

# MYSQL

## 数据库

### 什么是数据库

数据库是以一定方式储存在一起、能与多个用户共享、具有尽可能小的冗余度、与应用程序彼此独立的数据集合，可视为电子化的文件柜——存储电子文件的处所，用户可以对文件中的数据进行新增、查询、更新、删除等操作。

### 数据的存储方式

#### 1. 数据保存在内存

例如:数组,集合,new出来的对象存储在堆中.堆是内存中的一小块空间

优点：内存速度快 缺点：断电/程序退出,数据就清除了.内存价格贵

#### 2. 数据保存在普通文件 优点：永久保存 缺点：查找，增加，修改，删除数据比较麻烦，效率低

#### 3. 数据保存在数据库 优点：永久保存,通过 `SQL` 语句比较方便的操作数据库

### 数据库的优点

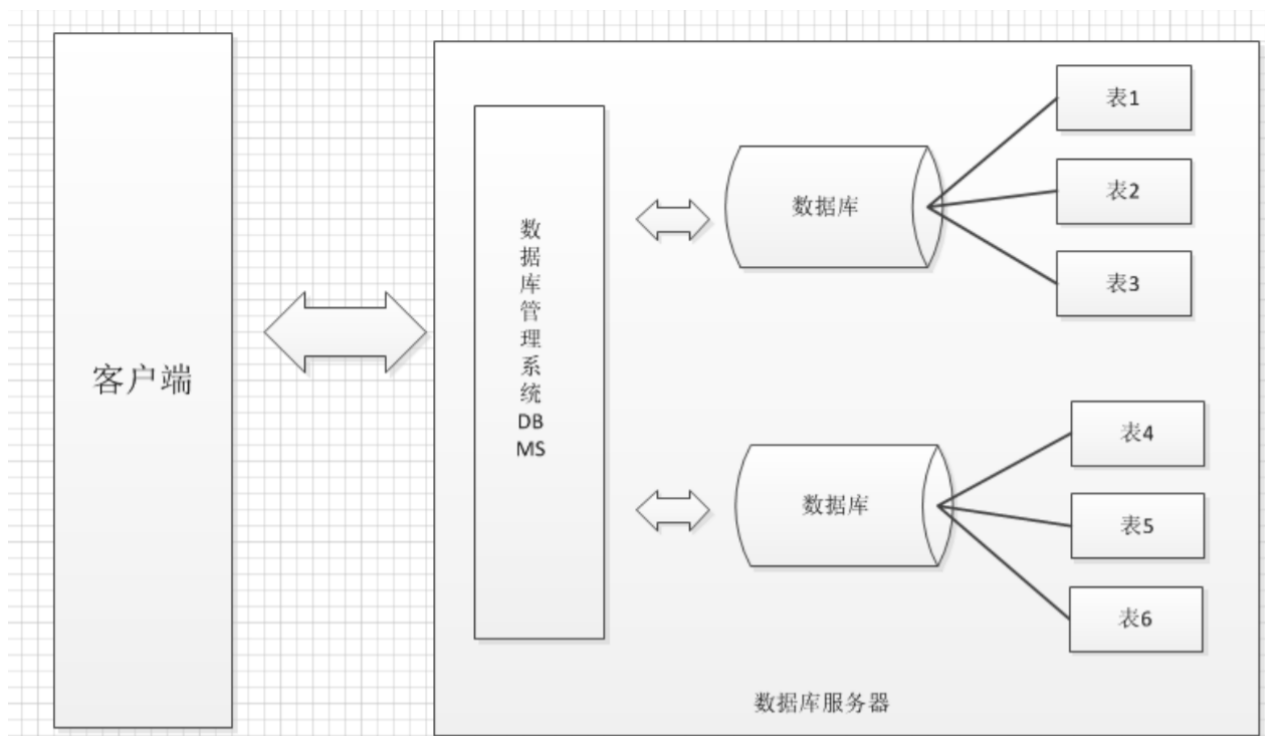
数据库是按照特定的格式将数据存储于文件中，通过 `SQL` 语句可以方便的对大量数据进行增、删、改、查操作，数据库是对大量的信息进行管理的高效的解决方案。

### 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

### 数据库管理系统、数据库和表的关系

数据库管理程序(DBMS)可以管理多个数据库，一般开发人员会针对每一个应用创建一个数据库。为保存应用中实体的数据，一般会在数据库创建多个表，以保存程序中实体的数据。数据库管理系统、数据库和表的关系如图所示：



先有数据库 → 再有表 → 再有数据 一个库包含多个表

## 常见数据库

Rank			DBMS	Database Model	Score		
Jul 2023	Jun 2023	Jul 2022			Jul 2023	Jun 2023	Jul 2022
1.	1.	1.	Oracle	Relational, Multi-model	1256.01	+24.54	-24.28
2.	2.	2.	MySQL	Relational, Multi-model	1150.35	-13.59	-44.53
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	921.60	-8.47	-20.53
4.	4.	4.	PostgreSQL	Relational, Multi-model	617.83	+5.01	+1.96
5.	5.	5.	MongoDB	Document, Multi-model	435.49	+10.13	-37.49
6.	6.	6.	Redis	Key-value, Multi-model	163.76	-3.59	-9.86
7.	7.	7.	IBM Db2	Relational, Multi-model	139.81	-5.07	-21.40
8.	8.	8.	Elasticsearch	Search engine, Multi-model	139.59	-4.16	-14.74
9.	9.	9.	Microsoft Access	Relational	130.72	-3.73	-14.37
10.	10.	10.	SQLite	Relational	130.20	-1.02	-6.48

