数据库内的所有表

```
show tables;
+----------------+
| Tables_in_test |
+----------------+
| employee       |
| emp7           |
+----------------+
```

创建好数据表之后，可以查看表结构。

```
describe <表名>; 或 desc <表名>;
```

示例：查看employee表的结构定义

```
desc dep;
+--------+------------+------+-----+---------+-------+
| Field  | Type       | Null | Key | Default | Extra |
+--------+------------+------+-----+---------+-------+
| e_id   | varchar(5) | YES  |     | NULL    |       |
| e_name | char(5)    | YES  |     | NULL    |       |
| e_age  | int(11)    | YES  |     | NULL    |       |
| d_id   | varchar(5) | YES  |     | NULL    |       |
+--------+------------+------+-----+---------+-------+
```

Null：表示该字段是否可以存储 NULL 值。
Key：表示该字段是否已编制索引。PRI 表示主键，UNI 表示 UNIQUE 唯一索引，MUL 表示某个给定值允许出现多次。
Default：表示该字段是否有默认值，如果有，值是什么。
Extra：表示可以获取的附加信息，如 AUTO_INCREMENT 等。

3.修改表

为实现数据库中表规范化设计，有时候需要对已经创建的表进行结构修改或调整。

3.1修改表名

```
alter table <原表名> rename <新表名>;
```

示例：将employee表名改为emp

```
alter table employee rename emp;
```

3.2修改字段名

```
alter table <表名> change <原字段名> <新字段名> <新数据类型>;
```

示例：修改字段名empid为empno

```
alter table emp change empid empno varchar(5);
```

3.3修改字段类型

```
alter table <表名> modify <字段名> <新数据类型>;
```

示例：修改comm的字段类型为int

```
alter table emp modify comm int;
```

3.4添加新字段

```
alter table <表名> add <新字段名> <数据类型> [约束条件] [first|after 参照字段名];
```

示例：在emp表中添加新字段city

```
alter table emp add city varchar(10);
```

3.5修改字段的排列位置

```
alter table <表名> modify <字段名> <数据类型> first|after 参照字段名;
```

示例：修改字段city的排列位置

```
alter table emp modify city varchar(10) after d_id;
```

3.6删除字段

```
alter table <表名> drop <字段名>;
```

示例：删除字段city

```
alter table emp drop city;
```

4.删除表

删除表指删除表结构的同时删除表中数据，因此需谨慎使用。

```
drop table [if exists] <表名> [,<表名1>,<表名2>,...<表名n>];
```

可删除多张表，if exists避免表不存在时报错。

示例：删除emp表

```
drop table emp;
```

数据操作

数据操作语言（DML）是对表中记录进行添加（INSERT）、更新（UPDATE）、删除（DELETE）等操作。

一.添加数据

向表中添加数据时，字段名与字段值的数据类型、个数、顺序必须一一对应。

1.手动添加

```
insert into <表名> [<字段1>[,<字段2>,...<字段n>]] values (<值1>[,<值2>,...<值n>]);
```

- 省略字段名，则默认依次插入所有字段。
- 批量添加多个表或多个值之间使用逗号分隔。

  示例1：指定字段名添加

  ```
  insert into dept(deptno,dname,loc)
  values (10,'accounting','new york'),(20,'research','dallas');
  ```

  示例2：不指定字段名添加

  ```
  insert into dept values (30,'sales','chicago'),(40,'operations','boston');
  ```

2.批量导入

本地文件路径中不能含有中文，路径中用'\\'或'/'。

```
load data infile '<本地文件路径>'into table <表名> fields terminated by '<分隔符>'
[ignore n lines];
```

示例：

```
load data infile "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/employee.csv"
into table emp
fields terminated by ','
ignore 1 lines;
```

员工号	员工姓名	职位	直属领导	雇佣时间	薪水	津贴	部门号
empno	ename	job	mgr	hiredate	sal	comm	deptno
7369	smitz	clerk	7902	1980-12-17	800	NULL	20
7499	allen	salesman	7698	1981-02-20	1600	300	30
7521	ward	salesmaz	7698	1981-02-22	1250	500	30
7566	jones	manager	7839	1981-04-02	2975	NULL	20
7654	martin	salesman	7698	1981-09-28	1250	1400	30
7698	blake	manager	7839	1981-05-01	2850	NULL	30
7782	clark	manager	7839	1981-06-09	2450	NULL	10
7788	scott	analyst	7566	1987-04-19	3000	NULL	20
7839	king	persident	NULL	1981-11-17	5000	NULL	10
7844	turner	salesman	7698	1981-09-08	1500	0	30
7876	adams	clerk	7788	1987-05-23	1100	NULL	20
7900	james	clerk	7698	1981-12-03	950	NULL	30
7902	ford	analyst	7566	1981-12-03	3000	NULL	20
7934	miller	clerk	7782	1982-01-23	1300	NULL	10

3.将查询结果添加到已存在的表中

需事先创建表结构，且与select子句的字段类型一一对应。

```
insert into <新表名> [<字段1>[,<字段2>,...<字段n>]]
select <字段1>[,<字段2>,...<字段n>] from <原表名> [where <查询条件>];
```

示例:

```
-- 创建表
create table employee(
empno int primary key auto_increment,
ename varchar(5) not null,
ename varchar(5),
mgr int,
hiredate date,
sal float,
comm float,
deptno int);
-- 添加数据
insert into employee (empno,ename,deptno)
select e_id,e_name,d_id from emp;
```

4.将查询结果添加到新表中

添加数据的同时创建新表

```
create table <新表名> as
select <字段1>[,<字段2>,...<字段n>] from <原表名> [where <查询条件>];
```

示例:

```
create table employee as
select e_id,e_name,d_id from emp;
```

二.更新数据

在MySQL中，可以使用update语句来修改、更新表中的数据。

```
update <表名> set <字段1>=<值1> [,<字段2>=<值2>…<字段n>=<值n>] [where <更新条件>];
```

- set子句用于指定表中要修改的字段名及其字段值。每个指定的值可以是表达式，也可以是该字段对应的默认值。如果指定的是默认值，可用关键字default。
- where子句用于限定表中要修改的行。若不指定，则修改表中所有的行。
- 修改多个字段值时，set子句的每个值用逗号分开即可。

示例1：将emp表中7369员工的姓名修改为'abc'

```
update emp set ename='abc' where empno=7369;
```

三.删除数据

在MySQL中，可以使用delete语句来删除表中记录。

```
delete from <表名> [where 删除条件];
```

where子句表示为删除操作限定删除条件，若省略则代表删除表中的所有行。

示例1：删除20部门的员工记录

```
delete from emp where deptno=20;
```

示例2：删除所有的员工记录

```
delete from emp;
```

清除表中所有记录

```
truncate 表名;
```

示例：

```
truncate emp;
```

delete与truncate的区别：

- delete可以添加删除条件删除表中部分数据，truncate只能删除表中全部数据。
- delete删除表中数据保留表结构，truncate直接把表删除(drop)然后再创建(create)一张新表，执行速度比delete快。

数据查询

数据查询是使用数据查询语言(DQL)从一个或多个表中查询所需的数据，查询结果会通过客户端反馈给用户。

一.虚拟结果集

查询结果是存储在内存中的虚拟结果集，并不是真实存在的表。当用户执行其他命令时，内存中的虚拟结果集就会被释放，想要再次查看就需要再次执行查询命令，所以查询不会修改数据库表中的记录。

