

# 数据库

## 数据库管理系统(DBMS)

由三部分组成

- 数据库文件集合：主要是一系列的数据文件
- 数据库服务器：主要负责对数据文件及文件中的数据进行管理
- 数据库客户端：主要负责和服务端通信，向服务端传输数据或者从服务端获取数据

## 关系型数据库

关系型数据库使用表格来存储元素，使用**关系模型来组织数据的数据库**，关系模型指的就是**二维表格模型**

## 关系型数据库核心元素

- 字段：一列数据类型相同的数据
- 记录：一行记录某个事物的完整信息的数据
- 数据表：由若干字段和记录组成
- 数据库：由若干数据表组成
- 主键(primary key)：一行的标识

## 数据类型

- 整型

。

类型	存储(bytes)	最小值	最大值
TINYINT	1	-128	-127
SMALLINT	2	-32768	32767
MEDIUNINT	3	-8388608	8388607
INT	4	-2147483648	2147483647
BIGINT	8	省略	省略

- 浮点型

- decimal:定点数，decimal(5,2)代表共5位数字，其中2位是小数

- 字符串

- char: 定长字符串，创建时字段占用硬盘空间的大小已经确定了
- varchar: 变长字符串，占用空间为内容的长度+1个字节(字符串结束'\0')
- text: 无默认值

- 枚举类型(enum)

- 语法: `gender enum('男','女','妖')`

- 时间类型

- data: 年-月-日
- datetime: 年-月-日-时-分-秒
- timestamp: 年-月-日-时-分-秒

## 约束

### 保证数据的完整性

约束类型	约束说明
NOT NULL	非空约束（设置非空约束，该字段不能为空）
PRIMARY KEY	主键约束（唯一性，非空性）
UNIQUE KEY	唯一约束（唯一性，可以空，但只能有一个）
DEFAULT	默认约束（该数据的默认值）
FOREIGN KEY	外键约束(需要建立两表之间的关系)
AUTO_INCREMENT	为整数列自动生成唯一的递增值

```
create table student(  
    id int unsigned PRIMARY KEY AUTO_INCREMENT COMMENT '学生id',  
    name varchar(10) NOT NULL COMMENT '姓名',  
    age int unsigned COMMENT '年龄',  
    class int unsigned COMMENT '班级',  
    gender enum("男","女") COMMENT '性别'  
    status char(1) DEFAULT '1' COMMENT '状态'  
);COMMENT '花名册'
```

```
-- 插入数据
insert into student (name,age,class,gender) values('小明',18,3,"男") # stat
insert into student (name,age,class,gender) values('小林',17,2,"男") # id会
```

## 常见命令

==组成：库->表->数据== 每一行以分号结尾

命令	作用
mysql -uroot -p	连接数据库
exit/quit/ctrl+d	退出数据库
select version();	查看版本信息
select now();	查看时间

## 数据库基本操作命令

命令	作用
show databases;	查看所有数据库
select database();	查看当前使用的数据库
create database 数据库名 charset = utf8;	创建数据库
use 数据库名;	使用数据库
drop database 数据库名;	删除数据库

```
-- 展示所有数据库
show databases;
-- 删除数据库
select database();
-- 创建数据库
create database itcast charset=utf8;
-- 使用数据库
use itcast;
-- 删除数据库
drop database itcast
```

### 数据表基本操作命令

命令	作用
show tables;	查看当前数据库中的所有表
desc 表名;	查看表结构
show create tabel 表名;	查看表的创建语句
create table 表名;	创建数据表
comment '注释内容'	为表或表的某一列添加注释

```
-- 创建数据表
create table xxx(
    id int unsigned primary key auto_increment not null,
    name varchar(20),
    age int unsigned default 0,
    high decimal(5,2),
    gender enum("男","女"),
    cls_id int unsigned
);
```

### 数据表结构修改命令

命令	说明
alter table 表名 add 列名 类型;	添加字段
alter table 表名 change 原名 新名 类型及约束;	重命名字段
alter table 表名 modify 列名 类型及约束;	修改字段类型
alter table 表名 drop 列名;	删除字段
drop table 表名;	删除表

```
-- 修改类型
alter table student modify name varchar(20);
-- 增加字段
alter table student add gender enum("男","女");
-- 删除字段
alter table student drop gender
```

```
-- 添加注释
alter table student modify name COMMENT '姓名'
```