**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。

**SyBase**：已经淡出历史舞台。提供了一个非常专业数据建模的工具 **PowerDesigner**。

**SQLite**：嵌入式的小型数据库，应用在手机端。

常用数据库：`MYSQL`，`Oracle` 在web应用中，使用的最多的就是 `MySQL` 数据库，原因如下：

1. 开源、免费
2. 功能足够强大，足以应付web应用开发

## 安装常见问题

### 将mysql添加到环境变量

将 `mysql` 的 `bin` 目录地址添加到 系统环境变量 --> `PATH` 中

### 将mysql添加到服务

以管理员的方式启动 `cmd` (命令提示窗口)，使用命令进入到 `[mysql]\bin`，执行如下命令。

```
mysqld --install (服务名)
```

如:

```
mysqld --install mysql
```

删除服务命令是:

```
mysqld --remove 服务名
```

### mysql端口被占用解决

在 `cmd` 窗口下执行如下命令:

```
netstat -ano|findstr 3306
```

查找正在执行的 3306 端口程序

```
>netstat -ano|findstr 3306
TCP    0.0.0.0:3306      0.0.0.0:0        LISTENING       716
TCP    [::]:3306        [::]:0            LISTENING       716
TCP    [::1]:54523      [::1]:3306       TIME_WAIT        0
TCP    [::1]:54524      [::1]:3306       TIME_WAIT        0
```

如果出现如图所示列表表示以上程序使用了3306端口，找到程序的 PID (最后一列)

去任务管理栏找到对应程序结束任务或者使用如下命令关闭进程：

```
1 | taskkill /pid xxx
```

## 忘记mysql密码

以下操作均以管理员方式进行。

1. 打开 cmd 关闭 mysql 服务, `net stop mysql`
2. 以管理员身份在命令提示窗口( cmd )中进入到 `mysql根目录->bin` 文件夹，输入：

```
1 | mysqld --console --skip-grant-tables --shared-memory
```

跳过权限认证

3. 重新打开一个 cmd 窗口，输入 `mysql`

```
mysql -uroot -p
```

不输入密码即可进去 `mysql`

4. 连接权限数据库：

```
use mysql
```

5. 修改数据库连接密码,置为空:

```
update user set authentication_string = '' where user = 'root';
```

6. 退出

```
exit  
  
quit
```

7. 关闭第一个窗口

重启 `mysql` 服务即可。

再重新进入 `mysql` ,修改密码

```
1 | ALTER USER 'root'@'localhost' IDENTIFIED BY 'new_password';
```

## 连接MySQL

```
1 | mysql -u 用户名 -p
```

- `mysql --help` 查询所有参数

## SQL 分类

- **DDL:** 数据定义语句。如: CREATE / ALTER / DROP
- **DML:** 数据操纵语句。如: INSERT / UPDATE / DELETE
- **DQL:** 数据查询语句。如: SELECT

所有的SQL都应该以英文状态下的分号结束 ;

# 建库 建表

数据库语句的关键词建议最好大写

- 创建数据库语法结构:

- `CREATE DATABASE [ IF NOT EXISTS ] db_name`
- `CREATE DATABASE` 表示创建数据库，是SQL中的关键词
- `db_name`是要创建的数据库名称

```
1 | CREATE DATABASE company_info;
```

数据库中，命名一般是使用 `_` 连接多个单词;

数据库中 SQL 语句执行失败后会有错误提示，错误提示包括错误信息和错误编号。我们可以直接拿错误编号去搜索。

有时在创建数据库时要设置数据库的编码。MySQL 8 默认编码为 `UTF-8mb4`，满足我们需求所以不需要设置。如果使用的是低版本数据库则需要在创建数据库时加上 `CHARACTER SET utf8` 去设置编码。或者使用 `ALTER DATABASE db_name CHARACTER SET UTF8;` 修改

- 使用数据库:

- `USE db_name`
- 使用USE关键词来指定要使用的数据库

```
1 | USE company_info;
```

- 删除数据库语法结构

- `drop database db_name`
- 使用DROP关键字删除数据库

```
1 | DROP DATABASE company_info;
```

- 显示所有的数据库

## 创建数据表

- 创建数据表的语法结构
  - CREATE TABLE tab\_name(  
col\_name datatype default null/number comment '注释',  
col\_name datatype  
) [CHARACTER set 编码格式];
    - 使用CREATE TABLE table关键词创建数据表
    - tab\_name是数据表的名称
    - col\_name是列名称
    - datatype是列的数据类型
    - DEFAULT 是默认值
    - COMMENT 是注释

```
1 CREATE TABLE dept(  
2 deptno INT DEFAULT 1 COMMENT '部门编号',  
3 deptname VARCHAR(20) DEFAULT NULL COMMENT '部门名称'  
4 )CHARACTER SET utf8;  
5  
6 -- 显示所有表  
7 SHOW tables;
```

数  
据  
类  
型  
名  
称

描述

数据类型名称	描述
<b>S M A L L I N T</b>	小的整数，带符号的范围是-32768到32767.无符号的范围是0到65535
<b>M E D I U M I N T</b>	中等大小整数-8388608到8388607，0到16777215
<b>I N T/ I N T E G E R</b>	普通大小的整数，-2147483648到2147483647，0到4294967295
<b>B I G I N T</b>	大整数，-9223372036854775808到9223372036854775807，0到18446744073709551615
<b>F L O A T</b>	小(单精度)浮点数，允许的值-3.402823466E+38到-1.175494351E-38,0和1.175494351E-38到3.402823466E+38,这些是理论限制，基于IEEE标准。实际的范围根据硬件或操作系统的不同可能稍微小些
<b>D O U B L E</b>	普通大小(双精度)浮点数，允许的值-1.7976931348623157E+380到-2.2250738585072014E-308,0和2.2250738585072014E-38到1.7976931348623157E+308.这些事理论限制，基于IEEE标准。实际的范围根据硬件或操作系统的不同可能稍微小些
<b>D A T E</b>	日期，支持的范围为‘1000-01-01’到‘9999-12-31’，MySQL以‘YYYY-MM-DD’格式显示DATE值，但允许使用字符串或数字为DATE列分配值