基本的查询语句就是SELECT语句，它至少包含两条信息：想查询什么，以及从哪里查询。

1.查询所有字段

在SELECT语句中使用星号“*”通配符查询所有字段。

```
select * from <表名1>[,<表名2>...,<表名n>];
```

示例:

```
select * from emp;
+-------+--------+------------+------+------------+------+------+--------+
| empno | ename  | job        | mgr  | hiredate   | sal  | comm | deptno |
+-------+--------+------------+------+------------+------+------+--------+
|  7369 | smith  | clerk      | 7902 | 1980-12-17 |  800 | NULL |     20 |
|  7499 | allen  | salesman   | 7698 | 1981-02-20 | 1600 |  300 |     30 |
|  7521 | ward   | salesman   | 7698 | 1981-02-22 | 1250 |  500 |     30 |
|  7566 | jones  | manager    | 7839 | 1981-04-02 | 2975 | NULL |     20 |
|  7654 | martin | salesman   | 7698 | 1981-09-28 | 1250 | 1400 |     30 |
|  7698 | blake  | manager    | 7839 | 1981-05-01 | 2850 | NULL |     30 |
|  7782 | clark  | manager    | 7839 | 1981-06-09 | 2450 | NULL |     10 |
|  7788 | scott  | analyst    | 7566 | 1987-04-19 | 3000 | NULL |     20 |
|  7839 | king   | persident  | NULL | 1981-11-17 | 5000 | NULL |     10 |
|  7844 | turner | salesman   | 7698 | 1981-09-08 | 1500 |    0 |     30 |
|  7876 | adams  | clerk      | 7788 | 1987-05-23 | 1100 | NULL |     20 |
|  7900 | james  | clerk      | 7698 | 1981-12-03 |  950 | NULL |     30 |
|  7902 | ford   | analyst    | 7566 | 1981-12-03 | 3000 | NULL |     20 |
|  7934 | miller | clerk      | 7782 | 1982-01-23 | 1300 | NULL |     10 |
+-------+--------+------------+------+------------+------+------+--------+
```

一般情况下，除非需要使用表中所有的字段，否则最好不要使用通配符“*”。使用通配符虽然可以节省输入查询语句的时间，但是获取不需要的字段通常会降低查询和所使用的应用程序的效率。通配符的优势是，当不知道所需的字段名时，可以通过通配符获取。

2.查询部分字段

需要查询部分字段时，用字段名替换通配符“*”。

```
select <字段名1>[,<字段名2>...,<字段名n>] from <表名1>[,<表名2>...,<表名n>];
```

如果不记得字段名，可以使用desc命令查看表结构。

示例:

```
select ename,job,sal from emp;
+--------+------------+------+
| ename  | job        | sal  |
+--------+------------+------+
| smith  | clerk      |  800 |
| allen  | salesman   | 1600 |
| ward   | salesman   | 1250 |
| jones  | manager    | 2975 |
| martin | salesman   | 1250 |
```

```
| blake  | manager    | 2850 |
| clark  | manager    | 2450 |
| scott  | analyst    | 3000 |
| king   | persident  | 5000 |
| turner | salesman   | 1500 |
| adams  | clerk      | 1100 |
| james  | clerk      |  950 |
| ford   | analyst    | 3000 |
| miller | clerk      | 1300 |
+--------+------------+------+
```

3.查询不重复的记录

在MySQL中需要查询表中不重复的记录时，可以使用distinct关键字过滤重复记录。

```
select distinct <字段名1>[,<字段名2>...,<字段名n>] from <表名>;
```

示例1：单个字段去重

```
select distinct deptno from emp;
+--------+
| deptno |
+--------+
|     20 |
|     30 |
|     10 |
+--------+
select distinct job from emp;
+------------+
| job        |
+------------+
| clerk      |
| salesman   |
| manager    |
| analyst    |
| persident  |
+------------+
```

示例2：多个字段去重

```
select distinct deptno,job from emp;
+--------+------------+
| deptno | job        |
+--------+------------+
|     20 | clerk      |
|     30 | salesman   |
|     20 | manager    |
|     30 | manager    |
|     10 | manager    |
|     20 | analyst    |
|     10 | persident  |
|     30 | clerk      |
|     10 | clerk      |
+--------+------------+
```

多个字段去重时，distinct关键字必须位于第一个字段前，多个字段完全一样的情况下，才会过滤。

4.设置别名

通常情况下我们通过计算字段得到的字段名都是比较长的，为了提高查询结果的可读性，我们可以使用as关键字设置别名，使查询语句更简洁。

要保证表别名不能与数据库中的其他表名或关键字重复。

MySQL支持两种别名：列别名和表别名。

```
select <字段名> [as] <字段别名> from <表名> [as] <表别名>;
```

- as关键字可省略

4.1列别名

有时字段名称是一些表达式，使查询结果的输出很难理解，可以使用列别名设置描述性名称。

示例1：查询所有员工总人数和平均工资

```
select count(*) as 员工数,avg(sal) as 平均工资 from emp;
+-----------+-------------------+
| 员工数    | 平均工资          |
+-----------+-------------------+
|        14 | 2073.214285714286 |
+-----------+-------------------+
```

示例2：别名包含空格时，需要加引号

```
select count(*) as 'count num',avg(sal) as 'avg sal' from emp;
+-----------+-------------------+
| count num | avg sal           |
+-----------+-------------------+
|        14 | 2073.214285714286 |
+-----------+-------------------+
```

示例3：查询员工数不低于5人的部门平均工资，按平均工资降序显示

```
select deptno 部门号,count(empno) 员工数,avg(sal) 平均工资
from emp
group by 部门号
having 员工数>=5
order by 平均工资 desc;
+-----------+-----------+--------------------+
| 部门号    | 员工数    | 平均工资           |
+-----------+-----------+--------------------+
|        20 |         5 |               2175 |
|        30 |         6 | 1566.6666666666667 |
+-----------+-----------+--------------------+
```

设置别名后可以在group by，having和order by子句中来引用该别名，而不能在WHERE子句中使用列别名。
因为SQL语句的执行顺序：

```
from -> where -> group by -> having -> select -> distinct -> order by -> limit
```

因此在where子句执行的时，设置别名的select子句还没有执行，即该别名还不存在，所以where子句不能使用，order by子句是在select后执行的，可以使用列别名。而group by和having子句虽然也是在select前执行，一般也不能在group by和having子句中使用别名，但是MySQL对此做了扩展。在MySQL5.7.5及之后的版本，ONLY_FULL_GROUP_BY sql mode默认开启，MySQL对标准SQL的扩展生效：

- 允许在select，having和order by子句中使用没有出现在group by中的字段。此时MySQL会随机选择没有出现在group by中的字段的值。
- 允许在group by和having子句中使用select中的别名。

4.2表别名

当表名重复引用时，可以使用别名为表添加不同的名称，用于简化和区分。

示例：查询所有领导姓名及其下属员工姓名

```
select e1.ename 上层领导,e2.ename 下属员工
from emp e1,emp e2
where e1.empno=e2.mgr;
+--------------+--------------+
| 上层领导     | 下属员工     |
+--------------+--------------+
| ford         | smith        |
| blake        | allen        |
| blake        | ward         |
| king         | jones        |
| blake        | martin       |
| king         | blake        |
| king         | clark        |
| jones        | scott        |
| blake        | turner       |
| scott        | adams        |
| blake        | james        |
| jones        | ford         |
| clark        | miller       |
+--------------+--------------+
```

表别名只在执行查询时使用，并不在返回结果中显示。而列别名在客户端的查询结果中显示，显示的结果集中字段名即为字段的别名。

二.条件查询

在 SELECT查询语句中，可以使用 WHERE 子句来指定查询条件，对表中的数据筛选，满足条件的记录会出现在结果集中。

```
select <字段名列表> from 表名 where <查询条件>;
```

