数据库

数据库概述

- 数据库：存储数据的仓库，数据是有组织的进行存储（DB）
- 数据库管理系统：操纵和管理数据库的大型软件（DBMS）
- SQL：操作关系型数据库的编程语言，定义了一套操作关系型数据库统一标准（SQL）
    - MySQL中小型数据库，开源
    - Oracle 收费数据库
    - Microsoft SQL Server
- 关系型数据库：建立在关系模型基础上，由多张相互连接的二维表组成的数据库
    - 特点：使用表储存数据，格式统一
    - 使用SQL语言操作，标准统一
- 数据模型：客户端通过DBMS操作数据库，创建二维表
- 客户端连接

SQL

SQL通用语法

- 通用语法
    - SQL语句可以单行或多行书写，以分号结尾
    - SQL语句可以使用空格/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，关键字建议使用大写
    - 注释
        - 单行注释：--注释内容/ #注释内容
        - 多行注释：/*注释内容*/

SQL分类

- DDL(Data Definition Language):数据定义语言，用来定义数据库对象
- DML（Data Manipulation Language）：数据操作语言，用来对数据库表中的数据进行增删改
- DQL（Data Query Language）：数据查询语言，用来查询数据库中表的记录
- DCL（Data Control Language）：数据控制语言，用来创建数据库用户，控制数据库的访问权限

DDL(数据定义语言)

- DDL数据库操作
    - 查询
        - 查询所有数据库：SHOW DATABASES;
        - 查询当前数据库:SELECT DATABASE();
    - 创建

```
CREAT DATABASE[IF NOT EXISTS]数据库名[DEFAULT CHARSET 字符集][COLLATE
排序规则];
```

- 删除

```
DROP DATABASE[IF EXISTS]数据库名;
```

- 使用/切换到某一数据库

```
USE 数据库名
```

- DDL表操作
  - 查询当前数据库所有表

```
SHOW TABLES;
```

  - 查询表结构/查询表中有哪些字段

```
DESC 表名;
```

  - 查询指定表的建表语句

```
SHOW CREATE TABLE 表名;
```

  - DDL表操作-创建

```
CREATE TABLE 表名(
    字段1 字段1类型[字段1注释],
    字段2 字段2类型[字段2注释],
    ...

)[表注释];
```

    最后一句没有逗号

  - DDL表操作-数据类型
    - 分为数值类型/字符串类型/日期类型
    - SQL 数据类型 (w3school.com.cn)
    - ```
      age TINY UNSIGNED,
      score DOUBLE(4, 1)[4为整体长度，1表示小数位数],
      ```
    - char类型性能高/varchar类型性能较差，因为需要根据输入计算数据位数

```
# 根据需求创建表
create table temp(
    id int comment '编号',
    worknum varchar(10) comment '工号',
    name varchar(10) comment '姓名',
    gender char(1) comment '性别',
    age tinyint unsigned comment '年龄',
    idcard char(18) comment '身份证号',
    entrydate date comment '入职时间'
) comment '员工表';
```
```

- DDL表操作-修改
  - 添加字段

```
ALTER TABLE 表名 ADD 字段名 类型(长度)[comment 注释][约束];
```

  - 修改数据类型

```
ALTER TABLE 表名 MODIFY 字段名 新数据类型(长度);
```

  - 修改数据类型和字段类型

```
ALTER TABLE 表名 CHANGE 旧字段名 新字段名 类型(长度)[注释][约束];
```

  - 删除字段

```
ALTER TABLE 表名 DROP 字段名;
```

  - 修改表名

```
ALTER TABLE 表名 RENAME TO 新表名;
```

  - 删除表

```
DROP TABLE [IF EXIST] 表名;
```

  - 删除指定表并重新创建该表(格式化)

```
TRUNCATE TABLE 表名;
```

DML(数据操作语言)

- DML-添加数据
  - 给指定字段添加数据

```
INSERT INTO 表名(字段1, 字段2,...) VALUES(值1, 值2, ...);
```

  - 给全部字段添加数据

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

  - 批量添加数据