数据类型名称	描述
<b>DATETIME</b>	日期和时间的组合。支持的范围是‘上面加上00：00：00’到‘上面第二个加上23：59：59’。MySQL以YYYY-MM-DD HH:MM:SS“格式显示DATETIME值，但允许使用字符串或数字为DATETIME列分配值
<b>TIMESTAMP</b>	时间戳，范围是'1970-01-01 00:00:00'到2037年
<b>TIME</b>	时间，范围是‘-838：59：59’到‘838：59：59’。MySQL以‘HH:MM:SS’格式显示TIME值，但允许使用字符串或数字为TIME列分配值
<b>YEAR</b>	两位或四位格式的年。默认是四位格式。在四位格式中，允许的值是1901到2155和0000。在两位格式中，允许的值是70到69，表示从1970到2069年。MySQL以yyyy格式显示YEAR值，但允许使用字符串或数字为YEAR列分配值
<b>CHAR(M)</b>	固定长度字符串，当保存时在右侧填充空格以达到指定长度。M表示列长度。M的范围是0到255个字符
<b>VARCHAR(M)</b>	变长字符串。M表示最大列长度。M的范围是0到65535。（VARCHAR的最大实际长度由最长的行的大小和使用的字符集确定。最大有效长度是65355字节）

数据类型名称	描述
BLOB [(M)]	最大长度为65535(216-1)字节，=的BLOB列，可以给出该类型的可选长度M。如果给出，则MySQL将列创建为最小的但是足以容纳M字节长度的值的BLOB类型
TEXT [(M)]	长字符串，最大长度为65535(216-1)字符的TEXT列。可以给出可选长度M。则MySQL将列创建为最小的但是足以容纳M字符长度的值的TEXT类型。

所有的数据类型[Data Types](#)

### 查看表结构

```
1 -- 查看表结构
2 DESCRIBE dept;
3 DESC dept;
```

信息	结果 1	剖析	状态		
Field	Type	Null	Key	Default	Extra
▶ deptno	int(11)	YES		1	
deptname	varchar(20)	YES		(Null)	

### 创建和某表结构一样的表

```
1 -- 创建和dept结构一样的表
2 CREATE TABLE d LIKE dept;
3
4 CREATE TABLE d AS select * from dept;
```

## 删除表

```
1 | DROP TABLE table_name
```

## 添加列

```
1 | ALTER TABLE d ADD id INT;
```

## 修改列属性

```
1 | ALTER TABLE d MODIFY id VARCHAR(20);
```

## 修改列名

```
1 | ALTER TABLE d CHANGE id ss VARCHAR(20);
```

**MODIFY** 和 **CHANGE** 关键字用于修改表的列名、数据类型以及列的约束。区别在于：

- **MODIFY** 不会修改列名，**CHANGE** 关键字允许同时修改列名。
- 使用 **CHANGE** 时，必须指定原始列名、新列名以及新的数据类型和列约束（如NOT NULL、DEFAULT等）

```
ALTER TABLE tb_name CHANGE old_column_name new_column_name  
new_data_type new_constraints;
```

## 删除列

```
1 | ALTER TABLE d DROP ss;
```

## 重命名表

```
1 | RENAME TABLE d TO dd;
```

## CRUD操作

- 对数据表中的数据操作通常有添加(Create)、查询(Retrieve)、修改(Update)、删除>Delete)、简称为CRUD。

### 添加数据

**INSERT INTO table\_name VALUES(值列表)**

**INSERT INTO table\_name (列列表) VALUES(值列表)**

```
1  -- 不推荐使用
2  INSERT INTO dept VALUE(1,'研发部');
3  -- 2
4  INSERT INTO dept VALUES(2,'销售部');
5  INSERT INTO dept VALUES(3,'行政部'),(4,'技术部');
6  -- 3
7  INSERT INTO dept(deptno,deptname)VALUES(5,'安保部');
```

区别:

- value和values的区别，values可以同时插入多条数据用逗号隔开
- dept和dept(列名，列名。。。)区别，如果不写列表必须按照列表创建时的顺序每一列都要添加
- 有列名的按照列名排列顺序添加

### 查询数据

```
1  -- 查询所有数据
2  SELECT * FROM dept;
3  -- 查询某列的数据
4  SELECT deptname FROM dept;
5  -- 根据条件查询*
6  SELECT deptno FROM dept WHERE deptname='销售部';
```

### 修改数据

```

1  --全部修改为6
2  UPDATE dept SET deptno=6;
3  --根据条件修改
4  UPDATE dept SET deptno=1 WHERE deptname='研发部';
5  UPDATE dept SET deptno=2 WHERE deptname='销售部';
6  UPDATE dept SET deptno=3 WHERE deptname='行政部';
7  UPDATE dept SET deptno=4 WHERE deptname='技术部';
8  UPDATE dept SET deptno=5 WHERE deptname='安保部';

