数据库

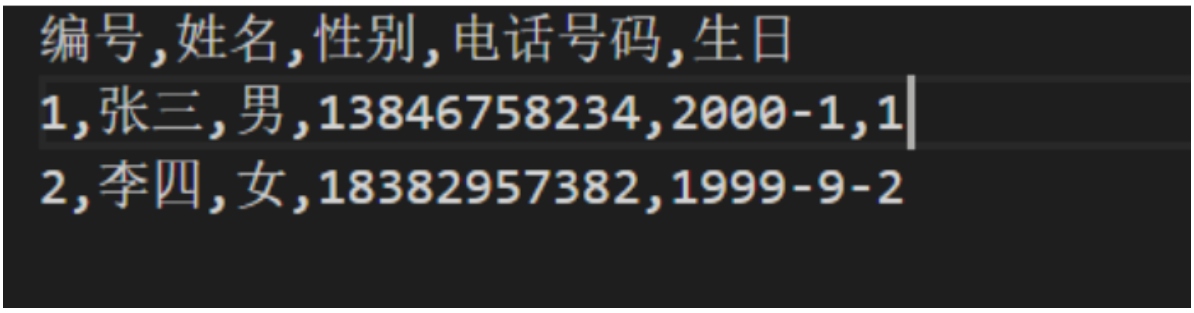
数据库概述

1.1 数据库概述

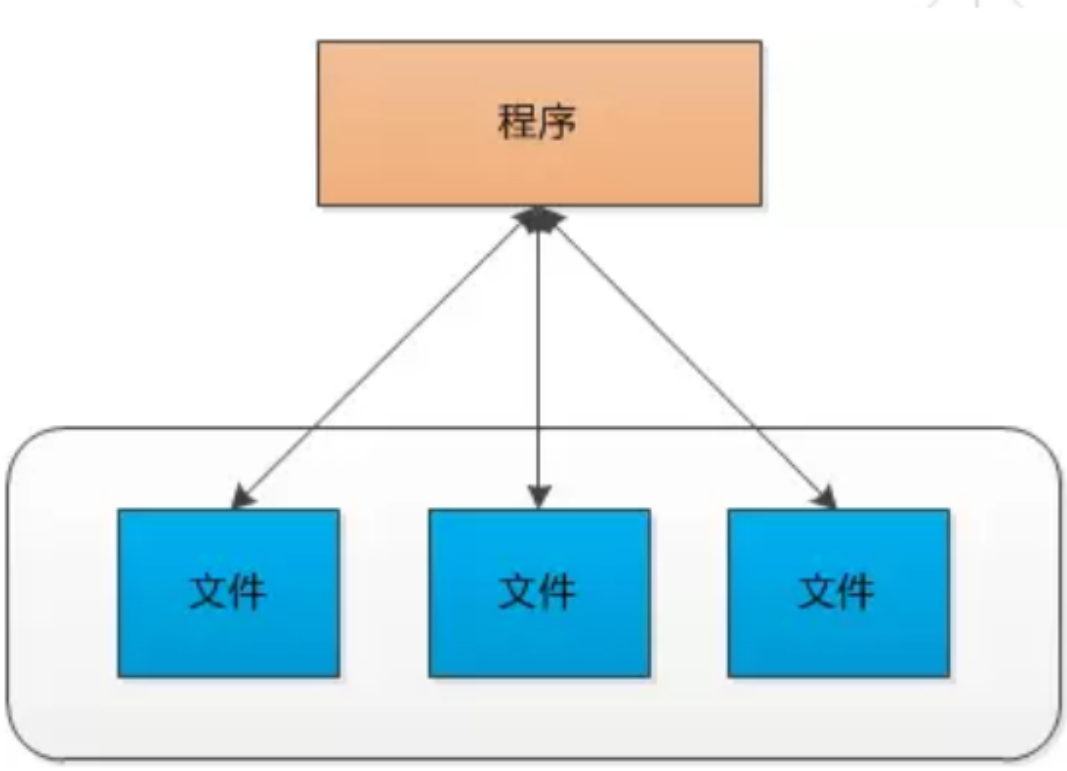
什么是数据库

数据库就是存储数据的仓库，其本质是一个文件系统，数据按照特定的格式将数据存储起来，用户可以对数据库中的数据进行增加，修改，删除及查询操作
一个小例子

假设我们现在没有数据库,我们想开发一个本地的电话本软件,也就是手机中通讯录,这个软件有记录的功能,需要记录联系人姓名,电话号码,生日,性别等信息,由于是要持久化数据,所以我们只能写到文件中,比如phone.txt.例如:

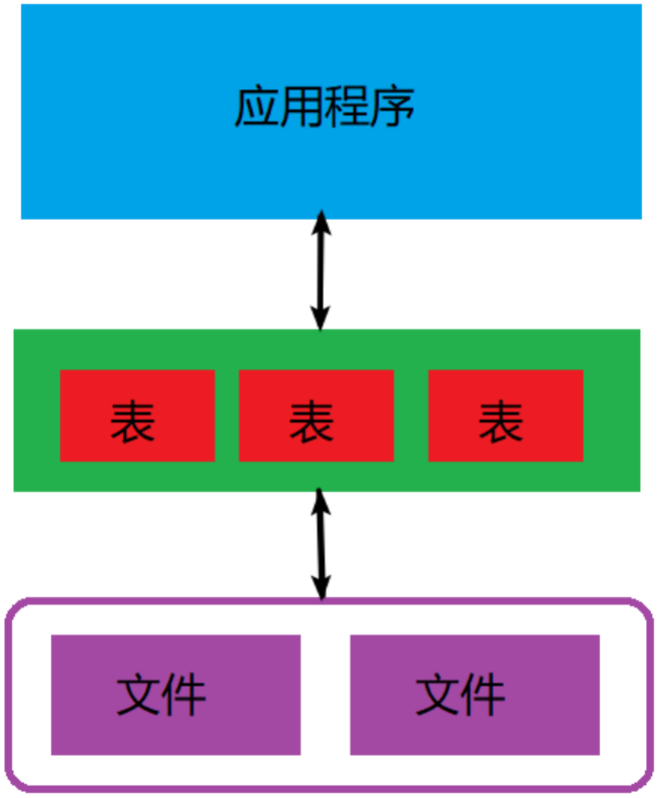


第一行是表头,其他行是内容,数据之间用逗号分隔,每行是一条数据,这样设计完成之后就可以按行读取,并且能够按照逗号进行拆分存入到文件中去了,现在的程序架构是这个样子的:



功能被完美的实现了,但是随着程序的演进发现了一些问题,例如想要找到所有的男性电话,或者查找今天过生日的人来给他们发一些过节短信等等的功能,需要不断的编码,而本质上都是在读取其中的文件信息,并且最要命的是无论什么功能都需要把所有的硬盘数据先加载进内存当中,即使这些数据时不需要的,那么如何能做到屏蔽掉文件系统,只读取一些需要的数据呢,编码界有一句老话"所有的计算机问题都可以通过

增加一个中间层来解决",于是你觉定增加一个中间层,这个中间层上有逻辑的数据结构,其实就是[编号,姓名,性别,电话号码,生日]这些东西, 这些东西被叫做表,而其中的每一项被称为"列",每一列都要有类型,例如字符型,日期型,数字型等等,并且可以使用专业的语句来进行查询,我们决定叫它SQL,即Structured Query Language 结构化查询语言,那么现在软件的架构变成了这种



同时由于在程序和物理层之间抽象了一层,所以在优化物理层存储的时候,可以不影响上层应用程序的逻辑,可以使用索引,B+树等缓存手段了

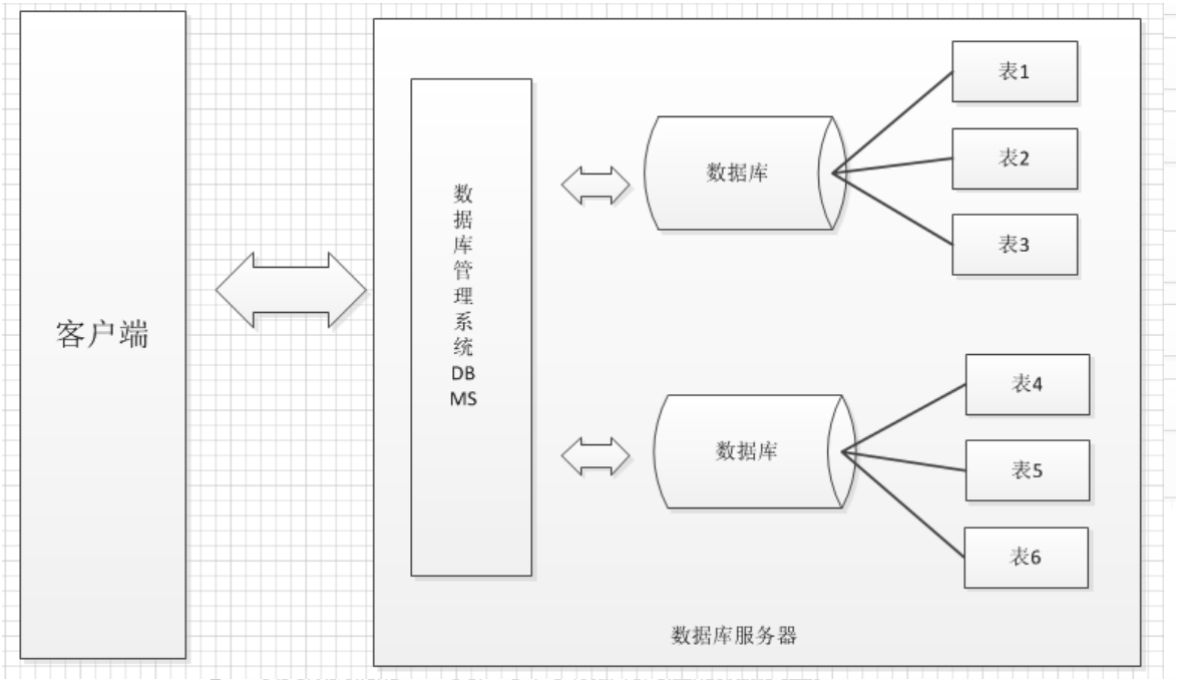
这就是关系型数据库

1.2 数据库管理系统

什么是数据库管理系统
数据库管理系统（DataBase Management System，DBMS）：指一种操作和管理数据库的大型软件，用于建立、使用和维护数据库，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表内的数据。
常见的数据库管理系统

```
MYSQL：开源免费的数据库，小型的数据库.已经被Oracle收购了.MySQL6.x版本也开始收费。
Oracle：收费的大型数据库，Oracle公司的产品。Oracle收购SUN公司，收购MYSQL。
DB2：IBM公司的数据库产品,收费的。常应用在银行系统中
SQLServer：MicroSoft 公司收费的中型的数据库。C#、.net等语言常使用
SQLite: 嵌入式的小型数据库，应用在手机端
```

数据库与数据库管理系统的关系



数据库主要用来存储、维护和管理数据的集合【容器】；
数据库管理系统就是一款数据库软件，可以对数据库进行管理和控制；
简单来说，相互依赖的关系（数据库中的数据是数据库管理系统来操作的，要是没有数据，后者也就没什么用了）