```
INSERT INTO 表名 (字段1, 字段2, ...) VALUES(值1, 值2, ...),(值1, 值2, ...);
INSERT INTO 表名 VALUES (值1, 值2, ...)(值1, 值2, ...);
```

- DML-修改数据

```
UPDATE 表名 SET 字段1=值1,字段2=值2,...[WHERE 条件];
update temp set name = 'happy', gender = '4' where id = 1;
```

修改语句的条件可以有，也可以没有，如果没有条件，则会修改整张表的所有数据

- DML-删除数据

```
DELETE FROM 表名 [where 条件];
```

DQL(数据查询语言)

- 在正常业务系统中，查询频次比增删改的频次高
- 编写顺序

```
SELECT
    字段列表
FROM
    表名列表
WHERE
    条件列表
GROUP BY
    分组字段列表
HAVING
    分组后条件列表
ORDER BY
    排序字段列表
LIMIT
    分页参数
```

- 基本查询
  - 查询多个字段

```
SELECT 字段1,字段2,字段3,...FROM 表名;
SELECT * FROM 表名;--尽量不使用该语句
```

  - 设置别名

```
SELECT 字段1[AS 别名1],字段2[AS 别名2]...FROM 表名;
```

  - 去除重复记录

```
SELECT DISTINCT 字段列表 FROM 表名;
/*select distinct gender '男' from temp;*/
```

- 条件查询

```
SELECT 字段列表 FROM 表名 WHERE 条件列表
```

2. 条件

比较运算符	功能
>	大于
>=	大于等于
<	小于
<=	小于等于
=	等于
<> 或 !=	不等于
BETWEEN ... AND ...	在某个范围之内(含最小、最大值)
IN(...)	在in之后的列表中的值，多选一
LIKE 占位符	模糊匹配(_匹配单个字符, %匹配任意个字符)
IS NULL	是NULL

逻辑运算符	功能
AND 或 &&	并且(多个条件同时成立)
OR 或 ||	或者(多个条件任意一个成立)
NOT 或 ！	非，不是

```sql
select name from temp;
select name as '姓名' from temp;
select distinct gender '男' from temp;
select * from temp where age = 23;
select * from temp where age > 20;
select * from temp where age is not null ;
select * from temp where age <> 23;#不等于
select * from temp where age > 23 and age < 50;
select * from temp where gender = '女' and age < 50;
select * from temp where name like '__';#查询姓名为两个字的员工信息
select * from temp where name like '%g';#查询姓名最后一个字符是g的员工信息，注意_和%的使用
```

- 聚合函数-将一列数据作为一个整体进行纵向计算
  - 常见聚合函数,NULL值不参与聚合运算

函数	function
count	统计数量
max	最大值
min	最小值
avg	平均值
sum	求和

  - 语法

```sql
SELECT 聚合函数(字段列表) FROM 表名;
```

```sql
select count(id) from temp;
select avg(age) from temp;
select max(age) from temp;
select avg(age) from temp where age > 30;
```

- 分组查询

```sql
SELECT 字段列表 FROM 表名[WHERE 条件]GROUP BY 分组字段名[HAVING 分组后过滤条件]
select name, age from temp where score > 60 and gender = '男' having name like '%i';
```

- WHERE和HAVING的区别
  - 执行时机不同：where是分组之前进行过滤，不满足where条件不参与分组，而having是分组之后对结果进行过滤
  - 判断条件不同：where不能对聚合函数进行判断，而having可以
- 执行顺序：where > 聚合函数 > having
- 分组之后查询的字段一般为聚合函数和分组字段，查询其他字段无任何意义

- 排序查询

```
SELECT 字段列表 FROM 表名 ORDER BY 字段1 排序方式, 字段2 排序方式;
```

  - 排序方式：
    - ASC: 升序
    - DESC：降序
  - ```
    select * from temp order by age desc, score asc;#先按照年龄降序排序，如果年龄相同，则按照成绩升序排序
    ```