<查询条件>是有运算符构成的逻辑判断表达式，结果取值为1或0。

1.常用运算符

- 算术运算符

运算符	作用
+	加
-	减
*	乘
/	除

- 比较运算符

运算符	作用
=	等于
>/>=	大于/大于等于
</<=	小于/小于等于
!=/<>	不等于
between...and...	值范围

- 逻辑运算符

运算符	作用
and	且
or	或
not	非

- 运算符优先级

优先级	运算符
1	()
2	算术运算符
3	比较运算符
4	逻辑运算符

示例1：查询职称为salesman的员工记录

```
select * from emp where job='salesman';
+-------+--------+------------+------+------------+------+------+--------+
| empno | ename  | job        | mgr  | hiredate   | sal  | comm | deptno |
+-------+--------+------------+------+------------+------+------+--------+
|  7499 | allen  | salesman   | 7698 | 1981-02-20 | 1600 |  300 |     30 |
|  7521 | ward   | salesman   | 7698 | 1981-02-22 | 1250 |  500 |     30 |
|  7654 | martin | salesman   | 7698 | 1981-09-28 | 1250 | 1400 |     30 |
|  7844 | turner | salesman   | 7698 | 1981-09-08 | 1500 |    0 |     30 |
+-------+--------+------------+------+------------+------+------+--------+
```

示例2：查询30号部门基本工资在1000到2000之间的员工信息

```
mysql> select * from emp where deptno=30 and sal between 1000 and 2000;
+-------+--------+------------+------+------------+------+------+--------+
| empno | ename  | job        | mgr  | hiredate   | sal  | comm | deptno |
+-------+--------+------------+------+------------+------+------+--------+
|  7499 | allen  | salesman   | 7698 | 1981-02-20 | 1600 |  300 |     30 |
|  7521 | ward   | salesman   | 7698 | 1981-02-22 | 1250 |  500 |     30 |
|  7654 | martin | salesman   | 7698 | 1981-09-28 | 1250 | 1400 |     30 |
|  7844 | turner | salesman   | 7698 | 1981-09-08 | 1500 |    0 |     30 |
+-------+--------+------------+------+------------+------+------+--------+
```

2.空值查询

在数据库中null为未知值，不属于任何数据类型，因此null和任何类型的数据进行运算的结果都为null，我们在进行运算时需要先处理null。

在WHERE子句查询条件中，可以使用IS关键字进行空值查询。

```
select <字段名列表> from <表名> where <字段名> is [not] null;
```

示例1：查询mgr为空的记录

```
select * from emp where mgr is null;
+-------+-------+-----------+------+------------+------+------+--------+
| empno | ename | job       | mgr  | hiredate   | sal  | comm | deptno |
+-------+-------+-----------+------+------------+------+------+--------+
|  7839 | king  | persident | NULL | 1981-11-17 | 5000 |    0 |     10 |
+-------+-------+-----------+------+------------+------+------+--------+
```

示例2：查询comm不为空的记录

```
select * from emp where comm is not null;
+-------+--------+------------+------+------------+------+------+--------+
| empno | ename  | job        | mgr  | hiredate   | sal  | comm | deptno |
+-------+--------+------------+------+------------+------+------+--------+
|  7369 | smith  | clerk      | 7902 | 1980-12-17 |  800 |    0 |     20 |
|  7499 | allen  | salesman   | 7698 | 1981-02-20 | 1600 |  300 |     30 |
|  7521 | ward   | salesman   | 7698 | 1981-02-22 | 1250 |  500 |     30 |
|  7566 | jones  | manager    | 7839 | 1981-04-02 | 2975 |    0 |     20 |
|  7654 | martin | salesman   | 7698 | 1981-09-28 | 1250 | 1400 |     30 |
|  7698 | blake  | manager    | 7839 | 1981-05-01 | 2850 |    0 |     30 |
|  7782 | clark  | manager    | 7839 | 1981-06-09 | 2450 |    0 |     10 |
|  7788 | scott  | analyst    | 7566 | 1987-04-19 | 3000 |    0 |     20 |
|  7839 | king   | persident  | NULL | 1981-11-17 | 5000 |    0 |     10 |
```

```
|  7844 | turner | salesman  | 7698 | 1981-09-08 | 1500 |    0 |     30 |
|  7876 | adams  | clerk     | 7788 | 1987-05-23 | 1100 |    0 |     20 |
|  7900 | james  | clerk     | 7698 | 1981-12-03 |  950 |    0 |     30 |
|  7902 | ford   | analyst   | 7566 | 1981-12-03 | 3000 |    0 |     20 |
|  7934 | miller | clerk     | 7782 | 1982-01-23 | 1300 |    0 |     10 |
+-------+--------+-----------+------+------------+------+------+--------+
```

3.模糊查询

在WHERE子句中，当我们不知道准确的查询条件时，可以使用LIKE关键字和通配符组成比较特定数据的搜索模式对文本字段进行模糊查询。

通配符是用来匹配值的一部分的特殊字符，同时使用LIKE关键字在WHERE子句中构成查询条件。通配符模糊查询只能用于文本字段（字符串），非文本数据类型字段不能使用通配符进行模糊查询。

```
select <字段名列表> from 表名 where 字段名 like <通配符>;
```

3.1百分号(%)

在模糊查询中，%表示任何字符出现任意次数。
示例1：查询姓名以a开头的员工信息

```
select * from emp where ename like 'a%';
+-------+-------+-----------+------+------------+------+------+--------+
| empno | ename | job       | mgr  | hiredate   | sal  | comm | deptno |
+-------+-------+-----------+------+------------+------+------+--------+
|  7499 | allen | salesman  | 7698 | 1981-02-20 | 1600 |  300 |     30 |
|  7876 | adams | clerk     | 7788 | 1987-05-23 | 1100 |    0 |     20 |
+-------+-------+-----------+------+------------+------+------+--------+
```

示例2：查询姓名中包含a的员工信息

```
select * from emp where ename like '%a%';
+-------+--------+-----------+------+------------+------+------+--------+
| empno | ename  | job       | mgr  | hiredate   | sal  | comm | deptno |
+-------+--------+-----------+------+------------+------+------+--------+
|  7499 | allen  | salesman  | 7698 | 1981-02-20 | 1600 |  300 |     30 |
|  7521 | ward   | salesman  | 7698 | 1981-02-22 | 1250 |  500 |     30 |
|  7654 | martin | salesman  | 7698 | 1981-09-28 | 1250 | 1400 |     30 |
|  7698 | blake  | manager   | 7839 | 1981-05-01 | 2850 |    0 |     30 |
|  7782 | clark  | manager   | 7839 | 1981-06-09 | 2450 |    0 |     10 |
|  7876 | adams  | clerk     | 7788 | 1987-05-23 | 1100 |    0 |     20 |
|  7900 | james  | clerk     | 7698 | 1981-12-03 |  950 |    0 |     30 |
+-------+--------+-----------+------+------------+------+------+--------+
```

3.2下划线（_）通配符

在模糊匹配中，_只匹配单个字符。
示例：查询姓名中第二个字符为a的员工信息

```
select * from emp where ename like '_a%';
+-------+--------+------------+------+------------+------+------+--------+
| empno | ename  | job        | mgr  | hiredate   | sal  | comm | deptno |
+-------+--------+------------+------+------------+------+------+--------+
|  7521 | ward   | salesman   | 7698 | 1981-02-22 | 1250 |  500 |     30 |
|  7654 | martin | salesman   | 7698 | 1981-09-28 | 1250 | 1400 |     30 |
|  7900 | james  | clerk      | 7698 | 1981-12-03 |  950 |    0 |     30 |
+-------+--------+------------+------+------------+------+------+--------+
```

