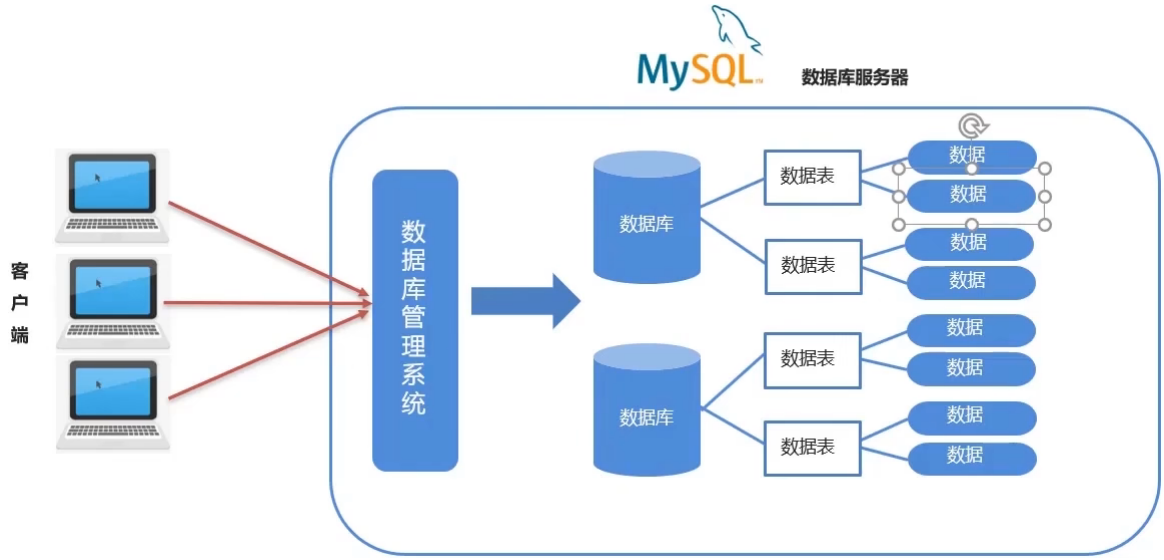
1.数据库概述

MySQL图解应用：



1.1 数据库相关概念

数据库：DataBase (DB)

- 按照一定格式存储数据的一些文件的组合
- 存储数据的仓库，数据文件，日志文件。具有一定特定格式的数据。

数据库管理系统：DataBase Management （DBMS)

- 专门用来管理数据库中的数据的，数据库管理系统可以对数据当中的数据进行增删改查

SQL： 结构化查询语言

- 使用DBMS负责执行SQL语句，来完成数据库中的增删改查。
- SQL是一套标准语言，主要学习SQL语句。SQL语句可以在MySQL、Oracle、DB2中使用。

三者之间的关系：

> DBMS ----执行-----> SQL -------操作----> DB

1.2登录MySQL：

mysql修改登录用户名和秘密

```
# 修改密码

-- 1.进入cmd
-- 2. 输入
mysql -u root -p;
-- 3.输入旧密码
Enter password: *****
-- 4.选择数据库
mysql> use 库名;
-- 输入修改语句信息
```

```
mysql> UPDATE user SET password =PASSWORD("新密码") WHERE user = '用户名';

-- 刷新
mysql> flush privileges;


#修改用户名:
-- 1.进入cmd
-- 2. 输入
mysql -u root -p;
-- 3.输入旧密码
Enter password: *****
-- 4.选择数据库
mysql> use 库名;
-- 输入修改语句信息
mysql> UPDATE user SET u =user("新用户名") WHERE user = 'root';
mysql> flush privileges;
mysql> exit
```

1.**在DOS窗口下：（隐藏密码形式）**

- 打开mysql：net start mysql
- 关闭mysql：net stop mysql
- 进入mysql指令：mysql -u root -p
- 关闭mysql指令：exit（退出）