## 表数据操作命令

命令	说明
<code>insert into 表名 values(...);</code>	全列插入：值的顺序与表结构字段的顺序完全一一对应
<code>insert into 表名(列1,...)values();</code>	部分列插入：值的顺序与给出的列顺序对应
<code>insert into 表名 values(...),(...);</code>	一次性插入多行数据
<code>insert into 表名(列1,...) values(值1...)(值1...);</code>	部分列多行插入

```
-- 插入
insert into student values(0,"小林",18,166.66,"男",2);
insert into student (name,age,height,gender,class) values("小林",18,166.66,"男",2);
insert into student values(0,"小林",18,166.66,"男",2),(0,"小白",17,154.43,"男",1);
```

## 修改查询数据

命令	说明
<code>select *from 表名;</code>	查询所有列数据
<code>select 列1,列2... from 表名;</code>	查询指定列数据
<code>update 表名 set 列1 = 值1,列2 = 值2 ...where 条件;</code>	修改数据
<code>select distinct from 表名;</code>	去重查询

```
-- 修改数据
update students set gender = '女'; # 对列的数据进行修改
update students set gender = '男' where id = 2; # 对id为2的进行修改

-- 查询数据
select * from students;
select * from students where id = 2; # 查询特定列数据
```

```
select name,age from students; #
select name as '姓名',age as '年龄' from students; # 为列取别名进行查询
select age as '年龄',name as '姓名', from students; # 改变字段显示顺序
select distinct gender from students; # 去重查
```

## 删除数据

命令	说明
delete from 表名 where 条件;	删除数据

```
-- 物理删除
delete from students where id = 4;

-- 逻辑删除
alter table students add is_delete bit default 0; # 用一个字段表示这条信息能否删除
update students set is_delete = 1 where id = 4; # 更新信息
```

## where查询

- 比较运算符
  - =: 等于
  - >: 大于
  - <: 小于
  - != 或 <>: 不等于
- 逻辑运算符
  - and: 有多个条件时必须同时成立
  - or: 有多个条件时满足任意一个条件时成立
  - not: 表示取反

## 模糊查询

关键字: **like**

- %表示任意多个任意字符
- \_表示一个任意字符

```
-- 查询姓名中以"小"开始的名字
select * from students where name like '小%';
-- 查询姓名中有"小"的所有名字
select * from students where name like '%小%';
-- 查询有2个字的名字
select * from students where name like '__';
-- 查询至少有2个字的名字
select * from students where name like '__%';
```

## 范围查询

- between and: 表示在一个连续范围内查询, between A and B 表示匹配的范围空间时 [A,B]
- in: 表示在一个非连续范围内查询

```
-- 查询年龄为18和30的姓名
select name from students where age in (18,30);
-- 查询年龄不是18和30的信息
select * from students where age not in (18,30);
-- 查询年龄为18到30之间的信息
select * from students where age between 18 and 30;
-- 查询年龄不是18和30之间的信息
select * from students where age not between 18 and 30;
```

## 空值判断

- 判断为空: is null
- 判断为非空: is not null

```
-- 查询身高为空的信息
select * from students where height is null;
```

## order排序查询

```
select * from 表名 order by 列1 asc|desc[,列2asc|desc,...]
```

- 将行数据按照列1进行排序, 列1值相同按照列2, 以此类推
- asc从小到大进行排序, 升序

- desc从大到小进行排序，降序
- 默认为升序排序

```
-- 查询年龄在18到34岁之间的男性，身高升序
select * from students where age between 18 and 34 and gender = '男' order
-- 查询年龄在18到34岁之间的女性，身高降序，年龄降序
select * from students where age between 18 and 34 and gender = '女' order
```

## 聚合函数

命令	说明
count(字段)	计算总行数
max(字段)	求此字段最大值
min(字段)	求此字段最小值
sum(字段)	求此字段之和
avg(字段)	求此字段平均值

```
-- 查询男性有多少人
select count(*) from students where gender = '男';
-- 查询最大的年龄
select max(age) from students;
-- 查询身高最小值
select min(height) from students;
-- 对年龄求和
select sum(age) from students;
-- 对平均年龄保留两位小数
select round(avg(age),2) from students;
```