```

## 删除数据

```

1  -- 删除数据 ,一定要加 where 条件
2  DELETE FROM dept WHERE deptno=5;
3  -- 全部删除
4  DELETE FROM dept;
5  -- 清空/截断 所有数据(慎用)
6  TRUNCATE TABLE dept;

```

### 区别

- delete from dd;
- truncate table dd;
- delete 是清空表中的数据
- truncate 是清空表数据，重新创建一个一样表

## where条件连接

当 sql 语句中的条件有多条时，可以将多个条件连接起来。他们之间的关系有以下几种：

### and

**a and b** : 表示 需要同时满足 a 条件 和 b 条件

### or

**a or b** : 表示 满足 a 条件 或 b 条件都可以

### in

**in(a, ... ,b)** : 表示在 a 及 b 这些值中都可以

### like

模糊查询, % 表示任意个字符 \_ 表示一个字符

# 数据备份和还原

## 命令行备份

### 备份结构

#### 1.备份表结构

```
1 | mysqldump -u root -p -d dbname table1 table2 ... > a.sql
```

#### 2.备份数据库的所有表结构

```
1 | mysqldump -u root -p -d dbname > b.sql
```

#### 3.备份多个数据库的所有表结构

```
1 | mysqldump -u root -p -d --databases db1 db2... > c.sql
```

#### 4.备份所有数据库的表结构

```
1 | mysqldump -u root -p -d --all-databases > d.sql
```

### 备份数据和结构

(相当于在备份结构的语法上去掉-d选项)

#### 1.备份表结构和数据

```
1 | mysqldump -u root -p dbname table1 table2 ... > a.sql
```

#### 2.备份数据库的所有表结构和数据

```
1 | mysqldump -u root -p dbname > b.sql
```

#### 3.备份多个数据库的表结构和数据

```
1 | mysqldump -u root -p --databases db1 db2 > c.sql
```

#### 4.备份所有数据库的表结构和数据

```
1 | mysqldump -u root -p --all-databases > d.sql
```

- `mysqldump -h 127.0.0.1 -u root -p root db_name>path;`
  - 使用`mysqldump` 命令备份数据库
  - `-h`指定数据库所在的服务器的ip地址
  - `-u`指定登录数据库的密码
  - `db_name`是要备份的数据库的名称
  - 使用输出目标操作符`>`,指定输出的文件具体路径`c:/back.sql`

## 备份表数据

```
1 | mysql -u root -p -e "selec 语句" dbname > 目标文件名
2 |
3 | select * from xxx into outfile '/tmp/stud.txt';
```

## 命令行还原

## 还原表结构和数据

```
1 | mysql -u root -p [dbname] < 目标文件
2 | mysql -h127.0.0.1 -uroot -proot db_name<back.sql
3 |
4 | load data infile '/tmp/stud.txt' into table students;
5 | source /backup/all_db_2013-09-08.sql
```

## 约束

查看约束 `SHOW CREATE TABLE table_name`

## 主键约束

- 主键约束最显著的特征是主键列中的值是不允许重复(唯一)的, 通过主键约束可强制表的实体完整性。当创建或更改表时可通过定义 `primary key` 约束来创建主键。一个表只能有一个 `primary key` 约束, 且 `primary key` 约束中的列不能接受 `NULL` 值。
- `alter table tab_name add constraint pk_name primary key (deptno);`

```
1  -- 创建表时
2  CREATE TABLE `table_name` (
3      `id` int PRIMARY KEY, -- 设置主键
4      `name` varchar(20)
5  );
6  CREATE TABLE `table_name` (
7      `id` int NOT NULL,
8      `name` varchar(20) ,
9      PRIMARY KEY (`id`) -- 设置主键
10 );
11 CREATE TABLE `table_name` (
12     `id` int NOT NULL,
13     `name` varchar(20) ,
14     constraint pk primary key(id) -- 设置主键
15 );
16
17 -- 设置主键是deptno
18 ALTER TABLE emp MODIFY empno INT PRIMARY KEY; -- 修改列的属性来添加主键约束
19 ALTER TABLE 表名称 ADD PRIMARY KEY(id);
20 ALTER TABLE dept ADD CONSTRAINT pk_name PRIMARY KEY(deptno);
21
22 -- 删除主键
23 ALTER TABLE 表名称 DROP PRIMARY KEY; -- 【推荐】
```

## 自增长列（标识列）

并不是所有表在设计完成后都能找到适合作为主键的列, 为此数据库提供了自增长列, 自增长列是数值类型( `INT` , `FLOAT` , `DOUBLE` 等)的, 其值是由数据库自动维护的, 是永远都不会重复的, 因此自增长是最适合作为主键列的。在创建表时, 通过 `auto_increment` 关键字来标识自增长列, 在MySQL数据库中自增长列可以是主键列, 也可以是唯一列（有唯一约束的列）。

特点:

1. 标识列必须和一个Key搭配（Key指主键、唯一、外键....）
2. 一个表最多有一个标识列
3. 标识列的类型只能是数值型



4. 标识列可以通过 `SET auto_increment_increment = 3;`, `SET @@auto_increment_increment = 3;` 设置步长（全局），可以通过插入行时手动插入标识列值设置起始值。（思考：什么情况下此设置会失效）