1.3 数据库表

通常，我们需要把从用户处收集到的数据进行存储，以便日后使用，这个概念就是程序猿经常说的数据持久化。当然实现数据持久化的方案很多，写数据库只是其中一个。

数据库中最频繁提到的一个概念就是“表”。其实，“表”就是人为地总结出来的、用以描述一类事物的一组描述信息，包括 1个或多个方面的信息。比如人的描述信息可以包括（姓名、年龄、职业）等。表中的每一条信息称为一条记录，比如某一个具体人的信息形成“人”这张表中的一条记录。

表结构：

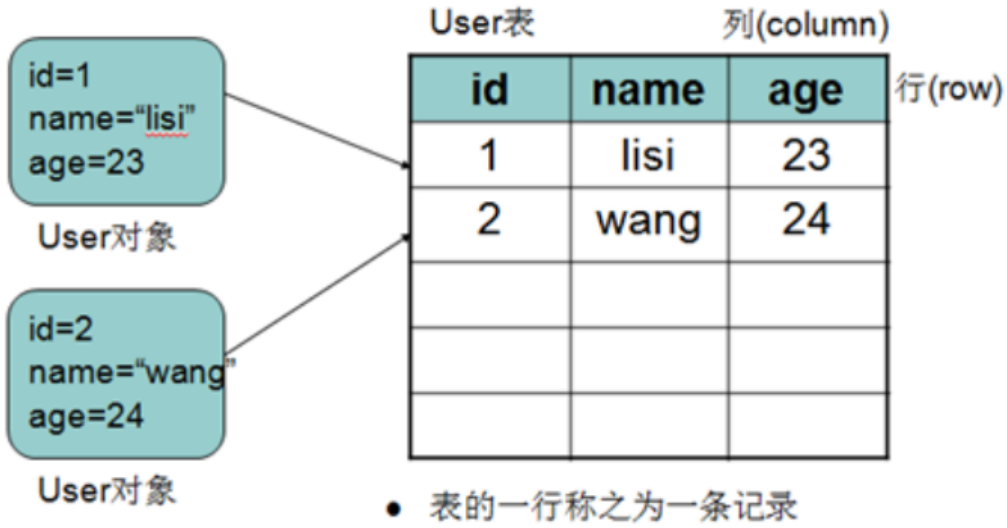
姓名
年龄
职业

表中的记录：

姓名	年龄	职业
张三	8	学生
李四	28	码农

1.4 数据表

根据表字段所规定的数据类型，我们可以向其中填入一条条的数据，而表中的每条数据类似类的实例对象。表中的一行一行的信息我们称之为记录



SQL语句

结构化查询语言(Structured Query Language)简称SQL，结构化查询语言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

sql 语句就是对数据库进行操作的一种语言。

2.1 SQL分类

- **数据定义语言**：简称DDL(Data Definition Language)，用来定义数据库对象：数据库，表，列等。关键字：create，alter，drop等
- **数据操作语言**：简称DML(Data Manipulation Language)，用来对数据库中表的记录进行更新。关键字：insert，delete，update等
- **数据控制语言**：简称DCL(Data Control Language)，用来定义数据库的访问权限和安全级别，及创建用户
- **数据查询语言**：简称DQL(Data Query Language)，用来查询数据库中表的记录。关键字：select，from，where等

2.2 SQL通用语法

- SQL语句可以单行或多行书写，以分号结尾
- 可使用空格和缩进来增强语句的可读性
- MySQL数据库的SQL语句不区分大小写，建议使用大写，例如：SELECT * FROM user
- 同样可以使用/**/的方式完成注释
- MySQL中常用的数据类型如下:

类型	描述
int	整型
double	浮点型
varchar	字符串类型
date	日期类型,格式为yyyy-MM-dd,只有年月日,没有时分秒

详细的数据类型

MySQL支持多种类型，大致可以分为三类：数值、日期/时间和字符串(字符)类型

2.3 数值类型

MySQL支持所有标准SQL数值数据类型。
这些类型包括严格数值数据类型(INTEGER、SMALLINT、DECIMAL和NUMERIC)，以及近似数值数据类型(FLOAT、REAL和DOUBLE PRECISION)。
关键字INT是INTEGER的同义词，关键字DEC是DECIMAL的同义词。
BIT数据类型保存位字段值，并且支持MyISAM、MEMORY、InnoDB和BDB表。
作为SQL标准的扩展，MySQL也支持整数类型TINYINT、MEDIUMINT和BIGINT。下面的表显示了需要的每个整数类型的存储和范围。

类型	大小	范围（有符号）	范围（无符号）	用途
TINYINT	1 Bytes	(-128，127)	(0，255)	小整数值
SMALLINT	2 Bytes	(-32 768，32 767)	(0，65 535)	大整数值
MEDIUMINT	3 Bytes	(-8 388 608，8 388 607)	(0，16 777 215)	大整数值
INT或INTEGER	4 Bytes	(-2 147 483 648，2 147 483 647)	(0，4 294 967 295)	大整数值
BIGINT	8 Bytes	(-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 Bytes	(-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 Bytes	(-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

2.4 日期和时间类型

表示时间值的日期和时间类型为DATETIME、DATE、TIMESTAMP、TIME和YEAR。

每个时间类型有一个有效值范围和一个"零"值，当指定不合法的MySQL不能表示的值时使用"零"值。

TIMESTAMP类型有专有的自动更新特性，将在后面描述。

类型	大小 (bytes)	范围	格式	用途
DATE	3	1000-01-01/9999-12-31	YYYY-MM-DD	日期值
TIME	3	'-838:59:59'/'838:59:59'	HH:MM:SS	时间值或持续时间
YEAR	1	1901/2155	YYYY	年份值
DATETIME	8	1000-01-01 00:00:00/9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	混合日期和时间值
TIMESTAMP	4	1970-01-01 00:00:00/2038 结束时间是第 **2147483647** 秒，北京时间 **2038-1-19 11:14:07**，格林尼治时间 2038年1月19日 凌晨 03:14:07	YYYYMMDD HHMMSS	混合日期和时间值，时间戳