3.3使用通配符的技巧

使用通配符进行模糊查询一般要比其他条件查询耗费更长的处理时间，所以我们要注意使用通配符的方法。

- 不要过度使用通配符。如果其他操作符能达到相同的目的，应该使用其他操作符。
- 在确实需要使用通配符时，也尽量不要把它们用在搜索模式的开始处，否则搜索起来是非常慢的。
- 注意通配符的位置。如果放错地方，可能不会返回想要的数据。

三.分组查询

在 SELECT 语句中，允许使用 GROUP BY 子句将查询结果按照一个或多个字段进行分组，字段值相同的为一组，对每个组进行聚合计算，实现数据的分组汇总。

```
select <字段名列表> from <表名> [where <查询条件>] group by <字段名列表>;
```

- select 后的字段名应该是分组字段和聚合字段。
- group by分组字段可以是一个或多个。
- 如果分组字段中包含有null值，则null值所在的行单独分为一组。

1.组内聚合

分组后需要对每个组内的数据进行聚合运算，可以使用sum、avg、count、max、min函数对数值型数据进行聚合计算，使用max和min函数对日期时间型数据进行聚合，使用group_concat函数对字符串型数据进行分组合并。

单字段分组：查询各部门的平均工资

```
select deptno,avg(sal) 平均工资
from emp
group by deptno;
+--------+--------------------+
| deptno | 平均工资           |
+--------+--------------------+
|     10 | 2916.6666666666665 |
|     20 |               2175 |
|     30 | 1566.6666666666667 |
+--------+--------------------+
```

多字段分组：查询各部门不同职位的平均工资

```
select deptno,job,avg(sal) 平均工资
from emp
group by deptno,job;
+--------+-----------+--------------+
| deptno | job       | 平均工资     |
+--------+-----------+--------------+
```

```
|     10 | clerk      |         1300 |
|     10 | manager    |         2450 |
|     10 | persident  |         5000 |
|     20 | analyst    |         3000 |
|     20 | clerk      |          950 |
|     20 | manager    |         2975 |
|     30 | clerk      |          950 |
|     30 | manager    |         2850 |
|     30 | salesman   |         1400 |
+--------+------------+--------------+
```

2.分组后筛选

在SELECT 语句中，使用GROUP BY子句进行分组后，如果需要对分组后的数据进行筛选，可以使用HANING子句指定筛选条件。

```
select <字段名列表> from <表名>
[where <查询条件>]
group by <分组字段列表>
having <筛选条件>;
```

HAVING 子句和 WHERE 子句非常相似，都是对数据进行过滤，HAVING 子句支持 WHERE 子句中所有的操作符和语法。

示例：查询各部门clerk的最低工资

```
select deptno,job,min(sal) 最低工资
from emp
where job='clerk'
group by deptno;
+--------+------------+--------------+
| deptno | job        | 最低工资     |
+--------+------------+--------------+
|     10 | clerk      |         1300 |
|     20 | clerk      |          800 |
|     30 | clerk      |          950 |
+--------+------------+--------------+

select deptno,job,min(sal) 最低工资
from emp
group by deptno,job
having job='clerk';
+--------+------------+--------------+
| deptno | job        | 最低工资     |
+--------+------------+--------------+
|     10 | clerk      |         1300 |
|     20 | clerk      |          800 |
|     30 | clerk      |          950 |
+--------+------------+--------------+
```

2.1WHERE与HAVING的区别：

- WHERE 子句主要用于过滤表，而 HAVING 子句主要用于过滤分组。
- WHERE 子句不可以使用聚合函数，HAVING 子句中可以包含聚合函数。
- HAVING 子句是在数据分组后进行过滤，WHERE 子句会在数据分组前进行过滤，WHERE 子句排除的行不包含在分组中。

示例：查询各部门平均工资大于3000的职位

```
select deptno,job,avg(sal)
from emp
where avg(sal)>3000
group by deptno,job;
ERROR 1111 (HY000): Invalid use of group function

select deptno,job,avg(sal)
from emp
group by deptno,job
having avg(sal)>3000;
+--------+-----------+----------+
| deptno | job       | avg(sal) |
+--------+-----------+----------+
|     10 | persident |     5000 |
+--------+-----------+----------+
```

2.2HAVING 子句中的筛选字段必须是可以出现在分组结果中的字段

示例：查询各部门manager的平均工资

```
select deptno,avg(sal) 平均工资
from emp
group by deptno
having job='manager';
ERROR 1054 (42S22): Unknown column 'job' in 'having clause'

select deptno,job,avg(sal) 平均工资
from emp
group by deptno,job
having job='manager';
+--------+------------+--------------+
| deptno | job        | 平均工资     |
+--------+------------+--------------+
|     10 | manager    |         2450 |
|     20 | manager    |         2975 |
|     30 | manager    |         2850 |
+--------+------------+--------------+
```

四.查询结果排序

SELECT语句查询出的数据如果不进行排序，数据一般将以它在底层表中出现的顺序显示，这可以是数据最初添加到表中的顺序。但是如果数据后来进行过更新或删除，则此顺序将会受到MySQL重用回收存储空间的影响。关系数据库设计理论认为，如果不明确规定排序顺序，则不应该假定查询出的数据的顺序有意义。
可以使用ORDER BY子句按一个或多个字段对SELECT语句查询出的数据进行排序。

```
select <字段名列表> from <表名> order by <字段名列表> [排序方向];
```

按多个列排序时，先按照第一个字段排序，仅在多个行第一个字段值相同时才按照第二个字段进行排序。如果第一个字段值都是唯一的，则不会按照第二个字段排序。

指定排序方向：asc升序，desc降序（没有指定排序方向时，默认是asc升序）

单字段排序：查询所有员工信息按sal降序显示

```
select *
from emp
order by sal desc;
+-------+--------+------------+------+------------+------+------+--------+
| empno | ename  | job        | mgr  | hiredate   | sal  | comm | deptno |
+-------+--------+------------+------+------------+------+------+--------+
|  7839 | king   | persident  | NULL | 1981-11-17 | 5000 |    0 |     10 |
|  7788 | scott  | analyst    | 7566 | 1987-04-19 | 3000 |    0 |     20 |
|  7902 | ford   | analyst    | 7566 | 1981-12-03 | 3000 |    0 |     20 |
|  7566 | jones  | manager    | 7839 | 1981-04-02 | 2975 |    0 |     20 |
|  7698 | blake  | manager    | 7839 | 1981-05-01 | 2850 |    0 |     30 |
|  7782 | clark  | manager    | 7839 | 1981-06-09 | 2450 |    0 |     10 |
|  7499 | allen  | salesman   | 7698 | 1981-02-20 | 1600 |  300 |     30 |
|  7844 | turner | salesman   | 7698 | 1981-09-08 | 1500 |    0 |     30 |
|  7934 | miller | clerk      | 7782 | 1982-01-23 | 1300 |    0 |     10 |
|  7521 | ward   | salesman   | 7698 | 1981-02-22 | 1250 |  500 |     30 |
|  7654 | martin | salesman   | 7698 | 1981-09-28 | 1250 | 1400 |     30 |
|  7876 | adams  | clerk      | 7788 | 1987-05-23 | 1100 |    0 |     20 |
|  7900 | james  | clerk      | 7698 | 1981-12-03 |  950 |    0 |     30 |
|  7369 | smith  | clerk      | 7902 | 1980-12-17 |  800 |    0 |     20 |
+-------+--------+------------+------+------------+------+------+--------+
```