2.**一步到位登录MySQL指令：（显示密码形式）**

- mysql -uroot -p123456　　// -p 后面的是MySQL 的密码

1.3MySQL常用命令

1. 查看数据库

```
show Database;
```

2. 使用数据库

```
use test;
```

use 数据库名；

3. 创建数据库

```
create database test1
```

create database 数据库名

4. 查看表

```
show tables;
```

5. sql中导入数据 （执行sql脚本）

```
source  全路径（指定sql脚本的路径）
```

路径不能出现中文！！！

6. 查看MySQL版本

```
select version();
```

7. 查看当前使用的数据库

```
select database();
```

1.4表：

数据库中的最基本单元：表

表结构：

- 行（row）：数据 / 记录
- 列（column）：字段 （每一个字段都有：字段名，数据类型，约束等属性）

- 字段名：
- 数据类型：字符型、日期型、时间型、整型等
- 约束：

1.5SQL语句分类：

1. DDL： **数据定义语言**（Data Definition Language）
    - 对**表结构**进行操作
    - create：创建
    - alter：修改
    - drop：删除
2. DML： **数据操作语言**（Data Manipulation Language）
    - 对**表当中的数据**进行**增删改查**，操作表中的数据data
    - inster：插入
    - delete：删除
    - update：修改
3. DQL： **数据查询语言**（Data Query Language）
    - select 语句
4. DCL： **数据控制语言**（Data Control Language）
    - 授权：GRANT
    - 撤销权限：REVOKE
5. TCL ：事务控制语言
    - 事务提交：commit
    - 事务回滚：rollback

2.CRUD操作

2.1 DQL

- SQL语句是通用的，以英文分号 " ; " 结束
- 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
    1. 单行注释：-- 注释内容或 # 注释内容(MySQL特有)
    2. 多行注释：/* 注释 */

1.基础查询

1. 基础查询（简单查询）

- 查询所有列 (字段)

```
SELECT * FROM 表名;     -- 查询所有字段   （效率低、可读性差、开发中不建议）
```

- 查询单个/多个字段

```
SELECT
  字段1，字段2,···,字段n
FROM
  表名;     -- 查询单个/多个字段
```

- 去重查询

```
SELECT DISTINCT  去重的字段1 FROM 表名;
SELECT DISTINCT  name FROM student;  -- 去重复的姓名
```

- 为字段起别名

```
SELECT  字段1 AS 别名名称 FROM 表名;  -- 方式一
SELECT 新字段名 = 字段1  FROM 表名;  -- 方式二
-- 列如：
SELECT name AS 姓名 FROM student; -- 方式一
SELECT name 姓名 FROM student; -- 方式二
SELECT 姓名 = name  FROM studnet;  -- 方式三
```

别名只是在显示中，并不会修改到表中的字段名

别名中间存在有空格，会出现报错，不符合语法，编译错误。

若是字段中间必须要有空格必须使用单引号或者双引号括起来

```
SELECT name  AS  '姓 名' FROM student;  -- 标准字符串，单引号括起来
SELECT name  AS  "姓 名" FROM student;  -- 不推荐，不标准
```

注意：

1. 单引号是标准的字符串形式
2. 双引号在mysql中可以使用，在oracle中不可以使用

- 字段可以使用**数学表达式**
  - 给字段进行加减乘除

2. 条件查询：

查询出符合条件的内容

- 条件查询语法

```
SELECT
  字段1,字段2,...,字段n,
FROM
  表名
WHERE
  条件;
```

- 条件符号：

符号	说明
=	等于
<>或!=	不等于
<	小于
<=	小于等于
>	大于
>=	大于等于
between …… and	两个值之间，当同于>= and <= 遵循左小右大
is null	为空（is not null 不能为空）
and	并且
or	或者
in	包含，相当于多个 or （not in：不包括在内）
like	模糊查询，支持%或下划线匹配
%	通配符：匹配任意个字符
_	下划线：匹配一个字符

```
SELECT name ,age FROM student WHERE age = 55 ; -- 等于

SELECT name ,age FROM student WHERE age != 55;  -- 不等于
SELECT name ,age FROM student WHERE age <> 55;

SELECT name ,age FROM student WHERE age < 55; -- 小于
SELECT name ,age FROM student WHERE age <= 55;  -- 小于等于
```

```sql

SELECT name ,age FROM student WHERE age > 55 ;  -- 大于
SELECT name ,age FROM student WHERE age >= 55;  -- 大于等于

SELECT name ,age FROM student WHERE age BETWEEN 23 AND 30;  -- 年龄在23到30之间
SELECT name ,age FROM student WHERE age >= 23 AND <= 30 ;

SELECT name ,age FROM student WHERE age IS NULL;  -- is null  is不能改为等号
SELECT name ,age FROM student WHERE age IS NOT NULL;  -- is not null is不能改为等号

SELECT name ,age FROM student WHERE name = '张三' AND age > 45;   -- end

SELECT name ,age FROM student WHERE name = '张三' OR  name = '小三';  -- OR 有一个就可以查找到

SELECT name ,age FROM student WHERE age > 15 AND name = '张三' OR name = '小三'; -- and 和 or 同时出现这一句中：先执行 and 在执行 or
SELECT name ,age FROM student WHERE age > 15 AND (name = '张三' OR name = '小三');  -- 正确的写法  让 or 先执行

SELECT name ,age FROM student WHERE age IN(15,35,45,25);  -- 年龄在15、25、35、45中的
SELECT name ,age FROM student WHERE age NOT IN(15,35,45,25); -- 年龄不在15、25、35、45中的name 和 age
```