- 分页查询
  - ```
    SELECT 字段列表 FROM 表名 LIMIT 起始索引,查询记录数;
    ```
  - 起始索引从0开始，起始索引=(查询页码-1)*每页显示记录数
  - 分页查询在不同的数据库中有不同的实现
  - 如果查询的是第一页数据，起始索引可以省略，直接简写为limit 10;
- DQL-执行顺序

DCL（数据控制语言）

- 用来管理数据库的用户，控制数据库的访问权限
- DCL-管理用户
  - 查询用户

    ```
    USE mydql
    SELECT * FROM user
    ```

    所有用户都是保存在mysql数据库中
  - 创建用户（创建后用户没有权限）

```
CREAT USER '用户名'@'主机名' IDENTIFIED BY '密码';


create user 'itcast'@'localhost' identified by '12345';
create user 'itcast'@'%' identified by '12345';#%代表任意主机
```

- 修改用户密码

```
ALTER USER '用户名'@'主机名' IDENTIFIED WITH mysql_native_password BY '新密码';
```

- 删除用户

```
DROP USER '用户名'@'主机名';
```

- DCL-权限控制

  MySQL中定义了很多种权限，但是常用的就以下几种：

权限	说明
ALL, ALL PRIVILEGES	所有权限
SELECT	查询数据
INSERT	插入数据
UPDATE	修改数据
DELETE	删除数据
ALTER	修改表
DROP	删除数据库/表/视图
CREATE	创建数据库/表

- 查询权限

```
SHOW GRANTS FOR '用户名'@'主机名'
```

- 授予权限

```
GRANT 权限列表 ON 数据库名.表名 TO '用户名'@'主机名';

grant all on itt.* to 'itcast'@'localhost';#授予所有权限
```

- 撤销权限

```
REVOKE 权限列表 ON 数据库名.表名 FROM '用户名'@'主机名';
```

函数

- MySQL中内置了很多函数

字符串函数

MySQL中内置了很多字符串函数，常用的几个如下：

函数	功能
CONCAT(S1,S2,...Sn)	字符串拼接，将S1，S2，... Sn拼接成一个字符串
LOWER(str)	将字符串str全部转为小写
UPPER(str)	将字符串str全部转为大写
LPAD(str,n,pad)	左填充，用字符串pad对str的左边进行填充，达到n个字符串长度
RPAD(str,n,pad)	右填充，用字符串pad对str的右边进行填充，达到n个字符串长度
TRIM(str)	去掉字符串头部和尾部的空格
SUBSTRING(str,start,len)	返回从字符串str从start位置起的len个长度的字符串

- 使用

```
SELECT 函数;
update temp set id = lpad(id, 3, '0');
```

数值函数

常见的数值函数如下：

函数	功能
CEIL(x)	向上取整
FLOOR(x)	向下取整
MOD(x,y)	返回x/y的模
RAND()	返回0~1内的随机数
ROUND(x,y)	求参数x的四舍五入的值，保留y位小数

- 通过数据库函数生成一个六位数的随机验证码

```
select lpad(round(rand()*1000000, 0), 6, '0');
```

日期函数

常见的日期函数如下：

函数	功能
CURDATE()	返回当前日期
CURTIME()	返回当前时间
NOW()	返回当前日期和时间
YEAR(date)	获取指定date的年份
MONTH(date)	获取指定date的月份
DAY(date)	获取指定date的日期
DATE_ADD(date, INTERVAL expr type)	返回一个日期/时间值加上一个时间间隔expr后的时间值
DATEDIFF(date1,date2)	返回起始时间date1 和 结束时间date2之间的天数

```
select date_add(now(), interval 70 day);#往后推7
```

流程控制函数

流程函数也是很常用的一类函数，可以在SQL语句中实现条件筛选，从而提高语句的效率。

函数	功能
IF(value , t, f)	如果value为true，则返回t，否则返回f
IFNULL(value1 , value2)	如果value1不为空，返回value1，否则返回value2
CASE WHEN [val1] THEN [res1] ... ELSE [default] END	如果val1为true，返回res1，... 否则返回default默认值
CASE [expr] WHEN [val1] THEN [res1] ... ELSE [default] END	如果expr的值等于val1，返回res1，... 否则返回default默认值