多字段排序：查询所有员工信息按deptno升序、sal降序显示

```
select *
from emp
order by deptno,sal desc;
+-------+--------+------------+------+------------+------+------+--------+
| empno | ename  | job        | mgr  | hiredate   | sal  | comm | deptno |
+-------+--------+------------+------+------------+------+------+--------+
|  7839 | king   | persident  | NULL | 1981-11-17 | 5000 |    0 |     10 |
|  7782 | clark  | manager    | 7839 | 1981-06-09 | 2450 |    0 |     10 |
|  7934 | miller | clerk      | 7782 | 1982-01-23 | 1300 |    0 |     10 |
|  7788 | scott  | analyst    | 7566 | 1987-04-19 | 3000 |    0 |     20 |
|  7902 | ford   | analyst    | 7566 | 1981-12-03 | 3000 |    0 |     20 |
|  7566 | jones  | manager    | 7839 | 1981-04-02 | 2975 |    0 |     20 |
|  7876 | adams  | clerk      | 7788 | 1987-05-23 | 1100 |    0 |     20 |
|  7369 | smith  | clerk      | 7902 | 1980-12-17 |  800 |    0 |     20 |
|  7698 | blake  | manager    | 7839 | 1981-05-01 | 2850 |    0 |     30 |
|  7499 | allen  | salesman   | 7698 | 1981-02-20 | 1600 |  300 |     30 |
|  7844 | turner | salesman   | 7698 | 1981-09-08 | 1500 |    0 |     30 |
|  7521 | ward   | salesman   | 7698 | 1981-02-22 | 1250 |  500 |     30 |
|  7654 | martin | salesman   | 7698 | 1981-09-28 | 1250 | 1400 |     30 |
|  7900 | james  | clerk      | 7698 | 1981-12-03 |  950 |    0 |     30 |
+-------+--------+------------+------+------------+------+------+--------+
```

排序字段中的null默认排在最前面：查询所有员工信息按comm升序显示

```
select *
from emp
order by comm asc;
+-------+--------+------------+------+------------+------+------+--------+
| empno | ename  | job        | mgr  | hiredate   | sal  | comm | deptno |
+-------+--------+------------+------+------------+------+------+--------+
|  7369 | smith  | clerk      | 7902 | 1980-12-17 |  800 |    0 |     20 |
|  7566 | jones  | manager    | 7839 | 1981-04-02 | 2975 |    0 |     20 |
```

```
|  7698 | blake  | manager   | 7839 | 1981-05-01 | 2850 |    0 |     30 |
|  7782 | clark  | manager   | 7839 | 1981-06-09 | 2450 |    0 |     10 |
|  7788 | scott  | analyst   | 7566 | 1987-04-19 | 3000 |    0 |     20 |
|  7839 | king   | persident | NULL | 1981-11-17 | 5000 |    0 |     10 |
|  7844 | turner | salesman  | 7698 | 1981-09-08 | 1500 |    0 |     30 |
|  7876 | adams  | clerk     | 7788 | 1987-05-23 | 1100 |    0 |     20 |
|  7900 | james  | clerk     | 7698 | 1981-12-03 |  950 |    0 |     30 |
|  7902 | ford   | analyst   | 7566 | 1981-12-03 | 3000 |    0 |     20 |
|  7934 | miller | clerk     | 7782 | 1982-01-23 | 1300 |    0 |     10 |
|  7499 | allen  | salesman  | 7698 | 1981-02-20 | 1600 |  300 |     30 |
|  7521 | ward   | salesman  | 7698 | 1981-02-22 | 1250 |  500 |     30 |
|  7654 | martin | salesman  | 7698 | 1981-09-28 | 1250 | 1400 |     30 |
+-------+--------+-----------+------+------------+------+------+--------+
```

五.限制查询结果数量

在使用SELECT 语句时，往往返回的是所有匹配的行，有些时候我们仅需要返回第一行或者前几行，这时候就需要用到LIMT 子句。

```
select <字段名列表> from <表名> limit [偏移量,] 行数;
```

- lmit接受一个或两个数字参数，参数必须是一个整数常量。
- 第一个参数指定第一个返回记录行的偏移量，第二个参数指定返回记录行的最大数目。
- 如果只给定一个参数，表示返回最大的记录行数目。
- 初始记录行的偏移量是0(而不是1)。

给定一个参数：查询基本工资最高的前5位员工

```
select ename,sal
from emp
order by sal desc
limit 5;
+-------+------+
| ename | sal  |
+-------+------+
| king  | 5000 |
| ford  | 3000 |
| scott | 3000 |
| jones | 2975 |
| blake | 2850 |
+-------+------+
```

给定两个参数：查询基本工资第6到10名的员工

```
select ename,sal
from emp
order by sal desc
limit 5,5;
+--------+------+
| ename  | sal  |
+--------+------+
| clark  | 2450 |
| allen  | 1600 |
| turner | 1500 |
| miller | 1300 |
| ward   | 1250 |
+--------+------+
```

六.多表查询

随着我们使用数据库的灵活性越来越高，当我们需要的数据在不同的表中时，就需要使用多表查询来检索我们需要的数据。

多表查询分为横向连接查询和纵向合并查询。

1.连接查询

连接查询是通过多张表中共有的关键字段，将多张表连成一张虚拟结果集，来补充字段信息，我们可以按照业务需求检索多张表中的数据。

SQL查询的基本原理：

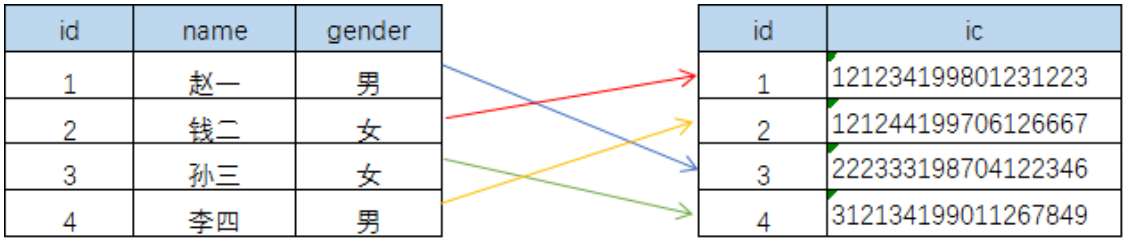
- 单表查询：根据where条件过滤表中的记录，然后根据select指定的列返回查询结果。
- 两表连接查询：使用on条件对两表进行连接形成一张虚拟结果集；然后根据where条件过滤结果集中的记录，再根据select指定的列返回查询结果。
- 多表连接查询：先对第一个和第二个表按照两表连接查询，然后用用连接后的虚拟结果集和第三个表做连接查询，以此类推，直到所有的表都连接上为止，最终形成一张虚拟结果集，然后根据where条件过滤虚拟结果集中的记录，再根据select指定的列返回查询结果。

多表连接的结果通过三个属性决定：

- 方向性：在外连接中写在前边的表为左表、写在后边的表为右表。
- 主附关系：主表要出所有的数据范围，附表与主表无匹配项时标记为null，内连接时无主附表之分。
- 对应关系：关键字段中有重复值的表为多表，没有重复值的表为一表。