- 模糊查询：LIKE

```sql
SELECT name ,age FROM student WHERE LIKE '%小%' ;-- 含有‘小’的名字
SELECT name ,age FROM student WHERE LIKE '张%' ;-- 姓张的
SELECT name ,age FROM student WHERE LIKE '%森' ;-- 以森结尾的
SELECT name ,age FROM student WHERE LIKE '%_垂%'; -- 第二个字为垂的
SELECT name ,age FROM student WHERE LIKE '%__森%'; -- 第二个字为森的

SELECT name ,age FROM student WHERE LIKE '%_%' ;-- 找出名字中含有下划线的。 因为下划线具有特殊含义，需要把下划线进行转义
SELECT name ,age FROM student WHERE LIKE '%\_%'; -- 正确 √
```

3. 排序查询：

排序总是在最后执行！！！

- 语法

```sql
SELECT
  字段1,字段2,···,字段n,
FROM
  表名
ORDER BY
  排序字段名1 [排序方式1], 排序字段名2 [排序方式2]  ...;
```

- 排序方式
  - ASC： 升序 （默认值），不写就是默认升序
  - DEAC： 降序
- 多个字段排序： 如果有多个排序条件，当前边的条件值一样时，才会根据第二条件进行排序

```
SELECT name ,age FROM student   ORDER BY age ; -- 按照年龄 默认升序
SELECT name ,age FROM student   ORDER BY age DESC;  -- 降序
SELECT name ,age FROM student   ORDER BY age ASC ; -- 指定升序

 -- 多个字段排序 ：如果有多个排序条件，当前边的条件值一样时，才会根据第二条件进行排序
SELECT name ,age,score FROM student ORDER BY age DESC,name DESC ;  -- 若是年龄相同，才能进行name排序

-- 根据字段的位置进行排序
SELECT name ,age FROM student   ORDER BY 2; -- 根据第二列排序
```

综合案例：查询年龄在20到50之间，并且更具年龄进行降序排序

```
SELECT
  name,age
FROM
  student
WHERE
  age >= 20 AND age <= 50
ORDER BY
  age DESC ;
```

4. 分组查询：

1. 分组函数

   分组函数在使用前必须进行分组，若是没有分组，则一个表就是一个组

   - 语法：

     ```
     SELECT
       分组函数名（列名）
     FROM
       表;
     ```

     - count： 计数
     - sum： 求和
     - avg： 平均值
     - max： 最大值
     - min： 最小值
   - 注意：

     1. 自动忽略NULL
     2. count(*) 和count(具体字段)的区别
        - count(*)： 统计总行数，不忽略null；
        - count(具体字段)：表示统计该字段下所有**不为NULL的元素的总数。**忽略null

3. 分组函数不能直接使用在where子句中。

4. 所有的分组函数可以组合在一起用。

2. 分组查询语法

- 先进行分组，对每一组的数据进行操作。

```
SELECT
  字段列表
FROM
  表名
[WHERE
  分组前条件限定]
GROUP BY
  分组字段名
[HAVING
  分组后条件过滤];
```

- where不参与分组，having是在分组之后的，where不能对聚合函数进行判断，having可以。
- **关键字执行顺序：from --> where --> group by --> having--> select -->order by**

```
SELECT COUNT(name) FROM student ;  -- 可以执行，因为SELECT在分组后执行
SELECT name ,age FROM student WHERE age > MIN(age) -- 不能执行 因为进行WHERE时 还没有进行分组
```

