

数据库

数据库基本概念

数据库

DataBase(数据库), 存储数据的仓库。

之前我们项目中将用户信息写入一个文件, 最终用一个目录users保存所有用户信息。这个users目录就可以称为

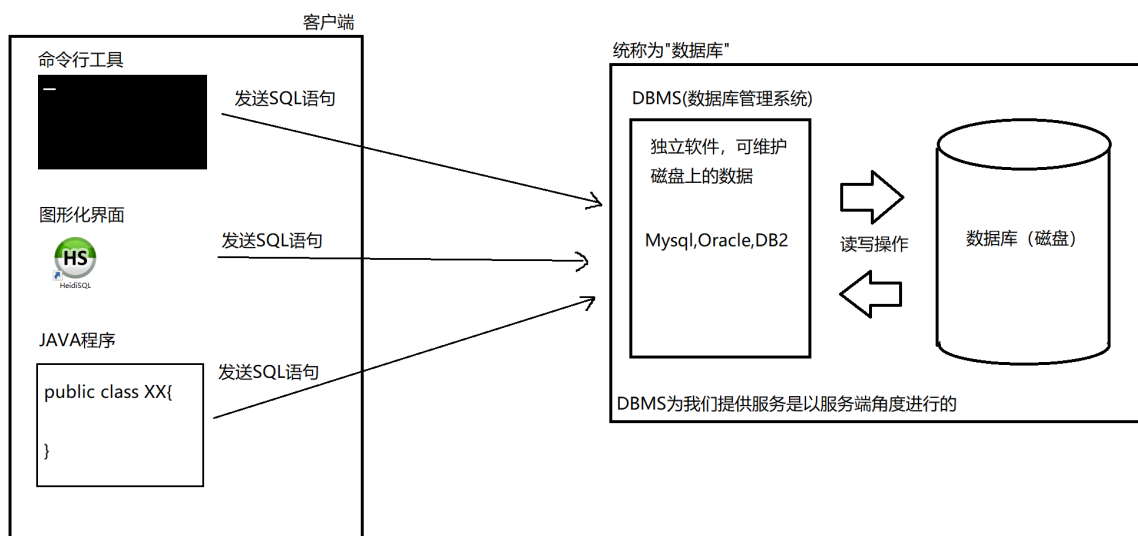
是一个数据库。利用文件系统进行操作, 经常已手动或半自动形式维护数据的操作, 缺点:低效。

数据库管理系统

DBMS(数据库管理系统)。可独立运行的软件, 用于维护数据的操作。

常见的DBMS有:

- mysql
- oracle
- db2
- sqlserver



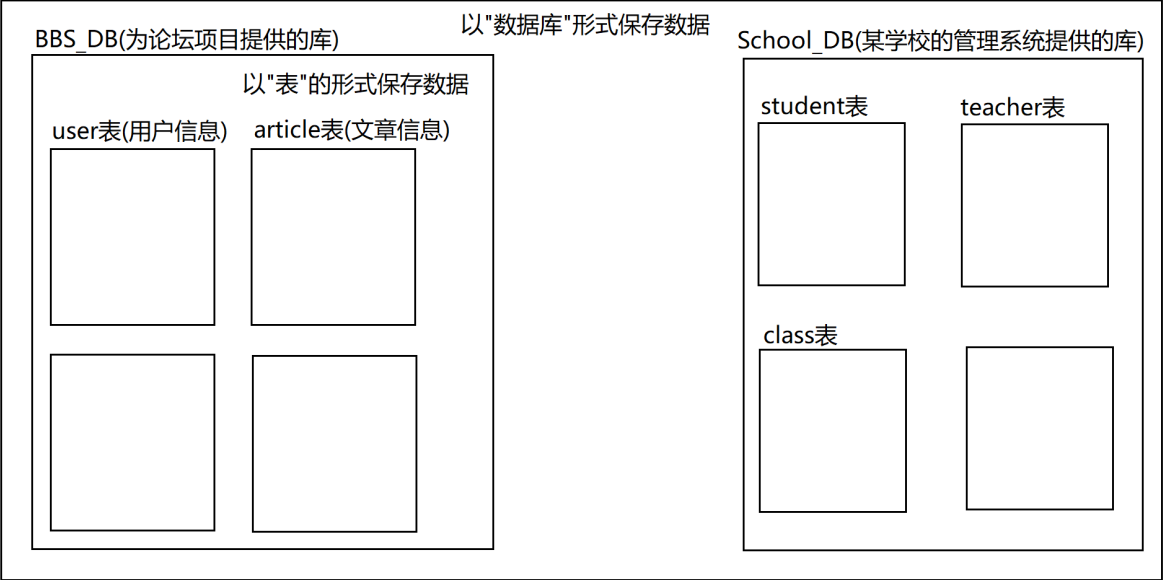
数据库与表的概念

以Mysql为例:

我们可以在Mysql中创建若干个数据库, 每个数据库用于一个项目。

每个数据库中又可以创建多个表, 表是用于保存一组数据的。

示意图:



数据表是由行和列构成

其中列被称为字段，就是一组数据中各部分信息。

其中行被称为记录，就是各部分信息组成的一条数据。

注:用面向对象的角度思考:

表相当于是一个java类。比如User类

字段相当于类中的属性。比如User类中有username,password,nickname,age四个属性

记录相当于类的一个实例。比如一个User实例就可以表示一个用户信息

记录 ➡

| username | password | nickname | age |
|----------|----------|----------|-----|
| 张三 | 123456 | 阿三 | 22 |
| 李四 | 556677 | 老四 | 33 |
| ... | ... | ... | ... |
| | | | |
| | | | |
| | | | |
| | | | |

↓ 字段

如何操作DBMS

所有的DBMS都支持通过SQL语句进行操作。我们向数据库发送特定的SQL语句来表达要进行某项操作。

SQL有标准：SQL92

所有的DBMS都支持SQL92标准。

注:Structured Query Language (SQL) 结构化查询语言

连接数据库的方式

1. 命令行形式
2. 第一方或第三方提供的图形化界面的客户端
3. 在集成开发环境中(IDEA,Eclipse)
4. JDBC(java 数据库连接), java程序中若需要使用数据库则这样连接(IDEA,Eclipse都采取这种方式)

SQL语句分类

- DDL 数据定义语言
CREATE,DROP,ALTER
对数据库对象进行操作的语言。数据库对象有:数据库, 表, 视图, 索引等。
- DML 数据操作语言
INSERT,UPDATE,DELETE
对表中的数据进行操作的语言。
- DQL 数据查询语言
SELECT
对表中的数据进行查询的语言。
- DCL 数据控制语言
DBA用于管理数据库的语言。
- TCL 事物控制语言
COMMIT,ROLLBACK
对DML数据操作保证具有原子性, 一致性。

DDL 数据定义语言

查看DBMS中已有的数据库

```
SHOW DATABASES
```

数据库相关操作

新建一个数据库

基本语法:

```
CREATE DATABASE 数据库名 [CHARSET=字符集名称]
```

例如:

```
新建一个名为mydb的数据库  
CREATE DATABASE mydb
```

创建数据库时可以指定字符集

```
CREATE DATABASE mydb1 CHARSET=UTF8  
CREATE DATABASE mydb2 CHARSET=GBK
```

查看数据库信息

```
SHOW CREATE DATABASE 数据库名
```

例:

```
SHOW CREATE DATABASE mydb1
```

删除数据库

```
DROP DATABASE 数据库名
```

例:

```
DROP DATABASE mydb1
```

使用一个数据库(切换一个数据库)

若希望保存数据，则数据必须保存在某张表上。而表必须保存在某个数据库上，因此后期为了对某个数据库的表进行操作，要先切换到该数据库上，才能进行操作。

```
USE 数据库名
```

例:切换到mydb数据库

```
USE mydb
```

练习:

1. 创建 mydb1和mydb2 数据库 字符集分别为utf8和gbk

```
CREATE DATABASE mydb1 CHARSET=utf8  
CREATE DATABASE mydb2 CHARSET=gbk
```
2. 查询所有数据库检查是否创建成功

```
SHOW DATABASES
```
3. 检查两个数据库的字符集是否正确

```
SHOW CREATE DATABASE mydb1  
SHOW CREATE DATABASE mydb2
```
4. 先使用mydb2 再使用 mydb1

```
USE mydb2  
USE mydb1
```
5. 删除两个数据库

```
DROP DATABASE mydb1  
DROP DATABASE mydb2
```

表相关操作

创建表

语法:

```
CREATE TABLE 表名(  
    字段名 类型,  
    字段名 类型,  
    ...  
)
```

例:

```
CREATE TABLE user(  
    id INT,  
    username VARCHAR(30),  
    password VARCHAR(30),  
    nickname VARCHAR(30),  
    age INT(3)  
)
```

INT在mysql中为整数类型。
VARCHAR在mysql中为字符串类型，长度为字节，
VARCHAR(30)则字符转换的字节最多30个，
若UTF-8编码则中文最多10个字(每个中文3字节)。
INT指定长度为位数。这里INT(3)为最多3位整数。

查看已创建的表的表结构

语法:

```
DESC 表名
```

例:

查看user表的表结构

```
DESC user
```

查看已创建表的详细信息

```
SHOW CREATE TABLE 表名
```

例如:

```
SHOW CREATE TABLE user
```

删除表

语法:

```
DROP TABLE 表名
```

例:

删除user表

```
DROP TABLE user
```

查看数据库中所有的表

```
SHOW TABLES
```

修改表

- 修改表名

语法:

```
RENAME TABLE 原表名 TO 新表名
```

例如:

将表user改名为userinfo

```
RENAME TABLE user TO userinfo
```

- 修改表结构

准备一张表测试：

```
CREATE TABLE hero(  
    name VARCHAR(30),  
    age INT(3)  
)
```

| Field | Type |
|-------|-------------|
| name | varchar(30) |
| age | int(3) |

○ 添加字段

- 向表末尾追加新的字段

```
ALTER TABLE 表名 ADD 字段名 类型
```

例:

```
ALTER TABLE hero ADD gender VARCHAR(10)
```

| Field | Type |
|--------|-------------|
| name | varchar(30) |
| age | int(3) |
| gender | varchar(10) |

- 将字段添加到表的第一个字段位置上

```
ALTER TABLE 表名 ADD 字段名 类型 FIRST
```

例:

在hero表第一个字段上添加id

```
ALTER TABLE hero ADD id INT FIRST
```

| Field | Type |
|--------|-------------|
| id | int(11) |
| name | varchar(30) |
| age | int(3) |
| gender | varchar(10) |

- 将字段插入到表中某个位置

将字段插入到表中某字段的后面

ALTER TABLE 表名 **ADD** 字段名 类型 **AFTER** 表中某字段

例:

将密码(pwd)字段插入到name和age之间

ALTER TABLE hero **ADD** pwd **VARCHAR**(30) **AFTER** name

| | Field | Type |
|---|--------|-------------|
| 1 | id | int(11) |
| 2 | name | varchar(30) |
| 3 | pwd | varchar(30) |
| 4 | age | int(3) |
| 5 | gender | varchar(10) |

○ 删除字段

ALTER TABLE 表名 **DROP** 字段名

例:

将pwd字段从hero表中删除

ALTER TABLE hero **DROP** pwd

○ 修改现有字段

ALTER TABLE 表名 **CHANGE** 原字段名 新字段名 类型

例:

- 将hero表中age字段的长度改为5

ALTER TABLE hero **CHANGE** age age **INT**(5)

| Field | Type |
|----------|-------------|
| 1 id | int(11) |
| 2 name | varchar(30) |
| 3 age | int(3) |
| 4 gender | varchar(10) |




| Field | Type |
|----------|-------------|
| 1 id | int(11) |
| 2 name | varchar(30) |
| 3 age | int(5) |
| 4 gender | varchar(10) |

- 将hero表中age字段的类型改为VARCHAR(10)

ALTER TABLE hero **CHANGE** age age **VHARCHAR**(10)


| Field | Type |
|----------|-------------|
| 1 id | int(11) |
| 2 name | varchar(30) |
| 3 age | int(5) |
| 4 gender | varchar(10) |



| Field | Type |
|----------|-------------|
| 1 id | int(11) |
| 2 name | varchar(30) |
| 3 age | varchar(10) |
| 4 gender | varchar(10) |

- 将hero表中gender字段名改为pwd
`ALTER TABLE hero CHANGE gender pwd VARCHAR(10)`

| Field | Type |
|----------|-------------|
| 1 id | int(11) |
| 2 name | varchar(30) |
| 3 age | varchar(10) |
| 4 gender | varchar(10) |



| Field | Type |
|--------|-------------|
| 1 id | int(11) |
| 2 name | varchar(30) |
| 3 age | varchar(10) |
| 4 pwd | varchar(10) |

- 注意事项:

修改表字段应当在表中没有数据时进行。如果表中已经存在数据，则修改字段可能不成功。

比如:

hero表中有10条记录，每条记录gender字段已经有值(每个人的性别)

若此时将表字段gender修改类型为INT。此时字符串转换int("男"怎么转成整数???)

比如:

hero表中pwd记录了每个人的密码。原长度为10。此时若修改字段长度为5，会导致原密码保存不下则修改失败。

练习:

1. 创建数据库mydb3 字符集gbk 并使用
`CREATE DATABASE mydb3 CHARSET=gbk`
`USE mydb3`
2. 创建t_hero英雄表，有名字和年龄字段
`CREATE TABLE t_hero(
name VARCHAR(30),
age INT(3)
)`
3. 修改表名为hero
`RENAME TABLE t_hero TO hero`
4. 查看表hero的信息
`SHOW CREATE TABLE hero`
5. 查询表hero结构
`DESC hero`
6. 删除表hero
`DROP TABLE hero`
7. 删除数据库mydb3
`DROP DATABASE mydb3`

作业:

1. 创建数据库mydb4 字符集utf8并使用
2. 创建teacher表 有名字(name)字段
3. 添加表字段: 最后添加age 最前面添加id(int型) , age前面添加salary工资(int型)
4. 删除age字段
5. 修改表名为t
6. 删除表t
7. 删除数据库mydb4