2.5 字符串类型

字符串类型指CHAR、VARCHAR、BINARY、VARBINARY、BLOB、TEXT、ENUM和SET。该节描述了这些类型如何工作以及如何在查询中使用这些类型。

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

注意：char(n) 和 varchar(n) 中括号中 n 代表字符的个数，并不代表字节个数，比如 CHAR(30) 就可以存储 30 个字符。

CHAR 和 VARCHAR 类型类似，但它们保存和检索的方式不同。它们的最大长度和是否尾部空格被保留等方面也不同。在存储或检索过程中不进行大小写转换。

BINARY 和 VARBINARY 类似于 CHAR 和 VARCHAR，不同的是它们包含二进制字符串而不要非二进制字符串。也就是说，它们包含字节字符串而不是字符字符串。这说明它们没有字符集，并且排序和比较基于列值字节的数值值。

BLOB 是一个二进制大对象，可以容纳可变数量的数据。有 4 种 BLOB 类型：TINYBLOB、BLOB、MEDIUMBLOB 和 LONGBLOB。它们区别在于可容纳存储范围不同。

有 4 种 TEXT 类型：TINYTEXT、TEXT、MEDIUMTEXT 和 LONGTEXT。对应的这 4 种 BLOB 类型，可存储的最大长度不同，可根据实际情况选择。

3. SQL

3.1 DDL

进入数据库

mysql -u账号 -p密码

```
mysql -uroot -p123456
```

数据库操作

- 创建数据库

```
create database 数据库名;
create database if not exists 数据库名; -- 如果不存在则创建
```

- 查看数据库

查看MySQL服务器中的所有数据库:

```
show databases;
```

查看某个数据库的定义信息

```
show create database 数据库名;
```

删除数据库

```
drop database 数据库名称;
```

切换数据库

```
use 数据库名;
```

表结构相关语句

- 创建表

格式:

```
create table 表名(
    字段名 类型(长度) 约束,
    字段名 类型(长度) 约束
);
```

例如:

```
CREATE TABLE sort (
  sid INT, #分类ID
  sname VARCHAR(100) #分类名称
);
CREATE TABLE sort (
  sid INT, #分类ID
  sname VARCHAR(100) #分类名称
)CHARSET=utf8;
```

- 主键约束
  主键是用于标识当前记录的字段。它的特点是非空，唯一。在开发中一般情况下主键是 **不具备任何含义**，只是用于标识当前记录

格式

1 在创建表时创建主键,在字段后面加上 primary key

```
CREATE TABLE 表名(
    id int primary key,
    ....
);
```

2 在创建表时创建主键,在表创建的最后来指定主键

```
CREATE TABLE 表名(
    id int,
    ....,
    primary key(id)
);
```

3 删除主键:

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

4 主键自增长(只适用于MySQL)

一般主键是自增长的字段,不需要指定,实现添加自增长语句,主键字段后加auto_increment,例如:

```
CREATE TABLE sort (
  sid INT PRIMARY KEY auto_increment, #分类ID
  sname VARCHAR(100) #分类名称
);
```

- 查看表

查看数据库中所有的表

```
show tables;
```

- 查看表结构

```
desc 表名;
```

```
mysql> show tables;
+--------------------+
| Tables_in_ssm_mall |
+--------------------+
| mall_cart          |
| mall_category      |
| mall_order         |
| mall_order_item    |
| mall_pay_info      |
| mall_product       |
| mall_shipping      |
| mall_user          |
+--------------------+
8 rows in set (0.00 sec)

mysql> desc mall_cart;
+-------------+----------+------+-----+---------+----------------+
| Field       | Type     | Null | Key | Default | Extra          |
+-------------+----------+------+-----+---------+----------------+
| id          | int(11)  | NO   | PRI | NULL    | auto_increment |
| user_id     | int(11)  | NO   | MUL | NULL    |                |
| product_id  | int(11)  | YES  |     | NULL    |                |
| quantity    | int(11)  | YES  |     | NULL    |                |
| checked     | int(11)  | YES  |     | NULL    |                |
| create_time | datetime | YES  |     | NULL    |                |
| update_time | datetime | YES  |     | NULL    |                |
+-------------+----------+------+-----+---------+----------------+
7 rows in set (0.02 sec)
```

查看建表语句

```
show create table 表名;
```

删除表

格式:

```
drop table 表名;
```

修改表结构

修改id为自增主键： modify

```
alter TABLE 表名 modify 字段 类型 primary key auto_increment;
```

删除列:

```
alter TABLE 表名 DROP 列名;
```

修改表名:

```
RENAME TABLE 表名 TO 新表名;
```

修改表的字符集

```
alter TABLE 表名 CHARACTER SET 字符集
```

修改列名

```
alter TABLE 表名 CHANGE 列名 新列名 列类型;
```

添加列

```
alter table 表名 add 列名 列类型;
```

mysql 常见ALTER TABLE操作

删除列

```
alter table table_name drop col_name;
```

增加列(单列)

```
alter table table_name add col_name col_type comment 'xxx';
```

增加列(多列)

alter table table_name **add** col_name col_type comment 'xxx', add col_name col_type(col_length) comment 'xxx';

增加表字段并指明字段放置为第一列

alter table table_name **add** col_name col_type COMMENT 'sss' FIRST;

增加表字段并指明字段放置为特定列后面

alter table table_name **add** col_name col_type after col_name_1;