对应关系：

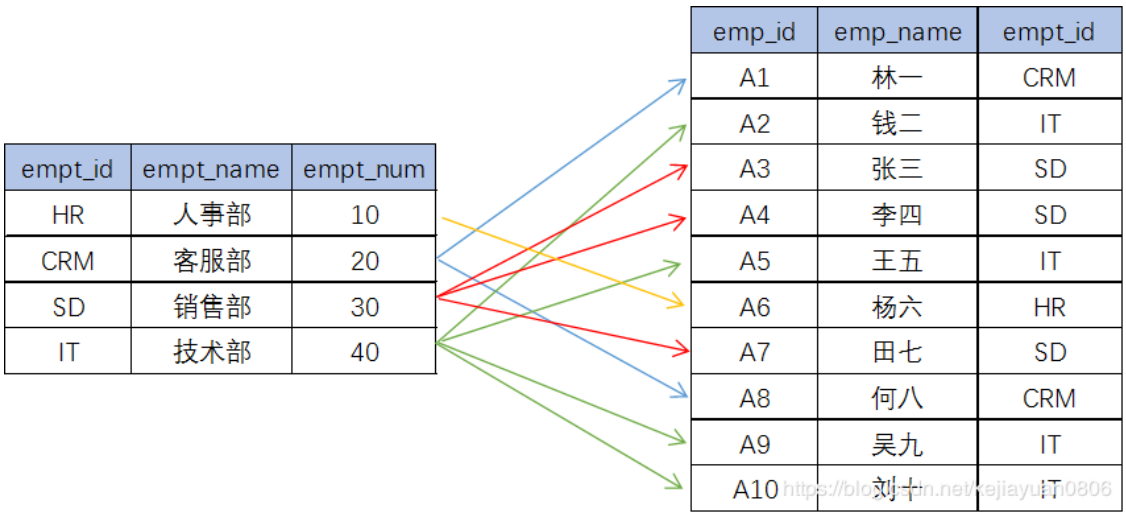
- 一对一



id	name	gender
1	赵一	男
2	钱二	女
3	孙三	女
4	李四	男

id	ic
1	121234199801231223
2	121244199706126667
3	222333198704122346
4	312134199011267849

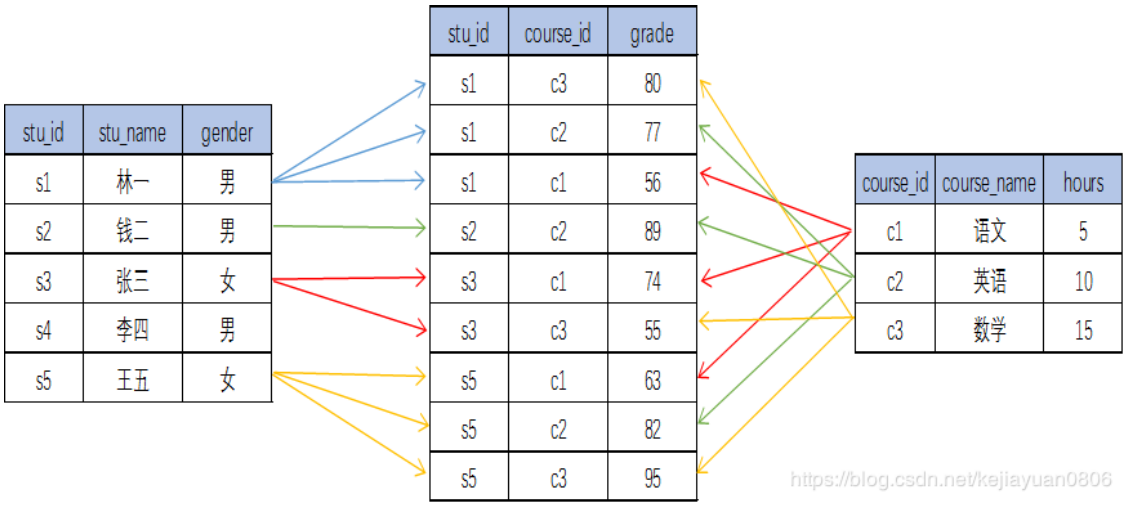
- 一对多或多对一



empt_id	empt_name	empt_num
HR	人事部	10
CRM	客服部	20
SD	销售部	30
IT	技术部	40

emp_id	emp_name	empt_id
A1	林一	CRM
A2	钱二	IT
A3	张三	SD
A4	李四	SD
A5	王五	IT
A6	杨六	HR
A7	田七	SD
A8	何八	CRM
A9	吴九	IT
A10	刘十	IT

- 多对多



stu_id	stu_name	gender
s1	林一	男
s2	钱二	男
s3	张三	女
s4	李四	男
s5	王五	女

stu_id	course_id	grade
s1	c3	80
s1	c2	77
s1	c1	56
s2	c2	89
s3	c1	74
s3	c3	55
s5	c1	63
s5	c2	82
s5	c3	95

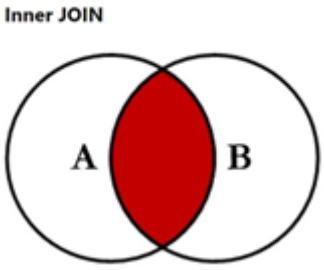
course_id	course_name	hours
c1	语文	5
c2	英语	10
c3	数学	15

1.1连接方式

MySQL支持的连接方式：内连接和外连接（左外连接、右外连接）

- **内连接**：按照连接条件，返回两张表中满足条件的记录。

```
select <字段名1>[,<字段名2>...,<字段名n>] from <表名1>[ inner] join <表名2> on <连接条件>;
```



等值连接：连接条件是两张表中的关键字段取值相等。

```
select ename,job,hiredate,sal,dname
from emp
inner join dept
on emp.deptno=dept.deptno;
+--------+-----------+------------+------+------------+
| ename  | job       | hiredate   | sal  | dname      |
+--------+-----------+------------+------+------------+
```

```
| clark  | manager   | 1981-06-09 | 2450 | accounting |
| king   | persident | 1981-11-17 | 5000 | accounting |
| miller | clerk     | 1982-01-23 | 1300 | accounting |
| smith  | clerk     | 1980-12-17 |  800 | research   |
| jones  | manager   | 1981-04-02 | 2975 | research   |
| scott  | analyst   | 1987-04-19 | 3000 | research   |
| adams  | clerk     | 1987-05-23 | 1100 | research   |
| ford   | analyst   | 1981-12-03 | 3000 | research   |
| allen  | salesman  | 1981-02-20 | 1600 | sales      |
| ward   | salesman  | 1981-02-22 | 1250 | sales      |
| martin | salesman  | 1981-09-28 | 1250 | sales      |
| blake  | manager   | 1981-05-01 | 2850 | sales      |
| turner | salesman  | 1981-09-08 | 1500 | sales      |
| james  | clerk     | 1981-12-03 |  950 | sales      |
+--------+-----------+------------+------+------------+
```

不等值连接：连接条件是两张表中的关键字段取值满足非等值比较运算。

```
select ename,job,hiredate,sal,grade
from emp
inner join salgrade
on sal between losal and hisal;
+--------+-----------+------------+------+-------+
| ename  | job       | hiredate   | sal  | grade |
+--------+-----------+------------+------+-------+
| smith  | clerk     | 1980-12-17 |  800 |     1 |
| allen  | salesman  | 1981-02-20 | 1600 |     3 |
| ward   | salesman  | 1981-02-22 | 1250 |     2 |
| jones  | manager   | 1981-04-02 | 2975 |     4 |
| martin | salesman  | 1981-09-28 | 1250 |     2 |
| blake  | manager   | 1981-05-01 | 2850 |     4 |
| clark  | manager   | 1981-06-09 | 2450 |     4 |
| scott  | analyst   | 1987-04-19 | 3000 |     4 |
| king   | persident | 1981-11-17 | 5000 |     5 |
| turner | salesman  | 1981-09-08 | 1500 |     3 |
| adams  | clerk     | 1987-05-23 | 1100 |     1 |
| james  | clerk     | 1981-12-03 |  950 |     1 |
| ford   | analyst   | 1981-12-03 | 3000 |     4 |
| miller | clerk     | 1982-01-23 | 1300 |     2 |
+--------+-----------+------------+------+-------+
```