DML 数据操作语言

DML:对表中数据进行操作的语言, 涵盖的内容为:INSERT(增),DELETE(删),UPDATE(改)

准备一张表:

```
CREATE TABLE person(  
    name VARCHAR(30),  
    age INT(3)  
)
```

插入数据(INSERT)

语法:

```
INSERT INTO 表名[(字段1, 字段2, ...)] VALUES(字段的1值, 字段2的值, ...)
```

注:在语法定义上"[]"中的内容表示可写可不写

例

```
INSERT INTO person(name, age) VALUES('张三', 25)  
INSERT INTO person(age, name) VALUES(33, '李四')
```

注:

- 数据库中字符串的字面量是使用单引号'表达的
- VALUES中指定的值要与指定的字段名个数,顺序,以及类型完全一致
- 查看表中数据

```
SELECT * FROM person
```

插入默认值

当插入数据时不指定某个字段, 那么该字段插入默认值。若创建表时字段没有显示的指定默认值时, 默认值插入NULL

例:

```
INSERT INTO person (name) VALUES ('王五')
```

| | name | age |
|---|------|--------|
| 1 | 张三 | 25 |
| 2 | 李四 | 33 |
| 3 | 王五 | <null> |

注:

- age字段没有指定, 因此插入默认值NULL
- 数据库中任何字段任何类型默认值都是NULL, 当然可以在创建表时使用DEFAULT指定。

修改person表中age的默认值为20

```
ALTER TABLE person CHANGE age age INT(3) DEFAULT 20
```

修改后向person表中插入数据

```
INSERT INTO person (name) VALUES ('赵六')
```

提示: 也可在创建person表时为字段指定默认值

```
CREATE TABLE person(  
    name VARCHAR(30) DEFAULT '无名氏',  
    age INT(3) DEFAULT 20  
)
```

为age添加默认值20:

| | Field | Type | Null | Key | Default |
|---|-------|-------------|------|-----|---------|
| 1 | name | varchar(30) | YES | | <null> |
| 2 | age | int(3) | YES | | 20 |

插入'赵六'

| | name | age |
|---|------|--------|
| 1 | 张三 | 25 |
| 2 | 李四 | 33 |
| 3 | 王五 | <null> |
| 4 | 赵六 | 20 |

全列插入

当我们插入数据是不指定任何字段名时，就是全列插入。此时VALUES子句中需要为每个字段指定值。并且要求(个数，顺序，类型必须与表完全一致)

例:

```
INSERT INTO person VALUES('钱七',21);
```

若某个字段有默认值，可以用DEFAULT表示默认值

```
INSERT INTO person VALUES('钱七',DEFAULT);
```

若某个字段想插入NULL值，可以使用NULL表达

```
INSERT INTO person VALUES('钱七',NULL);
```

下列为错误示例

```
INSERT INTO person VALUES('钱七');           列的值个数不匹配
```

```
INSERT INTO person VALUES(27,'钱七');        列的值类型不匹配
```

修改数据(UPDATE)

语法

```
UPDATE 表名 SET 字段1=新值1[, 字段2=新值2,...] [WHERE 过滤条件]
```

例

```
UPDATE person SET age=15
```

上述SQL执行后会导致person表中所有记录的age字段值都被改为15

UPDATE语句通常都要添加WHERE子句，用于添加要修改记录的条件，否则全表修改！

修改指定记录(添加WHERE子句)

例

将张三的年龄改成20岁

```
UPDATE person
```

```
SET age=20
```

```
WHERE name='张三'           仅将person表中某条记录name字段值为‘张三’的age改为20
```

将person表中年龄为16岁的人改为36岁

```
UPDATE person
```

```
Set age=36
```

```
WHERE age=16                person表中凡是满足WHERE条件的记录都会被修改
```

WHERE子句中常用的条件

=, <, <=, >, >=, <> (不等于使用<>。!=不是所有数据库都支持)

将person表中年龄大于30岁的人改为年龄29

```
UPDATE person
SET age=29
WHERE age>30
```

将一个计算表达式的结果作为值使用

将person表中每个人的年龄涨1岁

```
UPDATE person
SET age=age+1
```

将每条记录年龄的值+1后再修改回年龄字段

同时修改多个字段

将'李四'改为'李老四'且年龄改为55岁

```
UPDATE person
SET name='李老四', age=55
WHERE name='李四'
```

删除语句(DELETE)

语法:

```
DELETE FROM 表名 [WHERE 过滤条件]
```

注:DELETE语句通常都要添加WHERE子句, 否则是清空表操作

例:

删除'李老四'

```
DELETE FROM person
WHERE name='李老四'
```

仅删除person表中满足WHERE条件的记录

删除年龄大于等于30岁的人

```
DELETE FROM person
WHERE age>=30
```

清空表操作

```
DELETE FROM person
```

练习:

1. 创建数据库day1db 字符集utf8并使用

```
CREATE DATABASE day1db CHARSET=utf8
USE day1db
```
2. 创建t_hero表, 有name字段, 字符集utf8

```
CREATE TABLE t_hero(
    name VARCHAR(30)
)CHARSET=utf8
```
3. 修改表名为hero

```
RENAME TABLE t_hero TO hero
```
4. 最后面添加价格字段money, 最前面添加id字段, name后面添加age字段

```
ALTER TABLE hero ADD money INT
ALTER TABLE hero ADD id INT FIRST
ALTER TABLE hero ADD age INT AFTER name
```
5. 表中添加以下数据: 1, 李白, 50, 6888 2, 赵云, 30, 13888 3, 刘备, 25, 6888

```
INSERT INTO hero(id,name,age,money) VALUES(1, '李白', 50, 6888)
INSERT INTO hero(id,name,age,money) VALUES(2, '赵云', 30, 13888)
INSERT INTO hero(id,name,age,money) VALUES(3, '刘备', 25, 6888)
```
6. 修改刘备年龄为52岁

```
UPDATE hero
SET age=52
WHERE name='刘备'
```
7. 修改年龄小于等于50岁的价格为5000

```
UPDATE hero
SET money=5000
WHERE age<=50
```
8. 删除价格为5000的信息

```
DELETE FROM hero
WHERE money=5000
```
9. 删除表, 删除数据库

```
DROP TABLE hero
DROP DATABASE day1db
```

总结

- DML:数据操作语言, 它是对表中数据进行操作的语言, 涵盖操作:增(INSERT), 删(DELETE), 改(UPDATE)
- INSERT语句用于将数据插入表中
 - INSERT时指定的字段顺序可以与表不一致, 但是VALUES子句中值的顺序要与指定顺序一致
 - INSERT时可以不指定某字段, 那么此时该条记录该字段会插入默认值
 - INSERT时为某个字段插入其指定的默认值, 使用关键字DEFAULT。
字段若指定了默认值则使用该默认值, 否则字段默认值为NULL。
 - INSERT时使用NULL插入一个字段为null值。
 - INSERT时可以不指定字段名, 那么为全列插入, 此时VALUES子句指定的值顺序, 个数, 类型必须与表结构完全一致。
- UPDATE语句用于修改表中数据

- DELETE语句用于删除表中数据
- UPDATE语句和DELETE语句通常都要添加WHERE子句，否则是对全表所有记录操作。

数据库常用的数据类型

不同的数据库数据类型不完全一致

数字类型

整数类型:INT(m)和BIGINT(m)

- m:整数的位数
- int(5):最大保存一个5位整数。
 - 若保存的数字为18，实际存的时候前面会补0:00018

浮点类型:DOUBLE(m,n)

- m:数字的位数
 - n:小数部分的位数
 - m包含n
 - DOUBLE(5,3):该数字总共5位，其中有3位小数(意味着整数位2位)。
- 最大个保存的数字为:99.999
- 当保存的数字精度超过可保存精度时，会进行4舍5入

例如person表中salary字段的类型:DOUBLE(7,4)

```
INSERT INTO person (salary) VALUES(553.12568)
```

实际salary保存的数字为:553.1257

```
INSERT INTO person VALUES ('钱七',DEFAULT,553.12568);
```

| | name | age | salary |
|---|------|-----|----------|
| 1 | 张三 | 16 | <null> |
| 2 | 李四 | 16 | <null> |
| 3 | 王五 | 16 | <null> |
| 4 | 赵六 | 16 | <null> |
| 5 | 钱七 | 20 | 553.1257 |

- 当最大值需要进行四舍五入时，此时会报错:值超出了范围

```
67 ! INSERT INTO person VALUES ('孙八',DEFAULT,999.99999);
```

[22001][1264] Data truncation: Out of range value for column 'salary' at row 1

字符类型

定长字符:CHAR(n)

- n表示长度，单位是字符。
- name CHAR(10):name可以保存10个字符
- 最大值为255个字符
- 表中该字段保存数据时，实际在磁盘中保存数据一定是指定长度的字符。当实际保存的字符不足时会补充空格来达到长度。

例如:name插入数据时为'张三'。那么实际存储时后面还有8个空格要占够10个字符。

- 优点:每条记录占用的字节长度是固定的，这对于数据库扫描磁盘查询数据时效率高。
- 缺点:浪费磁盘空间

变长字符:VARCHAR(n)

- n表示长度，单位是字节。
- 最大值为65535个字节
- 表中该字段保存数据时，实际在磁盘中保存数据用多少占多少字节。

例如:name VARCHAR(30)

插入数据时name为'张三',实际存储时磁盘中仅6个字节。

- 优点:不浪费磁盘空间
- 缺点:查询速度慢

变长字符:TEXT(n)