使用MODIFY修改字段类型

alter table table_name **modify** column col_name col_type;

使用CHANGE修改字段类型

alter table table_name **change** col_name col_name col_type;

使用CHANGE修改字段名称

alter table table_name **change** old_col_name new_col_name col_type;

修改列类型、长度

alter table table_name **change** old_col_name new_col_name new_col_type;

查看表中列属性

show columns from table_name;

修改表名

rename table old_table_name to new-table-name;

为字段设置NULL和DEFAULT

alter table table_name modify col_name col_type not null default 100;

修改字段的默认值

alter table table_name alter col_name set default 10000;

字段删除默认值

alter table table_name alter col_name drop default;

3.2 DML操作

首先先知道查询表中所有数据的语句:

```
SELECT * FROM 表名;
```

DML是对表中的数据进行增、删、改的操作。不要与DDL混淆了,包含:

```
INSERT: 插入
UPDATE: 更新
DELETE: 删除
```

小知识:

```
在mysql中,字符串类型和日期类型都要用单引号括起来: ‘tom’ ‘2015-09-04’
空值：null
```

3.2.1 插入操作: INSERT:

语法:

```
INSERT INTO `表名` (`列名1`, 列名2 ...) VALUES(列值1, 列值2...);
```

插入多条数据

```
INSERT INTO `表名 (`列名1`, 列名2 ...) VALUES(列值1, 列值2...),(列值1, 列值2...),
(列值1, 列值2...);
```

把一张表的数据批量复制的另一张表:

```
insert into 数据库.表名（`列名1`, 列名2 ...) select `列名1`, 列名2 ... from 数据库.
表2
```

注意:

```
列名与与列值的类型、个数、顺序要一一对应
可以把列名当做python中的形参，把列值当做实参
值不要超出列定义的长度
如果插入空值，请使用null
插入的日期和字符一样，都使用引号括起来
```

注意：

1. 如果表名或者字段名和关键词冲突，使用着重号`将 内容包裹。
2. 插入数据时，非数值类型，必须带引号，数值类型带引号和不带引号，效果一样
3. 如果不带引号，程序会认为是列名：Unknown column '类名' in 'field list'

练习

创建表 emp 并插入数据,表结构如下创建表 emp 并插入数据,表结构如下

列名	列类型
id	int
name	varchar(100)
gender	varchar(10)
birthday	date
salary	float(10,2)
entry_date	date
resume	text

```sql
create table emp(
id int PRIMARY KEY auto_increment,
name varchar(100),
gender varchar(10),
birthday date,
salary float(10,2),
entry_date date,
resume text
);

INSERT INTO emp(id,name,gender,birthday,salary,entry_date,resume)
VALUES(1,'zhangsan','female','1990-5-10',10000,'2015-5-5-','good girl');

INSERT INTO emp(id,name,gender,birthday,salary,entry_date,resume)
VALUES(2,'lisi','male','1995-5-10',10000,'2015-5-5','good boy');

INSERT INTO emp(id,name,gender,birthday,salary,entry_date,resume)
VALUES(3,'wangwu','male','1995-5-10',10000,'2015-5-5','good boy');

-- 批量插入
INSERT INTO emp VALUES
(4,'zs','m','2015-09-01',10000,'2015-09-01',NULL),
(5,'li','m','2015-09-01',10000,'2015-09-01',NULL),
(6,'ww','m','2015-09-01',10000,'2015-09-01',NULL);
```

3.2.2 修改操作: UPDATE

语法:

```
UPDATE 表名 SET 列名1=列值1，列名2=列值2... WHERE 列名=值
```

练习

将所有员工薪水修改为5000元。

```
update emp set salary = 5000 ;
```

将姓名为zhangsan的员工薪水修改为3000元?

```
update emp set salary=3000 where name='zhangsan'
```

将姓名为lisi的员工薪水修改为4000元,gender改为female?

```
update emp set salary=4000,gender='female' where name='lisi';
```

将所有男性的薪水在原有基础上加1000

```
update emp set gender ='male' where gender='m';
update emp set salary = salary + 1000 where gender='male';
```

删除操作

语法

```
DELETE FROM 表名 [WHERE 列名=值]
```

注意：删除操作，先写where 条件 ， 没有where 删除与不要使用

练习

删除表中姓名为zhangsan的记录

```
delete from emp where name='zhangsan';
```

删除表中所有记录 (**尽量不要使用**)

```
delete from emp;
```

清空表数据:

```
truncate table 表名
```

3.3 DQL 操作

DQL数据查询语言 （重要）
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。
查询返回的结果集是一张 虚拟表。语法

```
 SELECT 列名 FROM表名
[WHERE -> GROUP BY -> HAVING -> ORDER BY]
```

```
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/

```

准备工作

创建表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);
```

创建 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

```sql
CREATE TABLE emp(
    empno       INT PRIMARY KEY auto_increment,
    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,'刘丹妮','店员',7902,'1980-12-17',800,NULL,20);
INSERT INTO emp values(7499,'苏园园','售货员',7698,'1981-02-20',1600,300,30);
INSERT INTO emp values(7521,'郑智楷','售货员',7698,'1981-02-22',1250,500,30);
INSERT INTO emp values(7566,'黎忠灼','经理',7839,'1981-04-02',2975,NULL,20);
INSERT INTO emp values(7654,'赵洪滨','售货员',7698,'1981-09-28',1250,1400,30);
INSERT INTO emp values(7698,'刘军','经理',7839,'1981-05-01',2850,NULL,30);
INSERT INTO emp values(7782,'李任拳','经理',7839,'1981-06-09',2450,NULL,10);
INSERT INTO emp values(7788,'朱国建','分析师',7566,'1987-04-19',3000,NULL,20);
INSERT INTO emp values(7839,'观景风','董事长',NULL,'1981-11-17',5000,NULL,10);
INSERT INTO emp values(7844,'黄晓萍','售货员',7698,'1981-09-08',1500,0,30);
INSERT INTO emp values(7876,'陈韩维','店员',7788,'1987-05-23',1100,NULL,20);
INSERT INTO emp values(7900,'李俊洲','店员',7698,'1981-12-03',950,NULL,30);
INSERT INTO emp values(7902,'李静','分析师',7566,'1981-12-03',3000,NULL,20);
INSERT INTO emp values(7934,'车韦霖','店员',7782,'1982-01-23',1300,NULL,10);
```

部门表:dept

字段名称	字段类型	说明
deptno	int	部分编号
dname	varchar(50)	部分名称
loc	varchar(50)	部门所在地

部门编码
部门名称
部门所在地