笛卡尔积连接：两张表中的每一条记录进行笛卡尔积组合，然后根据where条件过滤虚拟结果集中的记录。

```
select <字段名1>[,<字段名2>...,<字段名n>] from <表名1>,<表名2> where <筛选条件>;
```

```
select ename,job,hiredate,sal,dname
from emp,dept
where emp.deptno=dept.deptno;
+--------+-----------+------------+------+------------+
| ename  | job       | hiredate   | sal  | dname      |
+--------+-----------+------------+------+------------+
| clark  | manager   | 1981-06-09 | 2450 | accounting |
| king   | persident | 1981-11-17 | 5000 | accounting |
| miller | clerk     | 1982-01-23 | 1300 | accounting |
```

```
| smith  | clerk     | 1980-12-17 |  800 | research   |
| jones  | manager   | 1981-04-02 | 2975 | research   |
| scott  | analyst   | 1987-04-19 | 3000 | research   |
| adams  | clerk     | 1987-05-23 | 1100 | research   |
| ford   | analyst   | 1981-12-03 | 3000 | research   |
| allen  | salesman  | 1981-02-20 | 1600 | sales      |
| ward   | salesman  | 1981-02-22 | 1250 | sales      |
| martin | salesman  | 1981-09-28 | 1250 | sales      |
| blake  | manager   | 1981-05-01 | 2850 | sales      |
| turner | salesman  | 1981-09-08 | 1500 | sales      |
| james  | clerk     | 1981-12-03 |  950 | sales      |
+--------+-----------+------------+------+------------+
```

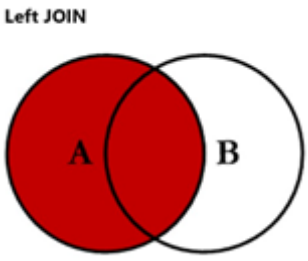
自连接：通过设置表别名，将一张表虚拟成多张表。

```
select t1.ename as 员工姓名,t2.ename as 领导姓名
from emp as t1
inner join emp as t2
on t1.mgr=t2.empid;
+----------+----------+
| 员工姓名 | 领导姓名 |
+----------+----------+
| smith    | ford     |
| allen    | blake    |
| ward     | blake    |
| jones    | king     |
| martin   | blake    |
| blake    | king     |
| clark    | king     |
| scott    | jones    |
| turner   | blake    |
| adams    | scott    |
| james    | blake    |
| ford     | jones    |
| miller   | clark    |
+----------+----------+
```

表限定符.：如果表1和表2中的字段名相同，则必须使用表限定符.指定引用的是哪个表中的字段。

- **左连接**：按照连接条件，返回两张表中满足条件的记录，以及左表中的所有记录，右表匹配不到显示为null。

```
select <字段名1>[,<字段名2>...,<字段名n>] from <表名1> left join <表名2> on <连接条件>;
```

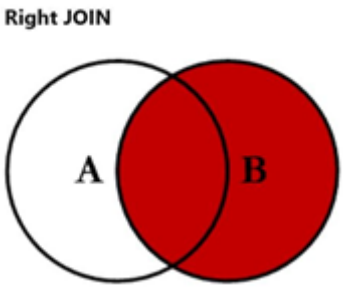


示例：查询每个部门的员工数（没有员工的部门，员工数统计为0）

```
select dept.deptno,dname,count(empid)
from dept
left join emp
on dept.deptno=emp.deptno
group by dept.deptno;
+--------+------------+--------------+
| deptno | dname      | count(empid) |
+--------+------------+--------------+
|     10 | accounting |            3 |
|     20 | research   |            5 |
|     30 | sales      |            6 |
|     40 | operations |            0 |
+--------+------------+--------------+
```

- **右连接**：按照连接条件，返回两张表中满足条件的记录，以及右表中的所有记录，左表匹配不到显示为null。

```
select <字段名1>[,<字段名2>...,<字段名n>] from <表名1> right join <表名2> on <连接条件>;
```



示例：查询每个部门的员工数（没有员工的部门，员工数统计为0）

```
select dept.deptno,dname,count(empid)
from emp
right join dept
on dept.deptno=emp.deptno
group by dept.deptno;
+--------+------------+--------------+
| deptno | dname      | count(empid) |
+--------+------------+--------------+
|     10 | accounting |            3 |
|     20 | research   |            5 |
|     30 | sales      |            6 |
|     40 | operations |            0 |
+--------+------------+--------------+
```

1.2连接规则

一表作为主表可以保证维度的完整性，多表作为主表可以保证度量的准确性。

在没有明确表示需要保证维度完整性的情况下，优先保证度量的准确性，所以将度量值所在的表作为主表。

度量字段通常存在于多表中，因此通常情况下可以将多表作为主表进行外连接。

- 确定查询的信息在哪几张表
- 确定表和表之间的对应关系和主附关系

- 确定表和表之间的连接条件

2.合并查询

纵向合并查询又称为联合查询，它是用union关键字把多条select语句的查询结果合并为一个结果集。

纵向合并的前提是被合并的结果集的字段数量、顺序和数据类型必须完全一致。字段名不一样的情况下，会将第一个结果集的字段名作为合并后的虚拟结果集的字段名。

```
select <字段1>[,<字段2>,…] from <表名1>
union[ all]
select <字段1>[,<字段2>,…] from <表名2>;
```

示例:

```
create table t1(key1 char,v1 int);
create table t2(key2 char,v2 int);

insert into t1 values('a',1),('a',2),('b',3),('c',4),('a',13);
insert into t2 values('b',10),('b',11),('a',12),('a',13),('e',14);
```

union去重:

```
select * from t1
union
select * from t2;
+------+------+
| key1 | v1   |
+------+------+
| a    |    1 |
| a    |    2 |
| b    |    3 |
| c    |    4 |
| a    |   13 |
| b    |   10 |
| b    |   11 |
| a    |   12 |
| e    |   14 |
+------+------+
```

union all不去重:

```
select * from t1
union all
select * from t2;
+------+------+
| key1 | v1   |
+------+------+
| a    |    1 |
| a    |    2 |
| b    |    3 |
| c    |    4 |
| a    |   13 |
| b    |   10 |
| b    |   11 |
| a    |   12 |
```

```
| a    |   13 |
| e    |   14 |
+------+------+
```

七.子查询

子查询又称为嵌套查询，它是指在一个select语句中包含另一个或多个完整的select语句。

子查询的语法规则：

- 子查询需要用圆括号括起来。
- 子查询最多可以嵌套到32层（个别查询可能会不支持32层嵌套）。
- 执行顺序由内到外，先执行内部的子查询，再执行外部的主查询。

1.子查询分类

按子查询返回的结果，可分为：

- 标量子查询：返回的结果是一个数据（单行单列）
- 行子查询：返回的结果是一行（单行多列）
- 列子查询：返回的结果是一列（多行单列）
- 表子查询：返回的结果是一张临时表（多行多列）

按子查询出现的位置，可分为：

- select子句中：将子查询返回的结果作为主查询的计算字段（标量子查询）
- where或having子句中：将子查询返回的结果作为主查询的筛选条件（标量子查询、行子查询、列子查询）
- from子句中：将子查询返回的结果作为主查询的一张表（表子查询）

2.子查询常用运算符

子查询出现在where或having子句中时，可以使用>、>=、<、<=、=、<>/!=等比较运算符或[not]in、any/some、all、[not]exists等操作符进行条件筛选。

操作符	说明	示例
[not]in	在/不在其中	where <字段名> [not]in (子查询)
any/some	其中任意一个/至少一个	where <字段名> <比较> any/some (子查询)
all	全部、每一个	where <字段名> <比较> all (子查询)
[not]exists	（不）存在	where exists (子查询)