## group分组查询

- **group by分组:**将查询结果按照1个或多个字段进行分组，字段值相同的为一组
- **按照什么分组就只能查询什么**

命令	说明
group concat(字段)	分组时统计命令里面的字段



having 条件判断	根据条件在分组时进行过滤
with roll up	对分完组的数据进行汇总

```
-- 按照性别分组，查询所有性别
select gender from students group by gender
-- 计算每种性别的人数
select gender,count(*) from students group by gender;
-- 查询同种性别中的姓名
select group concat(name),gender group by gender;
-- 查询平均年龄超过30岁的性别，以及姓名
select group concat(name),gender group by gender having avg(age) > 30;
-- 对分完组的数据汇总
select gender,count(*) from students group by gender with rollup;
```

## limit限制查询

- 可以用limit限制取出记录的数量，limit要写在sql语句的最后
- 语法： `limit 起始记录, 记录数`
- 说明：
  - 起始记录是指从第几条记录开始取，第一条记录的下标是0
  - 记录数是指从起始记录开始向后依次取的记录数，也就是要查询的数量

```
-- 查询前5个数据
select * from students limit 5;
-- 每页显示2个，第1个页面
select * from students limit 0,2;
-- 每页显示2个，第2个页面
select * from students limit 2,2;
-- 每页显示2个，第4个页面，按年龄从小到大排序
select * from students order by age asc limit 6,2;
```

## 连接查询

将多张表连接成一个大的数据集进行汇总显示

### 内连接

查询的结果为两个表符合条件匹配到的数据

**语法:** `select 字段 from 表1 inner join 表2 on 表1.字段1 = 表2.字段2`

- **on 是连接条件**

```
-- 直接连接两个表
select * from students inner join classes;
-- 限制连接条件
select * from students inner join classes on students.cls_id = classes.id;
-- 给数据表起别名, 起完别名一定要使用
select s.name, c.name from students as s inner join classes as c on s.cls_id = c.cls_id
```

## 外连接

查询的结果为两个表匹配到的数据和左(右)表特有的数据

- 左连接: `主表 left join 从表 on 连接条件`
- 右连接: `主表 right join 从表 on 连接条件`

```
-- 使用右连接
select * from students right join classes on students.cls_id = classes.id;
-- 使用左连接
select * from students left join classes on students.cls_id = classes.id;
```

## 自连接

使用自连接查询只需要使用一个表就可以加快查询速度

如下图所示:

 image-20241007190605047

获取广东省有哪些市

1. 获取广东省的aid
2. 获取pid为1的所有atitle

==可以自己和自己连接查询, 即创建两个别名, 然后使用连接查询==

```
-- 自连接
```

```
select * from areas as city inner join areas as province on city.pid = pro
```

## 子查询

把一个查询的结果当作另一查询的条件

```
-- 查出平均身高
select avg(height) from students;
-- 查出高于平均身高的信息
select * from students where height > (select avg(height) from students);
```

## 实战

image-20241008135010101

```
-- 查询类型cate_name为'超极本'的商品名称name、价格price
select name,price from goods where cate_name = '超极本';

-- 显示商品的种类
## 分组方式
select brand_name from goods group by brand_name;
## 去重
select distinct brand_name from goods;

-- 求所有电脑产品的平均值avg，保留两位小数
select round(avg(price),2) from goods;

-- 显示每种类型cate_name的平均价格
select avg(price),cate_name from goods group by cate_name;

-- 查询每种类型的商品中最贵，最便宜，平均价，数量
select cate_name,max(price),min(price),avg(price),count(*) from goods group by cate_name;

-- 查询所有价格大于平均价格的商品，并且按价格降序排序
## 查询平均价格
select avg(price) from goods;
## 使用子查询
select * from goods where price > (select avg(price) from goods) order by price desc;

-- 查询每种类型中最贵的电脑信息
## 查找每种类型中最贵的价格
select cate_name,max(price) as max from goods group by cate_name;
```

```
## 关联查询
select * from goods inner join (select cate_name,max(price) as max from gc
```