```sql
CREATE TABLE dept(
    deptno      INT,
    dname       varchar(14),
    loc     varchar(13)
);
INSERT INTO dept values(10, '财务部', '北京');
INSERT INTO dept values(20, '研发部', '广州');
INSERT INTO dept values(30, '销售部', '上海');
INSERT INTO dept values(40, '操作间', '深圳');
```

1. 基础查询

1.1 查询所有列

```
SELECT * FROM stu;
```

	sid	sname	age
1	S_1001	liuYi	35
2	S_1002	chenEr	15
3	S_1003	zhangSan	95
4	S_1004	liSi	65
5	S_1005	wangWu	55
6	S_1006	zhaoLiu	75
7	S_1007	sunQi	25
8	S_1008	zhouBa	45
9	S_1009	wuJiu	85
10	S_1010	zhengShi	5
11	S_1011	xxx	*<null>*

1.2 查询指定列

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

	sid	sname	age
1	S_1001	liuYi	35
2	S_1002	chenEr	15
3	S_1003	zhangSan	95
4	S_1004	liSi	65
5	S_1005	wangWu	55
6	S_1006	zhaoLiu	75
7	S_1007	sunQi	25
8	S_1008	zhouBa	45
9	S_1009	wuJiu	85
10	S_1010	zhengShi	5
11	S_1011	xxx	*<null>*

2. 条件查询

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； IS NOT NULL
- AND;

- OR;
- NOT;

2.2 查询性别为女,并且年龄为50的学生信息

```
1
2
```

	sid	sname	age	gender
1	S_1002	chenEr	15	female
2	S_1008	zhouBa	45	female
3	S_1010	zhengShi	5	female

2.3 查询学号为S_1001，或者姓名为liSi的记录

```
1
2
```

	sid	sname	age	gender
1	S_1001	liuYi	35	male
2	S_1004	liSi	65	female

2.4 查询学号为S_1001，S_1002，S_1003的记录

```
1
2
```

	sid	sname	age	gender
1	S_1001	liuYi	35	male
2	S_1002	chenEr	15	female
3	S_1003	zhangSan	95	male

2.5 查询学号不是S_1001，S_1002，S_1003的记录

```
1
2
```

	sid	sname	age	gender
1	S_1004	liSi	65	female
2	S_1005	wangWu	55	male
3	S_1006	zhaoLiu	75	female
4	S_1007	sunQi	25	male
5	S_1008	zhouBa	45	female
6	S_1009	wuJiu	85	male
7	S_1010	zhengShi	5	female
8	S_1011	xxx	*<null>*	*<null>*

2.6 查询年龄为null的记录

```
1
```

2.7 查询年龄在20-40之间学生的记录

```
1
2
```

或者

```
1
2
```

	sid	sname	age	gender
1	S_1001	liuYi	35	male
2	S_1007	sunQi	25	male

2.8 查询性别非男的学生记录

```
1
2
```

或者

```
1
```

	sid	sname	age	gender
1	S_1002	chenEr	15	female
2	S_1004	liSi	65	female
3	S_1006	zhaoLiu	75	female
4	S_1008	zhouBa	45	female
5	S_1010	zhengShi	5	female

2.9 查询姓名不为null的学生记录

```
SELECT *
FROM stu
WHERE sname IS NOT NULL;
```

3 模糊查询

前面介绍的所有操作符都是针对已知值进行过滤的,不管是匹配一个还是多个值,测试大于还是小于已知值,或者检查摸个范围的值,共同点是过滤中使用的值都是已知的.但是,这种过滤方法并不是任何时候都好用,例如当想查询中包含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
在使用like关键字时,通常和通配符配合使用

```
通配符: 用来匹配一部分的特殊字符
    _ : 匹配任意一个字符
    % : 任意0~n个字符
```

3.1 查询姓名由5个字母构成的学生记录

```

```

3.3 查询姓名以“z”开头的学生记录

```

```

3.4 查询姓名中第二个字母是i的学生记录

```
```

3.5 查询姓名中包含“a”字母的学生记录

```

```

4. 字段控制查询

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;
```

	sal
1	800.00
2	1600.00
3	1250.00
4	2975.00
5	2850.00
6	2450.00
7	3000.00
8	5000.00
9	1500.00
10	1100.00
11	950.00
12	1300.00