3. 案例

```
-- 1.找出每个工作岗位的工资和？
    -- 思路：按照工作岗位分组，然后对工资求和
SELECT
  job,SUM(sal)
FROM
  emp
GROUP BY
  job;

-- 2.找出 '每个部门，不同工作岗位'的最高薪资
  -- 两个字段进行分组
SELECT
  deptID,job,MAX(sal)
FROM
  emp
GROUP BY
  deptID,job ;

-- 3.找出部门的最高薪资，要求显示最高薪资大于3000的
(方法一)
SELECT
  deptID,MAX(sal)
FROM
  emp
GROUP BY
```

```
  deptID
HAVING
  MAX(sal) > 3000;
(方法二)
SELECT
  deptID,MAX(sal)
FROM
  emp
WHERE
  sal > 3000
GROUP BY
  deptID ;
-- 优化策略：where 和 having 优先选择where，where实现不了，再选择having。
-- having的执行效率低

-- 4.找出每个部门平均薪资，要求显示平均薪资高于2500的
  -- 这个就不能使用where
（错误的写法，where子句中不能进行写分组函数）
SELECT
  deptID,AVG(sal)
FROM
  emp
where
  avg(sal) >2500
GROUP BY
  deptID ;
(正确写法)
SELECT
  deptID,AVG(sal)
FROM
  emp
GROUP BY
  deptID
HAVING
  AVG(sal) > 2500;

-- 5.找出每个岗位的平均薪资，要求显示平均薪资大于1500的，出去mag岗位之外。
SELECT
  job,AVG(sal) avgsal
FROM
  emp
WHERE
  job!= 'mag'
GROUP BY
  job
HAVING
  AVG(sal) > 1500
ORDER BY
  avgsal DESC; -- jiang'x
```

- SELECT 语句中，含有GROUP BY 语句，SELECT后面只能跟：参与分组，以及分组函数的字段，其他字段不可以写上去 添加其他的字段在oracle中会报错
- HAVING 子句不能单独使用，必须和GROUP BY 一起使用。
- HAVING：对分组之后的进行条件刷选。

5. **分页查询：**

   - 语法：SELECT 字段名 FROM 表名 起始索引 查询数目的条数
   - -- 起始索引公式：当前页的起始索引 = （当前页码 - 1） *每页显示的条数

     分页查询limit是MySQL数据库的方言

     Oracle分页查询使用rownumber

     SQL Server分页查询使用top

6. 去重查询

   - 关键字

```
select distinct name from student;
```

   - distinct 只能出现在所有查询字段的最前方
   - 可以使用分组函数

2 链接查询

- 多表查询

链接方式分类

1. 内连接：

   完全能匹配上这个条件的数据查询出来

   多个表之间的关系的平等的关系

   1. 等值链接

```
-- 查询每个员工所在部门名称，显示员工名和部门名
-- SQL92  :结构不够清晰，
select
  s.sname,y.yname
from
  student s , yanjiu y
where
  s.sno = y.sno;

-- SQL99  ：表链接的条件是独立的，连接之后还可以继续添加条件  inner可以省略
select
  s.sname,y.yname
from
  student s
[inner] join
  yanjiu y
on
  s.sno = y.sno;
```

   2. 非等值链接

      条件不是等量关系

```sql
-- 工资介于salgrade中的最低和最高之间 用到两个表
select
  e.ename,e.sal,s.grade
from
  emp e
join
  salgrade s
on
  e.sal between s.losal and s.hisal

```

3. 自链接：一个表写成两个表

```sql
-- 查询员工的上级领导，要求显示员工名和对应的领导名
select
  a.ename  '员工名', b.ename '领导名'
from
  emp a
join
  emp b
on
  a.mgr = b.empno;
```

2. 外连接

多个表之间有主次的关系，主表和副表

主表就是要全部查询出来的，副表的符合条件才会查询出来

1. 左链接

```sql
-- 左外查询  左边的是主表
select
  s.sname,y.yname
from
  student s left join yanjiu y
on
  s.sno = y.sno;

-- 查询员工的上级领导，要求显示员工名和对应的领导名
select
  a.ename  '员工名', b.ename '领导名'
from
  emp a
left join
  emp b
on
  a.mgr = b.empno;
```

2. 右链接

```sql
-- 右外查询  右边的是主表  outer可以省略
select
  s.sname,y.yname
from
  student s
right [outer] join
  yanjiu y
on
  s.sno = y.sno;

```

左右链接之间可以相互转换

外连接的查询结果条数一定是 >= 内连接的查询结果条数

3. 全连接

笛卡尔积现象：当两张表进行链接查询，没有任何条件查询的时候，最终查询结果条数，是两张表条数的乘积

避免笛卡尔积现象：连接时加条件，满足这个条件的记录就筛选出来！

```sql
select
  sname,yname
from
  student,yanjiu
where
  student.sno = yanjiu.sno


-- 起别名
select
  s.sname,y.name
from
  student s,yanjiu y
where
  s.sno = y.sno

-- 减少链接次数