- n表示长度，单位是字符
- 最大值为65535

日期类型

- DATE:仅保存年月日
- TIME:时分秒
- DATETIME:保存年月日时分秒
- TIMESTAMP:时间戳，记录UTC时间。自1970-01-01 00:00:00到表达的时间所经过的毫秒

例

- 准备一张表测试各种类型

```
CREATE TABLE userinfo(  
    id INT,  
    username VARCHAR(30),  
    gender CHAR(1),  
    birth DATETIME,  
    salary DOUBLE(7,2)  
)
```

- 插入日期时，可以以字符串形式插入，格式为:"yyyy-MM-dd hh:mm:ss"

注:MM表示两位数字的月，mm表示两位数字的分


```
INSERT INTO userinfo(id,username,gender,birth,salary)
VALUES (1,'张三','男','2001-02-03 12:22:55',5000.99)
```

- 插入日期时，如果类型为DATETIME，插入字符串中可以不指定时分秒

```
INSERT INTO userinfo(id,username,gender,birth,salary)
VALUES (2,'李四','女','1998-08-24',6000)
```

此时时分秒都为0

| id | username | gender | birth | salary |
|----|----------|--------|---------------------|---------|
| 1 | 张三 | 男 | 2001-02-03 12:22:55 | 5000.99 |
| 2 | 李四 | 女 | 1998-08-24 00:00:00 | 6000 |

- DATETIME类型插入数据时不能忽略年月日

```
INSERT INTO userinfo(id,username,gender,birth,salary)
VALUES (3,'王五','男','12:15:32',5000)
```

会报错，DATETIME字段不能只插入时分秒

```
85 ! INSERT INTO userinfo(id,username,gender,birth,salary)
86 VALUES (3,'王五','男','12:15:32',5000)
```

[22001][1292] Data truncation: Incorrect datetime value: '12:15:32' for column 'mydb`.`userinfo`.`birth` at row 1

- 插入DOUBLE值时精度超过部分会四舍五入，整数部分若超过长度则会报错

```
INSERT INTO userinfo(id,username,gender,birth,salary)
VALUES (3,'王五','男','1985-06-01 12:15:32',100000)
```

```
85 ! INSERT INTO userinfo(id,username,gender,birth,salary)
86 VALUES (3,'王五','男','1985-06-01 12:15:32',100000)
```

[22001][1264] Data truncation: Out of range value for column 'salary' at row 1

约束条件

可以对表添加约束条件，这样一来仅当满足约束条件的操作才可以进行。这样做可以更好的为我们的业务服务，保证数据库的操作是服务业务要求的。

主键约束(PRIMARY KEY)

- 主键的特点：**非空且唯一**。符合该特点的值才可以用来标识表中唯一的一条记录。
- 通常一张表中的第一个字段都是主键字段，名字通常选取为:id
- 主键约束一张表只能为一个字段添加

例

```
CREATE TABLE user1(  
    id INT PRIMARY KEY,  
    name VARCHAR(30),  
    age INT(3)  
);  
  
INSERT INTO user1(id,name,age) VALUES (1,'张三',22)  
INSERT INTO user1(id,name,age) VALUES (2,'李四',33)
```

- 主键字段不能插入重复的值

```
98 ! INSERT INTO user1(id,name,age)  
99 VALUES(1,'王五',36);  
  
[23000][1062] Duplicate entry '1' for key 'PRIMARY'
```

- 主键字段不能插入NULL值

```
101 ! INSERT INTO user1(id,name,age)  
102 VALUES(NULL,'王五',36);  
  
[23000][1048] Column 'id' cannot be null
```

- 不能修改重复的值

```
104 ! UPDATE user1  
105 SET id=1  
106 WHERE name='李四'  
  
[23000][1062] Duplicate entry '1' for key 'PRIMARY'
```

- 不能修改为NULL

```
104 ! UPDATE user1  
105 SET id=NULL  
106 WHERE name='李四'  
  
[23000][1048] Column 'id' cannot be null
```

具体主键约束的字段通常会配合自增(AUTO_INCREMENT)使用

- 将字段定义为主键约束时，同时指定: **AUTO_INCREMENT**

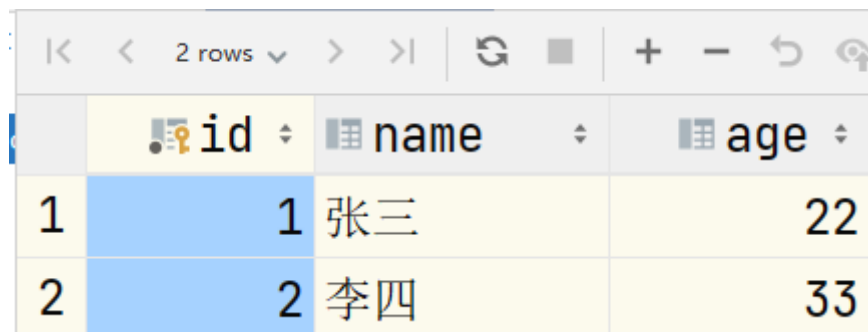
```
CREATE TABLE user2(  
  id INT PRIMARY KEY AUTO_INCREMENT,  
  name VARCHAR(30),  
  age INT(3)  
);
```