```
create table score(
    id int comment '编号',
    name varchar(15) comment '姓名',
    math int unsigned comment '数学',
    English int unsigned comment '英语',
    Chinese int unsigned comment '语文'
)comment '学生成绩表';
insert into score(id, name, math, English, Chinese) VALUES (1, 'zhangsan', 60, 79, 98),
                                                            (2, 'lisi', 78, 98, 50),
                                                            (3, 'wangwu', 53, 80, 87),
                                                            (4, 'joey', 40, 78, 72),
                                                            (5, 'ross', 87, 88, 83),
                                                            (6, 'chandler', 76, 40, 94);
select * from score;
select id, name,
       (case when math >= 85 then '优秀' when math < 85 and math >= 60 then '及格' else '不及格' end) as  '数学',
        (case when English >= 85 then '优秀' when English < 85 and English >= 60 then '及格' else '不及格' end) as '英语',
       (case when Chinese >= 85 then '优秀' when Chinese < 85 and Chinese >= 60 then '及格' else '不及格' end) as '语文'
from score;
```

约束

- 概念：约束是作用于表中字段上的规则，用于限制存储在表中的数据

- 目的：保证数据库中数据的正确，有效性和完整性

1. 概念：约束是作用于表中字段上的规则，用于限制存储在表中的数据。
2. 目的：保证数据库中数据的正确、有效性和完整性。
3. 分类：

约束	描述	关键字
非空约束	限制该字段的数据不能为null	NOT NULL
唯一约束	保证该字段的所有数据都是唯一、不重复的	UNIQUE
主键约束	主键是一行数据的唯一标识，要求非空且唯一	PRIMARY KEY
默认约束	保存数据时，如果未指定该字段的值，则采用默认值	DEFAULT
检查约束(8.0.16版本之后)	保证字段值满足某一个条件	CHECK
外键约束	用来让两张表的数据之间建立连接，保证数据的一致性和完整性	FOREIGN KEY

注意：约束是作用于表中字段上的，可以在创建表/修改表的时候添加约束。

- 对于一个字段来说可以添加多个约束
-

字段名	字段含义	字段类型	约束条件	约束关键字
id	ID唯一标识	int	主键，并且自动增长	PRIMARY KEY, AUTO_INCREMENT
name	姓名	varchar(10)	不为空，并且唯一	NOT NULL, UNIQUE
age	年龄	int	大于0，并且小于等于120	CHECK
status	状态	char(1)	如果没有指定该值，默认为1	DEFAULT
gender	性别	char(1)	无	

按照需求创建表

```sql
```sql
create table user(
    id int primary key  auto_increment comment '主键',
    name varchar(15) not null unique comment '姓名',
    age int check ( age > 0 and age < 120 ) comment '年龄',
    status char(1) default '1' comment '状态',
    gender char comment '性别'
)comment '用户表';
```
```

外键约束

- 概念：外键用来让两张表的数据之间建立连接，从而保证数据的一致性和完整性
- 语法
  - 添加外键

```sql
CREAT TABLE 表名(
    字段名 数据类型
    ...
    [constraint][外键名称]FOREIGN KEY(外键字段名)REFERENCES 主表(主表列名)
);


ALTER TABLE 表名 ADD CONSTRAINT 外键名称 FOREIGN KEY(外键字段名)
REFERENCES 主表(主表列名);
```

  - 删除外键

- 删除/更新行为

行为	说明
NO ACTION	当在父表中删除/更新对应记录时，首先检查该记录是否有对应外键，如果有则不允许删除/更新。(与 RESTRICT 一致)
RESTRICT	当在父表中删除/更新对应记录时，首先检查该记录是否有对应外键，如果有则不允许删除/更新。(与 NO ACTION 一致)
CASCADE	当在父表中删除/更新对应记录时，首先检查该记录是否有对应外键，如果有，则也删除/更新外键在子表中的记录。
SET NULL	当在父表中删除对应记录时，首先检查该记录是否有对应外键，如果有则设置子表中该外键值为null（这就要求该外键允许取null）。
SET DEFAULT	父表有变更时，子表将外键列设置成一个默认的值(Innodb不支持)