```

3 子查询

> SELECT语句中嵌套子查询，被嵌套的SELECT语句为子查询

where子查询

```sql
-- 找出比最低工资高的员工姓名和工资
SELECT
  ename, sal
FROM
  emp
WHERE
  sal >
  (SELECT MIN(sal) FROM emp);
```

from子句中的子查询

- from后面的子查询，可以将子查询的查询结果当作一个临时表

```sql
-- 找出每个岗位的平均工资的薪资等级
select
  t.*,s.grade
from
  (select job,avg(sal)  avgsal  from emp group by job) as t
join
  salgrade s
on
  t.avgsal between s.losal and s.hisal
```

select后面的子查询

- 返回的结果只能有一条数据，多余一条数据就会报错

4 union合并查询结果集

```sql
SELECT name ,age FROM student WHERE name = '张三'
UNION
SELECT name ,age FROM student WHERE name = '小三';
```

- 效率高，链接一次新表，匹配次数就会翻倍，union可以减少匹配次数，还可以将结果集拼接起来
- union在进行结果集的合并时，要求两个结果集的列数相同

5 limit 查询

- 将查询结果集的一部分取出来，使用在分页查询中

使用方法：

1. 完整写法： LIMIT ： startIndex（从0开始） ， length（长度）

```sql
SELECT
  ename, sal
FROM
  emp
ORDER BY
  sal desc
LIMIT
  5 ; -- 取前五条
```

```sql

SELECT
  ename, sal
FROM
  emp
ORDER BY
  sal desc
LIMIT
  1,5 ; -- 前六：1-6
```

通用分页

> 每页显示 6页条记录
>
> 第1页：limit 0，6 【0、1、2、3、4、5】
>
> 第2页：limit 6，6 【6 7 8 9 10 11】
>
> 第3页：limit 12，6
>
> 第n页：limit (n-1) * 6，6
>
> 开始的index = pageSize * （n - 1）

```sql
limit (pageNO - 1) * pageSize , pageSize
```

2.2 DDL

- 对表的结构进行操作

2.2.1 create：表的创建

1. 语法：

```sql
CREATE TABLE 表名
(
字段名1 数据类型,
字段名1 数据类型,
字段名1 数据类型,
字段名n 数据类型
);
```

表名规范：

```sql
以   t_table1  或者  tbl_ 开始  -- 可读性高
```

创建表实列：

```sql
-- 建立学生表
CREATE TABLE t_student
(
  sno int(10),
    name varchar(22),
    age  int(3),
```

```
    sex char(5),
    email varchar(255)
);

-- 建立学生表，指定默认值
CREATE TABLE t_student
(
  sno int(10),
    name varchar(22),
    age  int(3) default '女',
    sex char(5),
    email varchar(255)
);
```

2. 数据类型

- 数据类型就是属性，有一些常用的，例如：整数数据，字符数据，日期数据，货币数据等

数字类型

数据类型	范围	占用的字节
bigint	-2^{63} ~2^{63} -1	8字节
int	-2^{31}~2^{31}-1	4字节 （11字符）
smallint	- 2^{15} ~ 2^{15}-1	2字节
tinyint	0~255	1字节
float	-1.79E+308~3.40E+38	4或者8字节

时间类型

数据类型	输出
time	12：35：29.123 （精确度到秒后面三位）时分秒
date	2007-05-08 （年月日）短日期
smalldatetime	2007-05-08 12:35:00
datetime	2007-05-08 12:35:29.123(精确到后面 就是小数点后面三位) 长日期
datetime	2007-05-08 12:35:29.1234567(精确到小数点后面三七位)

字符串类型

类型	说明
char[(n)]	固定长度（255长度） 。 n用于定义字符串长度，**必须**在1~8000之间。

类型	说明
varchar[(n|max)]	可变长度。n用于定义字符串长度，**可以**在1~8000之间。
nchar[(n)]	固定长度的Unicode字符串数据。n用于定义字符串长度，**必须**在1~4000之间。
nvarchar	可变长度的Unicode字符串数据。n用于定义字符串长度，**必须**在1~4000之间。
clob	字符大对象。最多可以存储4G的字符串（超过255个字符，使用它）
blob	二进制大对象。存储图片、声音、视频等媒体数据（**插入数据时，必须使用IO流**）

2 .2.2 alter：修改表

修改数据表：

- 修改表名：ALTER TABLE 表名 RENAME TO 新的表名;
- 修改列名和数据类型：ALTER TABLE 表名 CHANGE 列名 新列名 新数据类型;
- 添加一列：ALTER TABLE 表名 ADD 列名 数据类型;
- 删除列：ALTER TABLE 表名 DROP 列名
- 修改数据类型：ALTER TABLE 表名 MODIFY 列名 新数据类型;