## 外键

- 一个表的主键A在另一个表B中出现，就说**A是表B的一个外键**
- 可以防止无效信息的插入

命令	说明
alter table 表名 add foreign key(字段) references 表名(字段);	添加外键
alter table 表名 drop foreign key 外键名;	删除外键
show create table goods;	查看建立表的过程

```
-- 设置外键
alter table goods add foreign key(cate_id) references goods_cate(id);
-- 删除外键
## 查看外键名字
show create table goods;
## 取消外键约束
alter table 表名 drop foreign key goods_ibfk_1;

-- 创建数据表时设立外键
alter table goods{
    id int primary key auto_increment not null,
    name varchar(20),
    price decimal(5,2),
    cate_id int unsigned,
    brand_id int unsigned

    foreign key (cate_id) references goods_cate(id),
    foreign key (brand_id) references goods_brand(id)
};
```

## 视图

- 好处：方便查询操作，减少复杂的sql语句
- 把复杂sql语句功能封装起来的一个虚表
- 视图存储的数据发生改变，视图也会跟着改变

命令	说明
create view 视图名称 as select 语句;	定义视图
show tables;	查看视图
select * from v_goods_info;	使用视图
drop view 视图名称;	删除视图

```
-- 查出学生的id, 姓名, 年龄, 性别和学生的班级
select s.id,s.name,s.age,s.gender,c.name as cls_name from students as s inner join classes as c on s.class_id=c.id;

-- 创建上述结果的视图
create view v_students as select s.id,s.name,s.age,s.gender,c.name as cls_name from students as s inner join classes as c on s.class_id=c.id;

-- 使用视图
select * from v_students;

-- 删除视图
drop view v_students;
```

## 事务

- 指作为一个基本工作单元执行的一系列sql语言的操作，要么完全执行，要么完全不执行
- 四大特性ACID：
  - 原子性
  - 一致性
  - 隔离性
  - 持续性

命令	说明
begin;或start transaction;	开启事务
commit;	提交事务
rollback;	回滚事务

```
-- 开启事务
begin;
update students set age = 100 where id = 1;
-- 提交事务
commit;
```

## 索引

==创建索引可以大大加快数据查询的时间==

命令	说明
show index from 表名;	查看表中已有的索引
alter table 表名 add index 索引名 (字段名);	创建索引
drop index 索引名称 on 表名;	删除索引

```
-- 验证索引性能
set profiling = 1 ;
## 未创建索引查找数据的时间
select * from test_index where title = 'ha-99999';
show profiles;
## 创建索引
alter table test_index add index(title);
## 创建索引查找数据的时间
select * from test_index where title = 'ha-99999';
show profiles;
```

## 范式

- 不同的规范要求被称为不同的范式，**越高的范式数据冗余越小**
- 数据冗余指的是数据之间的重复

### 1. 一范式

- 强调的是字段的原子性，一个字段不能再分为其他的字段

### 2. 二范式(满足一范式)

- 必须要有一个主键
- 非主键字段必须完全依赖于主键，不能只依赖于主键的一部分（即与主键有非常强的联系）

- 允许传递依赖

### 3. 三范式(满足二范式)

- 不允许传递依赖，非主键字段必须直接依赖于主键
- 不能存在：非主键字段A依赖于非主键字段B，非主键字段B依赖于主键的情况

## SQL注入

用户提交带有恶意的数据与SQL语句进行字符串方式拼接，从而影响了SQL语句的语义，产生数据泄露

```
from pymysql import Connection
## 获取连接对象
conn = Connection(host='localhost',port=3306,user='root',password='zxb0506')
print(conn.get_server_info())
## 获取游标对象
cursor = conn.cursor()
## 选择数据库
conn.select_db('world')
## 用游标对象执行sql语句
find_name = input() # or 1 or
sql = "select * from students where name = '%s'" % find_name
cursor.execute(sql)
## 获取数据
content = cursor.fetchall() # 会获取全部的数据
```

==安全的方式：==

- 构造参数列表
- 执行sql语句时传入参数

```
params = [find_name]
sql = 'select * from students where name = %s'
cursor.execute(sql,params)
```