```
ALTER TABLE 表名 ADD CONSTRAINT 外键名称 FOREIGN KEY (外键字段) REFERENCES 主表名(主表字段名) ON UPDATE CASCADE ON DELETE CASCADE;
```

多表查询

多表关系

- 项目开发中，根据业务需求和业务模块之间的关系分析并设计表结构，由于业务之间的相互关联，所以各个表结构之间也存在各种联系，基本分为
  - 一对多
  - 多对多
  - 一对一
- 一对多
  - 员工与部门的关系。一个部门可以有多个员工
  - 实现：在多的一方建立外键，指向一的一方的主键
- 多对多
  - 学生与课程的关系；一个学生可以选择多个课程，一个课程也可以有多个学生
  - 实现：建立第三张中坚表，中间至少包含两个外键，分别关联两方主键
  - ```
    create table student(
        id int auto_increment primary key comment '主键ID',
        name varchar(15) comment '姓名',
        num varchar(15) comment '学号'
    )comment '学生表';
    insert into student(id, name, num) VALUES (null, 'Monica', '2222'),
                                               (null, 'Joey', '2013'),
                                               (null, 'Ross', '5432'),
                                               (null, 'Chandler',
    '6524');
    create table course(
        id int auto_increment primary key comment '主键ID',
        name varchar(15) comment '课程名称'
    )comment '课程表';
    insert into course(id, name) VALUES (null, 'English'),
                                        (null, 'MySql'),
                                        (null, 'Chinese');
    create table student_course(
        id int auto_increment comment '主键' primary key ,
        studentid int not null comment '学生ID',
        courseid int not null comment '课程ID',
        constraint fk_courseid foreign key (courseid) references
    course(id),
        constraint fk_studentid foreign key (studentid) references
    student(id)
    )comment '学生课程中间表';
    ```
    ```

```
insert into student_course values (null, 1, 1),
                                  (null, 1, 2),
                                  (null, 2, 2),
                                  (null, 2, 3),
                                  (null, 3, 3);
```

- 一对一
  - 一对一关系多用于单表拆分，将一张表的基础字段放在一张表中，其他详情字段放在另一张表中，以提升操作效率
  - 实现：在任意一方加入外键关联另一方的主键，并且设置外键为唯一的(UNIQUE)

多表查询概述

- 从多张表中查询数据
- 笛卡尔积：在数学中指集合A和集合B的所有组合情况；在多表查询时需要消除笛卡尔积

```
select * from student, course where student.id = course.id;
# 用where消除笛卡尔积
```

- 多表查询分类
  - 连接查询
    - 内连接

连接查询-内连接

内连接查询语法：

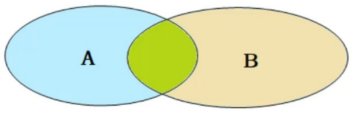
➢ 隐式内连接

```
SELECT 字段列表 FROM 表1 , 表2 WHERE 条件 ... ;
```

➢ 显式内连接

```
SELECT 字段列表 FROM 表1 [ INNER ] JOIN 表2 ON 连接条件 ... ;
```

内连接查询的是两张表交集的部分




```
select student.name, course.name from student, course where
student.id = course.id;#隐式内连接
select cour.name, stu.name from student stu inner join course
cour on stu.id = cour.id;# 显式内连接
```

```
select st.id, st.name,st.age, d.name from stuff st inner join
dept d on st.id = d.id where st.age<35;
```

    - 外连接
      - 左外连接：查询左表所有数据以及两张表交集部分的数据
      - 右外连接：查询右表所有数据以及两张表交集部分数据

```
select * from course as cour right outer join student sc on
cour.id = sc.id;
```

- 自连接：当前表与自身的连接查询，自连接必须使用表别名

```
SELECT 字段列表 FROM 表A 别名A JOIN 表A 别名B ON 条件....;
```

  - 自连接可以是内连接也可以是外连接

- 联合查询
  - 将多次查询的结果合并起来形成一个新的查询结果集
  -
```
SELECT 字段列表 FROM 表A
UNION[ALL] # all可以对查询结果去重
SELECT 字段列表 FROM 表B;
```