如果设置步长会从 1 + 步长 依次递增。

如果表中已有数据，会按照 1 + 步长进行计算，如果这个值已有则再加直到没有

## 自增长与主键

```
1 CREATE TABLE emp(  
2     empNo INT PRIMARY KEY AUTO_INCREMENT,  
3     job VARCHAR(10),  
4     mgr INT,  
5     sal DOUBLE,  
6     comm DOUBLE,  
7     deptno INT  
8 ) CHARACTER SET utf8;  
9  
10  
11 -- 设置自增长列的初始值  
12 CREATE TABLE temp(  
13     id INT PRIMARY KEY AUTO_INCREMENT,  
14     name VARCHAR(2)  
15 ) AUTO_INCREMENT=10;  
16  
17 -- 设置自增长列的初始值  
18 ALTER TABLE emp AUTO_INCREMENT = 10;  
19  
20 alter table 表名称 modify column id int auto_increment;  
21  
22 -- 删除自增长列  
23 ALTER TABLE 表名称 MODIFY COLUMN id INT;
```

## 唯一约束

对于非主键列中的值也要求唯一性时，就需要唯一约束

```
1 -- 创建表时  
2 CREATE TABLE `table_name` (  
3     `id` int NOT NULL,  
4     `name` varchar(20) UNIQUE, # 唯一约束  
5 );  
6 CREATE TABLE `table_name` (  
7     `id` int NOT NULL,  
8     `name` varchar(20) UNIQUE, # 唯一约束  
9 );
```

```

7   `id` int NOT NULL,
8   `name` varchar(20),
9   constraint uq_name unique(name),    #唯一约束
10 );
11 alter table 表名称 ADD unique(列名称);
12 ALTER TABLE dept ADD CONSTRAINT uq_name UNIQUE(deptname);
13
14 -- 删除唯一约束
15 ALTER TABLE tb_name DROP CONSTRAINT constraint_name;
16 [alter table 表名称 drop index 设置唯一时的名称;] -- 删索引

```

- 唯一约束要求值不能重复
- 可以存在多个空值( `NULL` )的数据
- 一张可以有多个唯一约束列吗
- 约束默认的名称为其列名
- 唯一约束创建后会自动创建一个唯一索引（索引后边会讲）

## 默认约束

为列中的值设置默认值，`default ...`,如果已经定了值，默认值就无效了

```

1  -- 创建表时
2  CREATE TABLE `table_name` (
3    `id` int DEFAULT NULL, # 默认约束
4    `name` varchar(20) unique,
5  );
6  alter table 表名称 modify column 列名 列类型 default 默认值;
7
8  -- 删除
9  alter table 表名称 modify column 列名 列类型; -- 将默认值改为 NULL
10 ALTER TABLE tb_name ALTER col_name DROP DEFAULT; -- 删除了默认值，新增时必须要有值

```

- 创建表时，不写默认值都默认 `NULL`（在无非空约束的情况下）
- 默认约束能和主键约束可以同时存在
- 默认约束不能和 `AUTO_INCREMENT` 同时使用

## 非空约束

**NOT NULL**：非空，用于保证该字段的值不能为空。例如学生表的学生姓名及学号等等

```

1  -- 创建表时
2  CREATE TABLE `table_name` (
3      `id` int NOT NULL, # 非空约束
4      `name` varchar(20),
5  );
6  alter table 表名称 modify column 列名 列类型 not null;
7
8  -- 删除
9  alter table 表名称 modify column 列名 列类型 [null];

```

## 外键约束

A表中列的值来自于另外一张表B的主键或唯一键的列称为 外键FK，将被引用值得表称为主表或父表，将引用值得表称为从表或子表。例如: `emp` 表中有 `deptno` 列，值来自于 `dept` 表的主键 `deptno`。`dept` 是主表，`emp` 是从表。

```

1  -- 创建表时
2  CREATE TABLE `dept` (
3      dept_no INT PRIMARY KEY,
4      dept_name VARCHAR(20),
5  )
6  CREATE TABLE `emp` (
7      `id` int NOT NULL,
8      `name` varchar(20),
9      `deptno` int,
10     CONSTRAINT fk_dept_no FOREIGN KEY(deptno) REFERENCES dept(dept_no)
11 );
12
13 alter table userinfo add constraint foreign key fk_dept_no (dept_no) REFERENCES
    dept(deptno);
14
15 -- 删除
16 ALTER TABLE tb_name DROP CONSTRAINT constraint_name;
17 alter table 表名称 drop foreign key 设置外键时的名称;

```

- `dept` 是主表，`userinfo` 是从表
- 在 `userinfo` 表中添加或修改时，`dept_no` 列的值必须是 `dept` 表中 `deptno` 字段中的存在值或者 `NULL`
- 删除从表数据可以直接删除
- 删除主表数据时，会先检查被删除数据在从表中有没有对此数据的关联（引用），如果有不能直接删除。

如果想要解除此限制需要先禁用外键约束【不推荐】

我们可以在创建约束时，设置级联操作【具体如何操作？】

- `on delete CASCADE / on update CASCADE` 级联删除 / 级联更新
- `ON DELETE SET NULL / ON UPDATE SET NULL`

## 检查约束

`check`

## 高级查询

### distinct

在 `select` 语句中，可以使用 `distinct` 关键字对查询的结果集进行去重。去重必须结果集中每个列的值都相同。

```
1 | select distinct 列1, ... , 列n from table_name [其他子句];
```

### order by

`order by` 用于对结果进行排序显示，可以使用 `ASC` / `DESC` 两种方式进行排序，可以有多个排序条件

- `ASC`：表示升序排序，如果不写即为此排序方式
- `DESC`：表示降序排序

```
1 | select [distinct] 列1, ... , 列n from table_name [其他子句] order by 排序列1 [DESC], 排序列2 [DESC];
```