2.2.3 drop：删除表

语法：

```
DROP TABLE 表名
DROP TABLE IF EXISTS  表名  -- 存在就删除
```

快速删除表

1. 语法

   ```
   truncate table 表名;
   ```

   - 特点：物理删除，删除效率高，不可以恢复
   - 删除数据，表结构保留
   - 不能删除单条数据

删除数据表：

- 删除表：DROP TABLE 数据表名;
- drop 表TABLE if exists 表名;

2.3DML

- 对表中的数据进行修改

2.3.1插入 insert

1. 语法

```
insert into 表名(字段名1,字段名2,字段名3,...) values(值1,值2,值3,...)
```

   - 字段名和值要一一对应：数量对应，数据类型要对应。

   插入实列：

```
insert into t_student(sno,name,age,sex,eamil) values(1,'张三',15,'女','1234567@qq.com')  -- 按照顺序

insert into t_student(name,sno,age,sex,eamil) values('张三',1,15,'女','1234567@qq.com')  -- 字段顺序可以调整，但是字段名和值一定要对应

insert into t_student(sno,name) values('张4',2) -- 只要某些字段,插入数据后其他的字段就会是null

insert into t_student() values(1,'李4',12,'女','123456744@qq.com') -- 省略字段名，就必须写上所有的字段名
```

2. 插入多条记录

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

3. **insert插入日期**

   格式化：format()

4. 将查询结果插入到一张表中（少用）

```
insert into 表名 select * from 表名 ;

select * into 新表名 from 表名;
```

2.3.2修改 Update

1. 语法格式

```
update
  表名
set
  字段名1 = 值1,
  字段名2 = 值2,
  字段名3 = 值3,
  字段名n = 值n
where
  条件;
```

   - 没有条件将会将整个表更改

2.3.3删除 delete

- 表中的数据被删除了，但是数据在硬盘上的真实存储空间不会被释放
- 删除效率低，可回滚

1. 语法格式

```
delete
from
  表名
where
  条件
```

   - 没有条件将会把整个表内的数据进行删除
2. 删除大表（删除表中的全部数据）

```

```

2.4 表

2.4.1 表结构的增删改查

- 一旦设计好表，就很少会对表进行修改
- 修改成本高，就要对java代码进行修改
- 使用工具（navicat）修改

2.4.2约束 （constraint）

- 给字段添加约束，保证表中数据的完整性，有效性！！！
- 保证表中的数据有效
- **列约束：直接在列后面添加约束**
- **表级约束：约束没有添加在列的后面**

1. **非空约束：not null**
2. **唯一约束：unique**
3. **主键约束：primary ley（PK）**
4. **外键约束：foreign key（FK）**

5. 级联更新
6. 检查约束：check（mysql不支持）

1 非空约束 （not null）

- 约束的字段不嫩改为null

```
CREATE TABLE t_student
(
  sno int(10) not null,
    name varchar(22) not null,
    age  int(3),
    sex char(5),
    email varchar(255)
);
```

2 唯一性约束 （unique）

- 可以为null， 但是不可以重复

```
drop table if exists t_student
CREATE TABLE t_student
(
  sno int(10) unique,  -- 列级约束
    name varchar(22),
    age  int(3),
    sex char(5),
    email varchar(255)
);

insert into t_student() values(1,'李4',12,'女','123456744@qq.com')
insert into t_student() values(2,'李4',12,'女','123456744@qq.com')
-- 在插入数据的时候，sno的值，则不能重复
```

- 多个字段字段联合起来具有唯一性

```
CREATE TABLE t_student
(
  sno int(10),
    name varchar(22),
    unique(sno,name)  -- sno ,name 联合起来唯一。表
);
insert into t_student() values(1,'李4',12,'女','123456744@qq.com')
insert into t_student() values(2,'李武',12,'女','123456744@qq.com')
-- 分开的sno，name可以重复，但是（sno，name） 联合起来就不可以重复
```

- not null 和unique联合使用

```
CREATE TABLE t_student
(
  sno int(10) not null unique,  -- sno变成主键
    name varchar(22)
);
```