  - 对于联合查询的多张表的列数必须保持一致，字段类型也需要保持一致
- 子查询
  - SQL语句中嵌套SELECT语句，称为嵌套查询，也称子查询
  - 根据子查询结果分类
    - 标量子查询（查询结果为一个值）

```
select * from stuff where stuff.id = (select id from dept
where name='财务部');
```

```
select d.id, d.name, (select count(*) from stuff stu where
stu.id = d.id) as '人数' from dept d;
```

    - 列子查询（查询结果为一列或者是多行）
      - 列子查询

子查询返回的结果是一列（可以是多行），这种子查询称为**列子查询**。

常用的操作符：IN 、NOT IN 、 ANY 、SOME 、 ALL

操作符	描述
IN	在指定的集合范围之内，多选一
NOT IN	不在指定的集合范围之内
ANY	子查询返回列表中，有任意一个满足即可
SOME	与ANY等同，使用SOME的地方都可以使用ANY
ALL	子查询返回列表的所有值都必须满足

```
select * from stuff where id in (select id from dept where
name = '财务部');
```

```
select * from stuff where salary > all(select salary from
stuff where name = '财务部')# 查询比财务部所有人都高的员工工资信息
```

        - 行子查询（查询结果为一行）
            - ```
              select * from stuff where (name, salary) = ('Ross', 5000);
              ```
        - 表子查询（查询结果为多行多列）
            - ```
              select * from stuff where (id, salary) in (select id, salary from stuff where id = (SELECT ID FROM dept WHERE name = '财务部'));
              ```
    - 根据子查询位置分为
        - where之后
        - from之后
        - select之后

事务

- 事务是指一组操作的集合，是不可分割的工作单位，会把所有操作作为一个整体一起向系统提交或撤销操作请求，这些操作要么同时成功，要么同时失败
- MySql中事务默认是自动提交的
- 事务操作
    - 开启事务

      ```
      start transaction;#不用设置事务提交方式
      ```
    - 查看/设置事务提交方式

      ```
      SELECT @@autocommit;
      SET @@autocommit = 0;#设置为手动提交
      ```
    - 提交事务

      ```
      COMMIT;
      ```
    - 回滚事务

      ```
      ROLLBACK;
      ```
- ```
  set @@autocommit = 0;
  update account set money = money - 1000 where name = 'Ross';
  update account set money = money + 1000 where name = 'Monica';
  select * from account where name = 'Ross';
  commit ;
  rollback;
  ```
  ```

- ```
  start transaction ;
  update account set money = money - 1000 where name = 'Chandler';
  update account set money = money + 1000 where name = 'Joey';
  select * from account where name = 'Chandler';
  commit ;
  rollback ;
  ```

- 事务的四大特性（ACID）（面试）
  - 原子性：事务是不可分割的最小操作单元，要么全部成功，要么全部失败
  - 一致性：事务完成时，必须使所有的数据都保持一致状态
  - 隔离性：数据库系统提供的隔离机制，保证事务在不受外部并发操作影响的独立环境下运行
  - 持久性：事务一旦提交或者回滚，他对数据库中的数据的改变就是永久的
- 并发事务问题：A事务与B事务同时操作数据库
  - 脏读：一个事务读到另外一个事务还没有提交的数据
  - 不可重复读：一个事务先后读取同一条记录，但两次读取的数据不同
  - 幻读：一个事务按照条件查询数据时，没有对应的数据行，但在插入数据时又发现这行数据已经存在，好像出现了一个幻影
- 事务隔离级别：用来解决并发事务所引发的问题
隔离级别	脏读	不可重复读	幻读
Read uncommitted	√	√	√
Read committed	×	√	√
Repeatable Read(默认)	×	×	√
Serializable	×	×	×

  - 效率由上到下降低
  - 查看事务隔离级别

    ```
    SELECT @@TRANSACTION_ISOLATION
    ```

  - 设置事务隔离级别

    ```
    SET[SESSION/GLOBAL] TRANSACTION ISOLATION LEVEL {READ UNCOMMITED\READ
    COMMITED\REPEATABLE READ\SERIALIZABLE}
    ```

    - SESSION是针对当前客户端窗口有效，global是针对所有客户端的窗口有效
  - 事务隔离级别越高，数据越安全，性能越低，一般使用默认的事务级别不做修改
    ```