数据是没有重复的

4.2 查看雇员的月薪和佣金之和

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;
```

而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:

empno	ename	job	mgr	hiredate	sal	comm	deptno	sal+comm
7369	梁芷瑞	店员	7902	1980-12-17	800	(Null)	20	(Null)
7499	杨鑫祥	售货员	7698	1981-02-20	1600	300	30	1900
7521	陈泽名	售货员	7698	1981-02-22	1250	500	30	1750
7566	吴世杰	经理	7839	1981-04-02	2975	(Null)	20	(Null)
7654	李晓辰	售货员	7698	1981-09-28	1250	1400	30	2650
7698	程晨强	经理	7839	1981-05-01	2850	(Null)	30	(Null)
7782	张泳佳	经理	7839	1981-06-09	2450	(Null)	10	(Null)
7788	彭钰盈	分析师	7566	1987-04-19	3000	(Null)	20	(Null)
7839	刘添文	董事长	(Null)	1981-11-17	5000	(Null)	10	(Null)
7844	林佳奇	售货员	7698	1981-09-08	1500	0	30	1500
7876	黎铭辉	店员	7788	1987-05-23	1100	(Null)	20	(Null)
7900	许旭濠	店员	7698	1981-12-03	950	(Null)	30	(Null)
7902	廖佳林	分析师	7566	1981-12-03	3000	(Null)	20	(Null)
7934	张辉	店员	7782	1982-01-23	1300	(Null)	10	(Null)

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

empno	ename	job	mgr	hiredate	sal	comm	deptno	sal+IFNULL(comm,0)
7369	梁芷瑞	店员	7902	1980-12-17	800	(Null)	20	800
7499	杨鑫祥	售货员	7698	1981-02-20	1600	300	30	1900
7521	陈泽名	售货员	7698	1981-02-22	1250	500	30	1750
7566	吴世杰	经理	7839	1981-04-02	2975	(Null)	20	2975
7654	李晓辰	售货员	7698	1981-09-28	1250	1400	30	2650
7698	程晨强	经理	7839	1981-05-01	2850	(Null)	30	2850
7782	张泳佳	经理	7839	1981-06-09	2450	(Null)	10	2450
7788	彭钰盈	分析师	7566	1987-04-19	3000	(Null)	20	3000
7839	刘添文	董事长	(Null)	1981-11-17	5000	(Null)	10	5000
7844	林佳奇	售货员	7698	1981-09-08	1500	0	30	1500
7876	黎铭辉	店员	7788	1987-05-23	1100	(Null)	20	1100
7900	许旭濠	店员	7698	1981-12-03	950	(Null)	30	950
7902	廖佳林	分析师	7566	1981-12-03	3000	(Null)	20	3000
7934	张辉	店员	7782	1982-01-23	1300	(Null)	10	1300

4.3 给列添加别名

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;
```

empno	ename	job	mgr	hiredate	sal	comm	deptno	total
7369	梁芷瑞	店员	7902	1980-12-17	800	(Null)	20	800
7499	杨鑫祥	售货员	7698	1981-02-20	1600	300	30	1900
7521	陈泽名	售货员	7698	1981-02-22	1250	500	30	1750
7566	吴世杰	经理	7839	1981-04-02	2975	(Null)	20	2975
7654	李晓辰	售货员	7698	1981-09-28	1250	1400	30	2650
7698	程晨强	经理	7839	1981-05-01	2850	(Null)	30	2850
7782	张泳佳	经理	7839	1981-06-09	2450	(Null)	10	2450
7788	彭钰盈	分析师	7566	1987-04-19	3000	(Null)	20	3000
7839	刘添文	董事长	(Null)	1981-11-17	5000	(Null)	10	5000
7844	林佳奇	售货员	7698	1981-09-08	1500	0	30	1500
7876	黎铭辉	店员	7788	1987-05-23	1100	(Null)	20	1100
7900	许旭濠	店员	7698	1981-12-03	950	(Null)	30	950
7902	廖佳林	分析师	7566	1981-12-03	3000	(Null)	20	3000
7934	张辉	店员	7782	1982-01-23	1300	(Null)	10	1300

给列起别名时，是可以省略AS关键字的：

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

5. 排序

排序使用 order by 列名 asc/desc 作为语法
默认是asc(升序) 可以指定 desc 降序

5.1 查询所有学生记录，按年龄升序排序

```
SELECT *
FROM stu
ORDER BY age ASC;
```

或者

```
SELECT *
FROM stu
ORDER BY age;
```

	sid	sname	age	gender
1	S_1011	xxx	<null>	<null>
2	S_1010	zhengShi	5	female
3	S_1002	chenEr	15	female
4	S_1007	sunQi	25	male
5	S_1001	liuYi	35	male
6	S_1008	zhouBa	45	female
7	S_1005	wangWu	55	male
8	S_1004	liSi	65	female
9	S_1006	zhaoLiu	75	female
10	S_1009	wuJiu	85	male
11	S_1003	zhangSan	95	male

5.2 查询所有学生记录，按年龄降序排序

```
SELECT *
FROM stu
ORDER BY age DESC;
```

	sid	sname	age	gender
1	S_1003	zhangSan	95	male
2	S_1009	wuJiu	85	male
3	S_1006	zhaoLiu	75	female
4	S_1004	liSi	65	female
5	S_1005	wangWu	55	male
6	S_1008	zhouBa	45	female
7	S_1001	liuYi	35	male
8	S_1007	sunQi	25	male
9	S_1002	chenEr	15	female
10	S_1010	zhengShi	5	female
11	S_1011	xxx	<null>	<null>

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;
```

empno	ename	job	mgr	hiredate	sal	comm	deptno
7839	刘添文	董事长	(Null)	1981-11-17	5000	(Null)	10
7788	彭钰盈	分析师	7566	1987-04-19	3000	(Null)	20
7902	廖佳林	分析师	7566	1981-12-03	3000	(Null)	20
7566	吴世杰	经理	7839	1981-04-02	2975	(Null)	20
7698	程晨强	经理	7839	1981-05-01	2850	(Null)	30
7782	张泳佳	经理	7839	1981-06-09	2450	(Null)	10
7499	杨鑫祥	售货员	7698	1981-02-20	1600	300	30
7844	林佳奇	售货员	7698	1981-09-08	1500	0	30
7934	张辉	店员	7782	1982-01-23	1300	(Null)	10
7521	陈泽名	售货员	7698	1981-02-22	1250	500	30
7654	李晓辰	售货员	7698	1981-09-28	1250	1400	30
7876	黎铭辉	店员	7788	1987-05-23	1100	(Null)	20
7900	许旭濠	店员	7698	1981-12-03	950	(Null)	30
7369	梁芷瑞	店员	7902	1980-12-17	800	(Null)	20

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；

6.1 COUNT

当需要纵向统计时使用COUNT(),COUNT小括号中可以放入指定列名,和* 如果是* 则代表查询的是结果集的行数,如果是列名,则是指定列的行数

查询emp表中记录数

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

	cnt
1	14

查询emp表中有佣金的人数

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

	cnt
1	4

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