### 分页查询limit子句

```
1 | select * from emp limit 0,2;
```

- 第一个参数0是表示从第几条开始查询 (这里的 0 是可以省略不写的);
- 第二个参数 表示查询出几条数据
- 后面不够的, 有多少写多少;

```
1 | select * from emp order by empNo limit 5;
2 | select * from emp limit 5,5;
3 |
4 | --
5 | select * from table_name limit (页码 - 1) * 每页数量, 每页数量;
```

## 聚合函数

Mysql中内置了 5 种聚合函数, 分别是: `SUM`、`max`、`min`、`avg`、`count`。

- `sum`: 求和

```
1 | select sum(列) from table_name [其他子句];
```

- `max`: 求最大值

```
1 | select max(列) from table_name [其他子句];
```

- `min`: 求最小值

```
1 | select min(列) from table_name [其他子句];
```

- `avg`: 求平均值

```
1 | select avg(列) from table_name [其他子句];
```

- `count`: 求数量

```
1 | select count(列) from table_name [其他子句];
```

## group by

**group by** 是对数据进行分组，分组时，表中有相同值的分为一组。分组后可以聚合查询。

**group by** 分组后的查询中，**select** 的列不能出现除了 **group by** 分组条件以及聚合函数外的其他列。

```
1 | select 列1, 列2, (聚合函数) from table_name group by 列1, 列2;
```

## having

**having** 是对 **group by** 分组后的结果集进行筛选。

```
1 | select 列1, 列2, (聚合函数) from table_name group by 列1, 列2 having 分组后条件;
```

## 综合查询

```
1 | SELECT DISTINCT emp.deptno FROM emp JOIN dept ON emp.deptno = dept.deptno WHERE  
   | bridate >= '2000-01-01' GROUP BY emp.deptno HAVING count(*) >= 2 ORDER BY count(*)  
   | DESC LIMIT 0, 5;
```

书写顺序是以上。

**SQL** 语句的执行顺序

from --> on --> join --> where --> group by --> having --> select --> distinct --> order  
by --> limit

## sql语句定义和执行顺序

## 多表查询

### 笛卡尔乘积现象

表查询中的笛卡尔乘积现象：多行表在查询时，如果定义了无效连接或者漏写了连接条件，就会产生笛卡尔乘积现象，所谓的笛卡尔乘积即是每个表的每一行都和其他表的每一行组合。

### 笛卡尔乘积现象

```
1 | SELECT * FROM emp,dept;
```

## 等值连接查询

通常是在存在主键外键关联关系的表之间的连接进行，使用"="连接相关的表

n个表进行等值连接查询，最少需要n-1个等值条件来约束

```
1 | --查询每个部门的所有员工
2 | select dept.dname,emp.ename from emp,dept where dept.deptno = emp.deptno;
```

## 自连接查询

表查询不仅可以在多个表之间进行查询，也可以在一个表之中进行多表查询

```
1 | --查询当前公司员工和所属上级员工的信息
2 | select e1.empno as 员工编号,e1.ename as 员工姓名,e2.empno as 领导编号,e2.ename as 领导姓名
   | from emp as e1,emp as e2 where e1.mgr = e2.empno;
```

## 内连接查询

内连接查询使用 `inner join` 关键字实现，`inner` 可以省略。内连接查询时，条件用 `on` 连接，多个条件使用 `()` 将其括起来。

```
1 | --查询每个部门的所有员工
2 | select dept.name,emp.name from emp inner join dept on emp.deptno = dept.deptno;
```

和等值查询差不多

## 外连接

外连接分为左外连接（`left outer join`）和右外连接（`right outer join`）其值 `outer` 可以省略。外连接查询时，条件用 `on` 连接，多个条件使用 `()` 将其括起来。

左外连接表示以左表为主表，右外连接表示以右表为主表。查询时将主表信息在从表中进行匹配

```

1  --查询每个部门的所有员工
2  select dept.name,emp.name from emp right join dept on emp.deptno = dept.deptno;
3
4  select dept.name,emp.name from emp left join dept on emp.deptno = dept.deptno;

```

## 子查询

### 单行子查询

#### 单行单列

```

1  --查询软件部门下的所有员工
2  select * from emp e where e.deptno = (select d.deptno from dept d where d.dname = '软件部');

```

### 多行子查询

如果子查询返回了多行记录，则称这样的嵌套查询为多行子查询，多行子查询就需要用到多行记录的操作符

如： `in` , `all` , `any`

```

1  --统计所有的员工分布在哪些部门的信息
2  select * from dept d where d.deptno in (select e.deptno from emp e);
3
4
5  --查询公司中比任意一个员工的工资高的所有员工
6  select * from emp e1 where e1.sal > any (select e1.sal from emp e2);
7
8  --查询公司中比所有的助理工资高但不是助理的员工
9  select * from emp e1 where e1.sal > all(select e2.sal from emp e2 where e2.joblike '%助理');

```

- `>any` 表示大于子查询中的任意一个值，即大于最小值
- `>all` 表示大于子查询中的所有值，即大于最大的值

### 多列子查询

#### 单行多列

```

1  --查询公司中和员工***相同薪水和奖金的员工
2  select * from emp e1 where (e1.sal,e1.comm) = (select e2.sal,e2.comm from emp e2 where e2.ename = '张青');

```



## 子查询[应用]

存在于另外一个SQL语句中、被小括号包起来的查询语句就是子查询。相对于子查询来说，在外部直接执行的查询语句被称作主查询

子查询分为：

- 单列子查询: 返回单行单列数据的子查询
- 单行子查询: 返回单行多列数据的子查询
- 多行子查询: 返回数据是多行单列的数据
- 关联子查询: 子查询中如果使用了外部主SQL中的表或列，就说这个子查询跟外部SQL是相关的