3 主键约束 （primary key :PK）

- 主键约束：一个表只能有一个主键
- 主键字段：字段添加主键约束
- 主键值：**每一行记录的唯一标识。**
  - 建议使用：int 、bigint、char等类型
  - 不建议使用：varchar来做主键，主键值定长的。
- 特征
  1. not null + unique （主键值不能为空，不能重复）
- 任何每一张表都要有主键，没有就是非法的

```
drop table if exists t_student
CREATE TABLE t_student
(
  sno int primary key ,
    name varchar(22)
);

-- 表级约束添加主键约束
drop table if exists t_student
CREATE TABLE t_student
(
  sno int ,
    name varchar(22),
    primary key(sno)
);

-- 一个字段做主键：单一主键

-- 多个字段做主键：复合主键  （不要使用）
drop table if exists t_student
CREATE TABLE t_student
(
  sno int ,
    name varchar(22),
    primary key(sno,name)
);
```

- 自然主键：自然数（使用多）
- 业务主键：主键值与业务紧密关联。（使用少，业务变动会影响到业务会改变）

```
-- 自动维护主键值
drop table if exists t_student
CREATE TABLE t_student
(
  sno int primary key auto_increment,  --  使sno自增：auto_increment
    name varchar(22)
);
```

4 外键约束（foreign key： FK)

- 术语：
  - 外键约束：
  - 外键字段：字段添加上外键约束
  - 外键值：外键字段中的每一个值。可以为null
- 父表：被引用的表
- 子表：引用的表

顺序： （理解，不要死记）

1. 删除表：
   - 先删除子表，再删除父表
2. 创建表
   - 先创建父表，再创建子表
3. 删除数据
   - 先删除子数据，再删除父数据
4. 插入数据
   - 先插入父，再插入子

```
CREATE TABLE t_calss   -- 父表
(
  classno int primary key ,  --  使sno自增：auto_increment
    classname varchar(22)
);

CREATE TABLE t_student  -- 子表
(
  sno int primary key auto_increment,
    name varchar(22),
    classcno int foreign key(classno) references t_class(classno)
);
```

2.5 数据处理函数

- 又称为：单行处理处理函数

2.2.1 单行处理函数

- 一个输入，对应一个输出

函数名	说明
lower	大写变成小写
upper	小写变成大学
substr	取子串（被截取的字符串，起始下标，截取的长度）
length	取长度
trim	去前后空格

函数名	说明
str_to_date	将字符串转换成日期
date_fromat	格式化日期
round	四舍五入
rand	生成随机数
ifnull	将null转换成一个具体值

```
SELECT loser(name),age FROM student  -- name 内容变成小写

SELECT round(123456.123,0) FROM studnet ; -- round(参数1，参数2)  参数1：要四舍五入的数；参数2：保留多少位小数
```

2.2.2 多行处理函数

- 多个输入，对应一个输出

分组函数 / 聚合函数 / 多行处理函数 在上面的分组查询中查看 这里的内容

3 存储引擎

（了解）

- 表存储 / 组织
- 建表时：指定存储引擎

```
engine  -- 指定存储引擎
CHARSET -- 指定表的字符编码
```

- MySQL支持的搜索引擎

```
show engines \G  -- 显示目前数据库版本支持的数据库引擎
```

  - MyISAM
    - MyISAM的表具有以下的特征：
      1. 格式文件—存储表结构的定义(mytable.frm)
      2. 数据文件—存储表行的内容(mytable.MYD)
      3. 索引文件–存储表上索引(mytable.MYI)：索引是一本书的目录，缩小扫描范围，提高效率
  - InnoDB：
    - MySQL默认存储引擎，重量级引擎
    - 支持事务，保证数据库的安全，支持事务回滚
    - 服务器崩溃了，支持自动恢复
    - 表和索引存储在一个表空间内
  - MEMORY：
    - 数据存储在内存当中，断电就没有

- 表数据及索引被存储在内存中，查询快，效率高，不需要与硬盘交互。

4 事务

transaction

- 事务：完整的业务逻辑。

事务的DML语句

```
insert
update
delete
```

- 数据的增删改查要考虑安全
- 多条DML语句共同来联合完成
- **多条语句同时成功，或者同时失败！**

4.1 实现事务

4.1.1 提交事务 （commit）

- 清空事务性活动的日志文件，将数据全部彻底持久化到数据库表中
- 提交事务标志着，事务的结束。全部成功的结束。

mysql 默认情况下是自动提交事务

- 自动提交不符合我们开发的业务，必须多条同时执行成功才进行提交业务。

```
-- 关系自动提交机制，先执行
start transaction
```

4.1.2 回滚事务 （rollback）

- 将之前所有的DML 操作全部撤销，并且清空事务性活动的日志文件
- 回滚事务标志着，事务的结束。全部失败的结束。

```
-- 在经过一系列的增删改查之后
rollback ;
```

回滚到上一次的提交

4.2 事务的特性

- A： 原子性

  说明事务是最小的工作单元，不可以再分。

- C： 一致性

  所有的事务要求：在同一个事务当中，所有操作必须同时进行，或者同时失败，保证数据的一致性。

- I ： 隔离性

  A事务和B事务之间具有一定的隔离。

  A和B事务同时操作同一个表，结果会怎么样?