标量子查询：查询基本工资高于公司平均工资的员工信息

```
select *
from emp
where sal > (select avg(sal) from emp);
+-------+-------+-----------+------+------------+------+------+--------+
| empid | ename | job       | mgr  | hiredate   | sal  | comm | deptno |
+-------+-------+-----------+------+------------+------+------+--------+
|  7566 | jones | manager   | 7839 | 1981-04-02 | 2975 | NULL |     20 |
|  7698 | blake | manager   | 7839 | 1981-05-01 | 2850 | NULL |     30 |
|  7782 | clark | manager   | 7839 | 1981-06-09 | 2450 | NULL |     10 |
|  7788 | scott | analyst   | 7566 | 1987-04-19 | 3000 | NULL |     20 |
|  7839 | king  | persident | NULL | 1981-11-17 | 5000 | NULL |     10 |
|  7902 | ford  | analyst   | 7566 | 1981-12-03 | 3000 | NULL |     20 |
+-------+-------+-----------+------+------------+------+------+--------+
```

行子查询：查询和smith同部门同职位的员工

```
select *
from emp
where (deptno,job)=(select deptno,job from emp where ename='smith') and
ename<>'smith';
+-------+-------+-------+------+------------+------+------+--------+
| empid | ename | job   | mgr  | hiredate   | sal  | comm | deptno |
+-------+-------+-------+------+------------+------+------+--------+
|  7876 | adams | clerk | 7788 | 1987-05-23 | 1100 | NULL |     20 |
+-------+-------+-------+------+------------+------+------+--------+
```

列子查询：where子句中可使用[not]in、any/some、all、[not]exists等操作符进行条件筛选。

[not]in子查询：查询普通员工的信息

```
select *
from emp
where empid not in (select mgr from emp where mgr is not null);
+-------+--------+----------+------+------------+------+------+--------+
| empid | ename  | job      | mgr  | hiredate   | sal  | comm | deptno |
+-------+--------+----------+------+------------+------+------+--------+
|  7369 | smith  | clerk    | 7902 | 1980-12-17 |  800 | NULL |     20 |
|  7499 | allen  | salesman | 7698 | 1981-02-20 | 1600 |  300 |     30 |
|  7521 | ward   | salesman | 7698 | 1981-02-22 | 1250 |  500 |     30 |
|  7654 | martin | salesman | 7698 | 1981-09-28 | 1250 | 1400 |     30 |
|  7844 | turner | salesman | 7698 | 1981-09-08 | 1500 |    0 |     30 |
|  7876 | adams  | clerk    | 7788 | 1987-05-23 | 1100 | NULL |     20 |
|  7900 | james  | clerk    | 7698 | 1981-12-03 |  950 | NULL |     30 |
|  7934 | miller | clerk    | 7782 | 1982-01-23 | 1300 | NULL |     10 |
+-------+--------+----------+------+------------+------+------+--------+
```

any/some子查询：查询基本工资高于30号部门任意员工的员工信息

```
select *
from emp
where sal>any(select sal from emp where deptno=30) and deptno<>30;
+-------+--------+-----------+------+------------+------+------+--------+
| empid | ename  | job       | mgr  | hiredate   | sal  | comm | deptno |
+-------+--------+-----------+------+------------+------+------+--------+
|  7782 | clark  | manager   | 7839 | 1981-06-09 | 2450 | NULL |     10 |
|  7839 | king   | persident | NULL | 1981-11-17 | 5000 | NULL |     10 |
|  7934 | miller | clerk     | 7782 | 1982-01-23 | 1300 | NULL |     10 |
|  7566 | jones  | manager   | 7839 | 1981-04-02 | 2975 | NULL |     20 |
|  7788 | scott  | analyst   | 7566 | 1987-04-19 | 3000 | NULL |     20 |
|  7876 | adams  | clerk     | 7788 | 1987-05-23 | 1100 | NULL |     20 |
|  7902 | ford   | analyst   | 7566 | 1981-12-03 | 3000 | NULL |     20 |
+-------+--------+-----------+------+------------+------+------+--------+
```

all子查询：查询基本工资高于30号部门所有员工的员工信息

```
select *
from emp
where sal>all(select sal from emp where deptno=30);
+-------+-------+-----------+------+------------+------+------+--------+
| empid | ename | job       | mgr  | hiredate   | sal  | comm | deptno |
+-------+-------+-----------+------+------------+------+------+--------+
|  7566 | jones | manager   | 7839 | 1981-04-02 | 2975 | NULL |     20 |
|  7788 | scott | analyst   | 7566 | 1987-04-19 | 3000 | NULL |     20 |
|  7839 | king  | persident | NULL | 1981-11-17 | 5000 | NULL |     10 |
|  7902 | ford  | analyst   | 7566 | 1981-12-03 | 3000 | NULL |     20 |
+-------+-------+-----------+------+------------+------+------+--------+
```

exists子查询：如果dept表中存在30号部门则查询该部门的员工信息

```
select *
from emp
where deptno=30 and exists (select deptno from dept where deptno=30);
+-------+--------+----------+------+------------+------+------+--------+
| empid | ename  | job      | mgr  | hiredate   | sal  | comm | deptno |
+-------+--------+----------+------+------------+------+------+--------+
|  7499 | allen  | salesman | 7698 | 1981-02-20 | 1600 |  300 |     30 |
|  7521 | ward   | salesman | 7698 | 1981-02-22 | 1250 |  500 |     30 |
|  7654 | martin | salesman | 7698 | 1981-09-28 | 1250 | 1400 |     30 |
|  7698 | blake  | manager  | 7839 | 1981-05-01 | 2850 | NULL |     30 |
|  7844 | turner | salesman | 7698 | 1981-09-08 | 1500 |    0 |     30 |
|  7900 | james  | clerk    | 7698 | 1981-12-03 |  950 | NULL |     30 |
+-------+--------+----------+------+------------+------+------+--------+
```

表子查询：查询各部门工资最高的员工

```
select emp.*
from emp
join (select deptno,max(sal) as max_sal from emp group by deptno) as t
on emp.deptno=t.deptno
where sal=max_sal;
+-------+-------+-----------+------+------------+------+------+--------+
| empid | ename | job       | mgr  | hiredate   | sal  | comm | deptno |
+-------+-------+-----------+------+------------+------+------+--------+
|  7698 | blake | manager   | 7839 | 1981-05-01 | 2850 | NULL |     30 |
|  7788 | scott | analyst   | 7566 | 1987-04-19 | 3000 | NULL |     20 |
|  7839 | king  | persident | NULL | 1981-11-17 | 5000 | NULL |     10 |
|  7902 | ford  | analyst   | 7566 | 1981-12-03 | 3000 | NULL |     20 |
+-------+-------+-----------+------+------------+------+------+--------+
```

3.子查询优化

MySQL从4.1版本开始支持子查询，使用子查询进行SELECT语句嵌套查询，可以一次完成很多逻辑上需要多个步骤才能完成的SQL操作。

子查询虽然很灵活，但是执行效率并不高。执行子查询时，MySQL需要为内层子查询的查询结果建立一个临时表，然后外层主查询在临时表上进行查询和筛选。查询完毕后再撤销这些临时表，这里多了一个创建和销毁临时表的过程。因此，子查询的速度会受到一定的影响，如果查询的数据量比较大，这种影响就会随之增大。

优化方法：可以使用连接查询（join）代替子查询，连接查询不需要建立临时表，因此其速度比子查询快。

所有的连接查询都可以替换为子查询，但并不是所有的子查询都可以用连接查询代替。当where子句中需要使用聚合函数作为筛选条件时，只能使用子查询。