### 单列子查询

导入 `scott`用户表`sql.sql`

查看 `emp` 表中与 `SMITH` 岗位相同的员工信息

1. 先尝试查看 `SMITH` 所从事的岗位
2. 查询从事 `clerk` 工作的员工

### 单行子查询

查询 与 `SMITH` 在同一个部门且岗位相同的员工的信息

1. 查询 `SMITH` 所在的部门和从事的岗位
2. 查询在 `20` 部门从事 `clerk` 岗位的员工信息
3. 可以成对比较，也可以把子查询当成一张虚拟表使用

### 多行子查询

查询`emp`表中与`20`部门员工岗位相同的员工信息

1. 查询`20`部门的所有岗位

- 2. 剔除重复行
- 3. 根据20部门的岗位来查询emp表中的员工

关联子查询

查询哪些员工与SMITH不在同一个部门

```
1 | SELECT * FROM emp e
2 | WHERE EXISTS ( SELECT * FROM emp p WHERE p.ename = 'SMITH' AND p.deptno != e.deptno
   | );
```

单行函数

菜鸟教程-MySQL函数

字符串函数

SELECT XXX FORM dual

函数	示例	结果	描述
upper	select upper('sdfd');	SDFD	将字母转换为大写
lower	select lower('ABC');	abc	将字母转换为小写
concat	select concat('hello', 'world');	'hello world'	字符串连接
substr/substring	select substr('hello world', 2); select substr('abcd', 2, 3)	'ello world'	截取字符串
length	select length('hello world');	11	获取字符串长度
instr	select instr('hello world', 'world');	7	获取子字符串在父字符串中的索引

函数	示例	结果	描述
trim	select trim(' hello ');	'hello'	去除两端空格
ltrim	select ltrim(' hello');	'hello'	去掉左端的空格
rtrim	select rtrim('hello ');	'hello'	去掉右端的空格
replace	select replace('hello java','java','world');	'hello world'	替换文本

注：这里的字符串是从1开始的（不是0）

数学函数

函数	示例	结果	作用
round(x,[y])	select round(5.64,1);	5.6	对指定的值进行四舍五入是可以指定数值位数y
truncate(x,y)	select truncate(5.6,0);	5	对指定的数及进行截取操作，指定保留位数y
ceil(x)	select ceil(4.56)	5	返回不小于指定的值x得最小整数，向上取整
floor(x)	select floor(8.5);	8	返回不大于指定的值x的最大整数，向下取整
abs(x)	select abs(-12);	12	取绝对值
MOD(x,y)	SELECT MOD(5, 2) FROM dual	1	返回 x 除以 y 以后的余数
PI()	SELECT PI() FROM dual	3.141 593	返回圆周率
POW(x,y)/POWER(x,y)	SELECT POW(2,4)	16	返回 x 的 y 次方
SQRT(x)	SELECT SQRT(4)	2	平方根

函数	示例	结果	作用
current_timestamp()	<pre>select current_timestamp( );</pre>	2019-11-07 20:53:47	获取当前时间戳
current_date()/CURDATE()	<pre>select current_date();</pre>	2019-11-07	获取当前日期
current_time()/CURTIME()	<pre>select current_time();</pre>	20:56:00	获取当前时间
now	<pre>select now();</pre>	2019-11-07 20:57:15	获取当前时间+日期
ADDDATE(d,n)	<pre>SELECT ADDDATE("2017-06-15", INTERVAL 10 DAY);</pre>	2017-06-25	计算起始日期 d 加上 n 天的日期
ADDTIME(t,n)	<pre>SELECT ADDTIME('2011-11-11 11:11:11', 5);</pre>	2011-11-11 11:11:16	n 是一个时间表达式，时间 t 加上时间表达式 n
DATEDIFF(d1,d2)	<pre>SELECT DATEDIFF('2001-01-01','2001-02-02')</pre>	-32	计算日期 d1- d2 之间相隔的天数
DATE_ADD(d, INTERVAL expr type)	<pre>SELECT DATE_ADD("2017-06-15", INTERVAL 10 DAY);</pre>	2017-06-25	计算起始日期 d 加上一个时间段后的日期。type 值

函数	示例	结果	作用
DATE_SUB(date, INTERVAL expr type)	SELECT DATE_SUB("2017-06-15", INTERVAL 10 DAY);	2017-06-05	计算起始日期 d 减上一个时间段后的日期。type 值
DATE_FORMAT(d, f)	SELECT DATE_FORMAT('2011-11-11 11:11:11', '%Y-%m-%d %r')	2011-11-11 11:11:11 AM	按表达式 f 的要求显示日期 d
DAY(d)	SELECT DAY("2017-06-15")	15	返回日期值 d 的日期部分
DAYNAME(d)	SELECT DAYNAME('2011-11-11 11:11:11')	Friday	返回日期 d 是星期几，如 Monday, Tuesday
DAYOFMONTH(d)	SELECT DAYOFMONTH('2011-11-11 11:11:11')	11	计算日期 d 是本月的第几天
DAYOFWEEK(d)	SELECT DAYOFWEEK('2011-11-11 11:11:11')	6	日期 d 今天是星期几，1 星期日，2 星期一，以此类推
DAYOFYEAR(d)	SELECT DAYOFYEAR('2011-11-11 11:11:11')	315	计算日期 d 是本年的第几天
EXTRACT(type FROM d)	SELECT EXTRACT(MINUTE FROM '2011-11-11 11:11:11')	11	从日期 d 中获取指定的值，type 指定返回的值。
FROM_DAYS(n)	SELECT FROM_DAYS(1111)	0003-01-16	计算从 0000 年 1 月 1 日开始 n 天后的日期
HOUR(t)	SELECT HOUR('1:2:3')	1	返回 t 中的小时值