也可以修改现有的表字段

```
ALTER TABLE user1 CHANGE id id INT PRIMARY KEY AUTO_INCREMENT
```

- 当主键字段具有自增功能时，插入数据则可以忽略主键字段的插入

```
INSERT INTO user2 (name,age) VALUES('张三',22)  
INSERT INTO user2 (name,age) VALUES('李四',33)
```



| | id | name | age |
|---|----|------|-----|
| 1 | 1 | 张三 | 22 |
| 2 | 2 | 李四 | 33 |

- 当显示的给自增字段插入NULL值时，该字段的值仍然会自增。(不推荐)

```
INSERT INTO user2 (id,name,age) VALUES(NULL,'王五',36)
```

非空约束(NOT NULL)

被非空约束修饰的字段表中每条记录该字段必须有值，不能为NULL

例

```
CREATE TABLE user3(  
  id INT PRIMARY KEY AUTO_INCREMENT,  
  name VARCHAR(30) NOT NULL,  
  age INT(3)  
)
```

- 非空约束在查看表结构时有所体现

```
DESC user3
```

| | Field | Type | Null | Key | Default | Extra |
|---|-------|-------------|------|-----|---------|----------------|
| 1 | id | int(11) | NO | PRI | <null> | auto_increment |
| 2 | name | varchar(30) | NO | | <null> | |
| 3 | age | int(3) | YES | | <null> | |

- 不能将NULL值插入具有非空约束的字段中

```
INSERT INTO user3 (name,age) VALUES(NULL,22)
```

126 ! `INSERT INTO user3 (name,age) VALUES(NULL,22)`

[23000][1048] Column 'name' cannot be null

- 插入数据时不能忽略具有非空约束的字段(字段默认插入NULL)

```
INSERT INTO user3(age) VALUES(33)
```

128 ! `INSERT INTO user3(age) VALUES(33)`

[HY000][1364] Field 'name' doesn't have a default value

DQL数据查询语言

DQL是用于查询表中数据的语言。

语法:

```
SELECT 子句
FROM 子句
WHERE 子句
JOIN...ON...子句
GROUP BY 子句
HAVING 子句
ORDER BY子句
```

一条DQL语句至少要包含的两个子句分别是SELECT子句和FROM子句

- SELECT子句用于指定查询的字段名称。选定的字段会体现在查询结果集中
- FROM子句用于指定数据来自那些表(指定查询的表)

基础查询

由SELECT和FROM构成。

语法:

- SELECT子句和FROM子句

```
SELECT 字段1[, 字段2, 字段3, ...]
FROM 表1[, 表2, 表3...]
```

- SELECT子句中可以使用*表达查询所有的字段

实际开发中不推荐使用*,因为查询低效

例

查询teacher表中的所有数据

```
SELECT * FROM teacher
```

上述SQL执行时，数据库会先解析"*",这个操作会导致数据库先查看内部数据字典了解表的字段后才能查看表中数据。由于查询是非常频繁的操作，因此每次查看数据字段无疑是浪费性能且消耗时间的操作！全字段查询时应当在SELECT子句中将所有字段全部列出来(后期java程序执行时)。如果是手动临时查询，可以使用*。

查看每个老师的名字，职称，工资，性别

```
SELECT name,title,salary,gender
FROM teacher
```

WHERE子句

在DQL中用于筛选查询的记录。最终数据库仅将满足WHERE子句条件的记录体现在结果集中。

例

- 查看职称为"一级讲师"的老师名字和工资？

```
1:查询的是老师信息，能确定FROM子句中的表名为teacher
2:查看的是老师的名字和工资，能确定SELECT子句中的字段name和salary
3:由于仅查看"一级讲师",能确定WHERE子句过滤条件为title='一级讲师'
SELECT name,salary,title
FROM teacher
WHERE title='一级讲师'
```

- 查看除'刘苍松'老师以外的其他老师的名字，工资和年龄？

```
SELECT name,salary,age
FROM teacher
WHERE name<>'刘苍松'
```

- 查看所有职位是'大队长'的同学的名字年龄和性别

```
SELECT name,age,gender
FROM student
WHERE job='大队长'
```

- 查看年龄在30(含)岁以上的老师都有谁？ 查看老师的名字,年龄,性别,工资

```
SELECT name,age,gender,salary
FROM teacher
WHERE age>=30
```