## SQL

可以用于几乎所有的关系型数据库

SQL语言分类：

- 数据定义：DDL
  - 库的创建删除，表的创建删除等
- 数据操纵：DML
  - 新增数据，删除数据，修改数据等
- 数据控制：DCL
  - 新增用户，删除用户，密码修改，权限管理等
- 数据查询：DQL
  - 基于需求查询和计算数据

特征：

- 大小写不敏感
- 可多行写，以分号结束
- 支持注释：
  - 单行注释：--注释内容(--后面一定要有一个空格)
  - 单行注释：#注释内容
  - 多行注释：/\* 注释内容 \*/

#### DDL（数据定义）

- 库管理：
  - 查看数据库： `show databases ;`
  - 使用数据库： `use 数据库名称 ;`
  - 创建数据库： `create database 数据库名称 [CHARSET UTF8] ;`
  - 删除数据库： `drop database 数据库名称 ;`
  - 查看当前使用的数据库： `select databse() ;`
- 表管理：
  - 查看表： `show tables ;`
  - 删除表： `drop table 表名称 ;`或 `drop table if exists 表名称 ;`
  - 创建表： `create table 表名称( 列名称 列类型, 列名称 列类型, ..... );`
    - 列类型有int, float, data(日期类型), timestamp(时间戳类型), varchar(长度): 文本

#### DML（数据操纵）

- 插入数据： `insert into 表[(列1, 列2, ..., 列N)] values(值1, 值2, .....值N)[, (值1, 值2, .....值N), (值1, 值2, .....值N), ..., (值1, 值2, .....值N)]`
- 数据删除： `delete from 表名称 [where 条件判断]` (和py中的差不多)，不加where条件整张表都会被删除



- 数据更新: `update 表名 set 列 = 值 [where 条件判断]` , 不加where条件也可以

## DQL (数据查询)

- 基础查询: `select 字段列表/* (查询所有列) from 表 where 条件判断`
- 分组聚合: `select 字段列表/聚合函数(要求的列) from 表 where 条件判断 group by 列`
  - group by中出现谁, select后面才能跟谁
  - 聚合函数有:
    - sum: 求和
    - avg: 求平均值
    - min: 求最小值
    - max: 求最大值
    - count: 求数量
- 结果排序: 使用order by关键字排序, `select 字段列表/聚合函数(要求的列) from 表 where 条件判断 group by 列 order by 列 (ASC: 从小到大, DESC:从大到小)`
- 分页限制: 使用limit关键字对结果分页展示, `limit n[,m]` (从第n + 1条开始, 向后取m条)

## python执行SQL语句

- 语法和上面说的一样, 只不过要先创立连接对象和游标对象, 后续通过连接对象建立与数据库连接, 通过游标对象进行数据的增删查改
- 在进行使数据产生变化的操作时, 需要进行 `commit()` , 进行手动确认, 操作才会生效; 也可以在连接对象设置autocommit参数, 令其为true

### 使用步骤

1. 导入pymysql包
2. 创建连接对象
3. 获取游标对象
4. pymysql完成增删改查操作
5. 关闭游标和链接

```
from pymysql import Connection
## 获取连接对象
conn = Connection(host='localhost', port=3306, user='root', password='zxb0508')
print(conn.get_server_info())
## 获取游标对象
cursor = conn.cursor()
```

```

## 选择数据库
conn.select_db('world')
## 用游标对象执行sql语句
sql = 'select * from students'
cursor.execute(sql)
## 获取一条数据
content = cursor.fetchone()
## 获取所有数据
content = cursor.fetchall()
print(content)
## 关闭连接
cursor.close()
conn.close()

```

## 增删改操作

==凡是涉及增删改都需要用到commit操作==

```

from pymysql import Connection
## 获取连接对象
conn = Connection(host='localhost',port=3306,user='root',password='zxb0508')
## 获取游标对象
cursor = conn.cursor()
## 选择数据库
conn.select_db('world')
## 插入数据
sql = "insert into students(name) values('老王')"
cursor.execute(sql)
## 查询数据
sql = 'select * from students;'
cursor.execute(sql)
content = cursor.fetchall()
for i in content:
    print(i)
## 提交操作
conn.commit()
## 关闭连接
cursor.close()
conn.close()

```