函数	示例	结果	作用
LAST_DAY(d)	SELECT LAST_DAY("2017-06-20");	2017-06-30	返回给给定日期的那一月份的最后一天
MAKEDATE(year, day-of-year)	SELECT MAKEDATE(2017, 3);	2017-01-03	基于给定参数年份 year 和所在年中的天数序号 day-of-year 返回一个日期
MINUTE(t)	SELECT MINUTE('1:2:3')	2	返回 t 中的分钟值
MONTHNAME(d)	SELECT MONTHNAME('2011-11-11 11:11:11')	November	返回日期当中的月份名称, 如 November
MONTH(d)	SELECT MONTH('2011-11-11 11:11:11')	11	返回日期d中的月份值, 1 到 12
QUARTER(d)	SELECT QUARTER('2011-11-11 11:11:11')	4	返回日期d是第几季节, 返回 1 到 4
SECOND(t)	SELECT SECOND('1:2:3')	3	返回 t 中的秒钟值
SUBDATE(d, n)	SELECT SUBDATE('2011-11-11 11:11:11', 1)	2011-11-10 11:11:11	日期 d 减去 n 天后的日期
WEEK(d)	SELECT WEEK('2011-11-11 11:11:11')	45	计算日期 d 是本年的第几个星期, 范围是 0 到 53
WEEKDAY(d)	SELECT WEEKDAY("2017-06-15");	3	日期 d 是星期几, 0 表示星期一, 1 表示星期二
WEEKOFYEAR(d)	SELECT WEEKOFYEAR('2011-11-11 11:11:11')	45	计算日期 d 是本年的第几个星期, 范围是 0 到 53

函数	示例	结果	作用
YEARWEEK(date, mode)	SELECT YEARWEEK("2017-06-15");	201724	返回年份及第几周（0到53），mode 中 0 表示周天，1表示周一，以此类推

**type** 值可以是：

- MICROSECOND 微秒
- SECOND 秒
- MINUTE 分钟
- HOUR 小时
- DAY 天
- WEEK 周
- MONTH 月
- QUARTER 季度
- YEAR 年
- MINUTE\_SECOND 分钟:秒
- HOUR\_SECOND 小时:分钟 : 秒
- HOUR\_MINUTE 小时:分钟
- DAY\_SECOND 天 小时:分钟:秒
- DAY\_MINUTE 天 小时:分钟
- DAY\_HOUR 天 小时
- YEAR\_MONTH 年-月

可以被用在**format**字符串：

- %M 月名字(January.....December)
- %W 星期名字(Sunday.....Saturday)
- %D 有英语前缀的月份的日期(1st, 2nd, 3rd, 等等。)
- %Y 年, 数字, 4 位
- %y 年, 数字, 2 位
- %a 缩写的星期名字(Sun.....Sat)
- %d 月份中的天数, 数字(00.....31)
- %e 月份中的天数, 数字(0.....31)
- %m 月, 数字(01.....12)

- %c 月, 数字(1.....12)
- %b 缩写的月份名字(Jan.....Dec)
- %j 一年中的天数(001.....366)
- %H 小时(00.....23)
- %k 小时(0.....23)
- %h 小时(01.....12)
- %I 小时(01.....12)
- %l 小时(1.....12)
- %i 分钟, 数字(00.....59)
- %r 时间,12 小时(hh:mm:ss [AP]M)
- %T 时间,24 小时(hh:mm:ss)
- %S 秒(00.....59)
- %s 秒(00.....59)
- %p AM或PM
- %w 一个星期中的天数(0=Sunday .....6=Saturday )
- %U 星期(0.....52), 这里星期天是星期的第一天
- %u 星期(0.....52), 这里星期一是星期的第一天

## 设计数据库步骤

软件开发的步骤大致上可以分为：

需求分析，概要设计，详细设计，代码编写，运行测试，部署发行

数据库是在代码编写前完成的

数据库设计可分为这几个过程



## 创建概念模型

P53~p60

## 数据库设计范式

数据库的设计有五大设计范式。常用的有三大设计范式，称之为第一范式( 1NF ), 第二范式( 2NF ), 第三范式( 3NF ), 他们是逐步为严格的，满足第二范式，就必须满足先满足第一范式。满足第三范式时就必须首先满足第二范式

### 第一范式

第一范式要求单个表中每个列必须是原子列（即每一个列都是不可再分的最小数据单元），列不存在重复属性，每个实体的属性也不存在多个数据项。

- 1.原子列
- 2.不出现重复属性
- 3.不允许出现多个数据项

### 第二范式

第二范式是在满足第一范式的基础之上，要求数据表里的所有数据都要和该数据表的主键有完全依赖关系。

### 第三范式

第三范式是在满足第二范式的基础之上，每一个非主键列都直接依赖主键列，不依赖其他非主键列，即数据库中不能存在传递函数的依赖关系。

优点：

1. 范式化的数据库更新起来更加的快；
2. 范式化之后只有很少的重复数据，只需要修改更少的数据；
3. 范式化的表更小，可以在内存中直接执行；
4. 很少的冗余数据，在查询时候需要更少的distinct后者group by语句。

缺点：

1. 范式化的设计会产生更多的表；
2. 在查询的时候经常需要很多的表连接查询，到值查询性能降低；