使用AND"与"和OR"或"来连接多个条件进行查询

- AND:都为真时才为真
- OR:都为假时才为假

例

- 查看7岁的大队长都是谁?列出:名字, 年龄, 性别, 职位

```
SELECT name,age,gender,job
FROM student
WHERE job='大队长' AND age=7
```

- 查看班级编号小于6的所有中队长的名字, 年龄, 性别, 职位和所在班级编号

```
SELECT name,age,gender,job,class_id
FROM student
WHERE class_id<6 AND job='中队长'
```

- 查看所有一级讲师和三级讲师的名字, 职称, 工资?

```
SELECT name,title,salary
FROM teacher
WHERE title='一级讲师' OR title='三级讲师'
```

- 查看所有大队长, 中队长和小队长的名字, 年龄, 性别

```
SELECT name,age,gender
FROM student
WHERE job='大队长' OR job='中队长' OR job='小队长'
```

- 查看班级编号在6(含)以下的所有大队长和中队长的名字, 年龄, 职位, 班级编号

AND的优先级是高于OR的

可以使用()提高优先级

```
SELECT name,age,job,class_id
FROM student
WHERE class_id<=6 AND job='大队长' OR job='中队长'
```

上述SQL的条件应当读作:查看班级编号6以下的大队长和所有班级编号的中队长

```
SELECT name,age,job,class_id
FROM student
WHERE class_id<=6 AND (job='大队长' OR job='中队长')
```

提高OR的优先级满足查询要求

IN(列表):等于列表其中之一(在列表中)

例

- 查看所有的大队长，中队长，小队长的名字，年龄，性别

```
SELECT name,age,gender
FROM student
WHERE job='大队长' OR job='中队长' OR job='小队长'
```

等价于

```
SELECT name,age,gender
FROM student
WHERE job IN('大队长','中队长','小队长')
```

- 查看所有一级讲师，二级讲师，三级讲师的老师名字，职称和工资？

```
SELECT name,title,salary
FROM teacher
WHERE title IN ('一级讲师','二级讲师','三级讲师')
```

NOT IN(列表):不在列表中，不能等于列表中的所有项

- 查看一级讲师和二级讲师以外的所有老师的名字，职称，工资？

```
SELECT name,title,salary
FROM teacher
WHERE title NOT IN('一级讲师','二级讲师')
```

- 查看不是中队长和大队长以及小队长的其他学生的名字，年龄，职位

```
SELECT name,age,job
FROM student
WHERE job NOT IN('大队长','中队长','小队长')
```

BETWEEN...AND...:在一个范围之内

- 查看工资在2000-5000之间的老师的名字，工资，职称？

```
SELECT name,salary,title
FROM teacher
WHERE salary>=2000 AND salary<=5000
等价于
SELECT name,salary,title
FROM teacher
WHERE salary BETWEEN 2000 AND 5000
                        下限      上限
```

- 查看年龄在7-10岁之间的学生的名字，年龄，性别

```
SELECT name,age,gender
FROM student
WHERE age BETWEEN 7 AND 10
```

- 查看年龄在20-35岁之间的男老师的名字，职称，年龄

```
SELECT name,title,age
FROM teacher
WHERE age BETWEEN 20 AND 35
AND gender='男'
```

- 查看3-5楼的班级名称都是什么？

```
SELECT name
FROM class
WHERE floor BETWEEN 3 AND 5
```

DISTINCT去重操作。在结果集中去除指定字段值相同的记录

语法

```
SELECT DISTINCT 字段1[,字段2,...]
FROM 表名
...
```

- DISTINCT关键字必须紧跟在SELECT子句之后
- DISTINCT后面可以指定多个字段，当这几个字段组合相同的记录会在结果集中被去除

例

- 查看老师的职称都有哪些？


```
SELECT title
```

```
FROM teacher
```

上述SQL语句的查询结果集是展现每个老师的职称，与查询需求不匹配

```
SELECT DISTINCT title
```

```
FROM teacher
```

将查询结果集中重复的title去除后得到正确效果

- 查看学生的职位都有哪些?

```
SELECT DISTINCT job
```

```
FROM student
```

- 查看各年龄的职位都有哪些?

```
SELECT DISTINCT age, job
```

```
FROM student
```

练习:

1. 查看负责课程编号(subject_id)为1的男老师都有谁?
2. 查看工资高于5000的女老师都有谁?
3. 查看工资高于5000的男老师或所有女老师的工资?
4. 查看所有9岁学生的学习委员和语文课代表都是谁?
5. 查看工资在6000到10000之间的老师以及具体工资?
6. 查看工资在4000到8000以外的老师及具体工资?
7. 查看老师负责的课程编号都有什么?
8. 查看所有女老师的职称都是什么?
9. 查看7-10岁的男同学的职位都有哪些?
10. 查看一级讲师和二级讲师的奖金(comm)是多少?
11. 查看除老板和总监的其他老师的工资和奖金是多少?