```
1
2
```

	人数
1	5

统计月薪与佣金之和大于2500元的人数

```
1
2
```

	cnt
1	6

查询有佣金的人数，有领导的人数

```
1 SELECT COUNT(comm), COUNT(mgr) FROM emp;
```

	`COUNT(comm)`	`COUNT(mgr)`
1	4	13

6.2 SUM和AVG

当需要纵向求和时使用sum()函数。当需要求平均值时使用avg()函数

查询所有雇员的月薪和

```
1 SELECT SUM(sal) FROM emp;
```

	`SUM(sal)`
1	29025.00

查询所有雇员月薪和，以及所有雇员佣金和

```
1
2
```

	`SUM(sal)`	`SUM(comm)`
1	29025.00	2200.00

统计所有员工的平均工资

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

	`AVG(sal)`
1	2073.214286

6.3 MAX 和 MIN

MAX和MIN 是用来查询最大值和最小值的

查询员工的最高工资和最低工资:

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

	`MAX(sal)`	`MIN(sal)`
1	5000.00	800.00

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，这说明要使用部门来分组

```
注意:
凡是和聚合函数同时出现的列名，一定要写在group by 之后
分组时候是无法体现单个数据的
group by 一般会合聚合函数配合使用,单独使用的时候意义不大
```

7.1 分组查询

查询每个部门编号和每个部门的工资和:

```
SELECT deptno, SUM(sal)
FROM emp
GROUP BY deptno;
```

	deptno	`SUM(sal)`
1	10	8750.00
2	20	10875.00
3	30	9400.00

查询每个部门的部门编号以及每个部门的人数

```

```

查询每个部门的编号以及每个部门工资大于1500的人数:

```

```

7.2 HAVING字句

- 查询工资总和大于9000的部门编号以及工资总和:

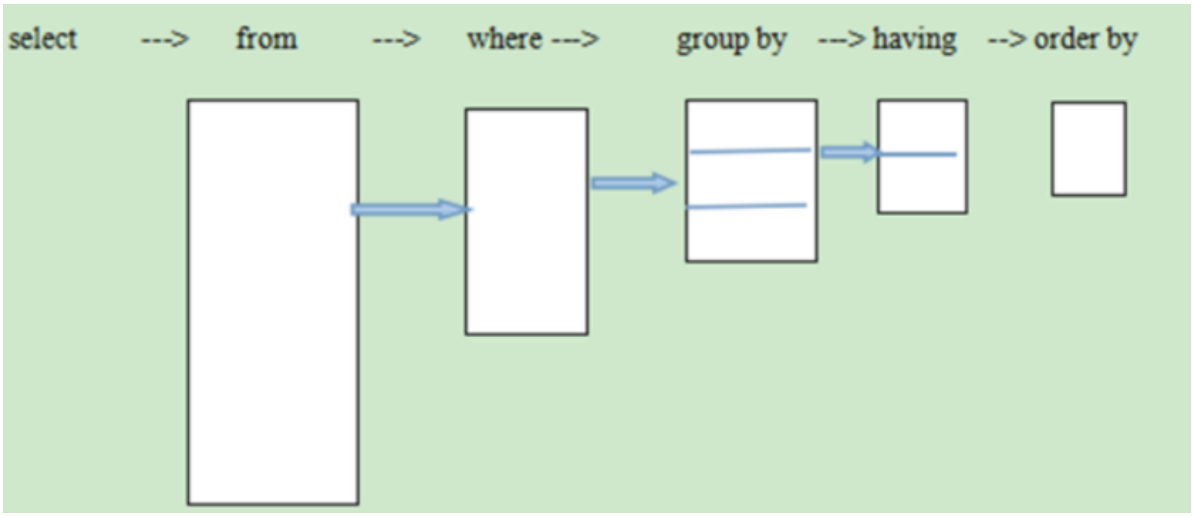
```
SELECT deptno, SUM(sal)
FROM emp
GROUP BY deptno
HAVING SUM(sal) > 9000;
```

	deptno	`SUM(sal)`
1	20	10875.00
2	30	9400.00

having和where的区别

```
having是在分组后对数据进行过滤,而where是在分组前对数据进行过滤
having后面可以使用聚合函数(统计函数),where后面不可以使用聚合函数
```

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



统计出各个部门的各个岗位中,平均工资>1000的信息

```

```

8. LIMIT

LIMIT用来限定查询结果的起始行，以及总行数。

- 查询5行记录，起始行从0开始

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

empno	ename	job	mgr	hiredate	sal	comm	deptno
7369	梁芷瑞	店员	7902	1980-12-17	800	(Null)	20
7499	杨鑫祥	售货员	7698	1981-02-20	1600	300	30
7521	陈泽名	售货员	7698	1981-02-22	1250	500	30
7566	吴世杰	经理	7839	1981-04-02	2975	(Null)	20
7654	李晓辰	售货员	7698	1981-09-28	1250	1400	30

查询语句的书写顺序:

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

查询语句的执行顺序:

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