- D：持久性

  事务最终结束的一个屏障。事务提交，将没有保存到硬盘上的数据保存到硬盘上。

4.2.1 事务的隔离性

1 事务的隔离级别

1. 读未提交：read uncommitted （最低的隔离级别） **《提交之前就可以读到》**
   - 事务A可以读取到事务B未提交的数据
   - 脏读现象 （Dirty read）
   - 理论情况下
2. 读已提交：read committed **《提交之后才能读到》**
   - 事务A只能读取到事务B提交之后的数据
   - 解决了脏读现象
   - 不可重复读取数据
   - 每一次读取到的数据是真实数据
3. 可重复读：repeatable read **《提交之后也读不到 ： 读取到开启事务时的数据。事务不结束数据就不会改变》**
   - 事务A开启之后，每一次在事务A中读取到的数据都是一致的。即使事务B修改数据，事务A中读取到的数据依然没有改变。
   - 解决了不可以重复读问题
   - **出现幻影读：读取的数据都是幻想**

4. 序列化 / 串行化：serializable （最高的隔离级别）
   - 效率最高，解决所有问题
   - 事务要进行排队，不能并发！！

- 查看事务隔离级别：

```
select @@tx_isolation
```

- 设置全局隔离级别

```
set global transaction isolation level read uncommitted  -- 读wei
```

5 索引

- 索引是在数据表字段上添加的，**提高查询效率**
- 一个字段可以添加一个索引，多个字段联合起来也可以添加索引

添加索引条件：

1. 数据量庞大
2. 以条件查询的形式存在
3. 很少的DML（insert 、update、delete）操作

5.1创建索引

1. 创建索引语法：

```
create index 索引名 on 表名（字段）;
```

2. 删除索引

```
drop index 索引名 on emp;
```

底层原理：二叉树（B-tree）

3. 查看索引

```
explain select * from 表 where 条件
```

5.2索引失效

1. 模糊查询尽量避免以“%”开始，则会开始进行索引查询，否则不会进行索引查询。索引查询时必须知道第一个字母是什么
2. 使用 or 的情况失效，or两边字段同时有索引才会走索引。
3. 复合索引，没有使用左侧的列查找
4. where当中索引列参加了运算
5. 在where当中索引列使用了函数

5.3 索引的分类

1. 单一索引： 单个字段添加索引
2. 复合索引： 两个或多个字段添加索引
3. 主键索引： 主键上添加索引
4. 唯一性索引： 具有unique 约束的字段上添加索引 （唯一性较弱的字段上索引用处不大）

union 不会使用索引失效

6 视图

> view

- 不同角度看同一份数据

6.1 创建视图

```
create view 视图名 as select * from 表名
```

- 室友DQL语句才能以view的形式创建

6.2 删除视图

```
drop view 视图名
```

6.3 视图的CRUD

- 视图的操作会影响到原表的操作。
- 视图是一张临时表

7 DBA命令

7.1 新建用户

```
CREATE USER username IDENTIFIED BY 'password';
```

- username：你将创建的用户名,
- password：该用户的登陆密码,密码可以为空,如果为空则该用户可以不需要密码登陆服务器.

7.2 授权

```
grant all privileges on dbname.tbname to 'username'@'login ip' identified by 'password' with grant option;
```

1. dbname=*表示所有数据库
2. tbname=*表示所有表
3. login ip=%表示任何ip
4. password为空，表示丌需要密码即可登录
5. with grant option; 表示该用户还可以授权给其他用户

用户权限表

权限名	权限描述
alter	修改数据库的表
create	创建新的数据库戒表
delete	删除表数据
drop	删除数据库/表
index	创建/删除索引
insert	添加表数据

权限名	权限描述
select	查询表数据
update	更新表数据
all	允许任何操作
usage	只允许登录

7.2.1 回收授权

```
revoke privileges on dbname[.tbname] from username;
```

刷新权限：

```
flush privileges;
```

7.3 数据的导入导出

7.3.1数据的导出

- 在dos命令窗口进行

  导出指定数据库

  ```
  C:\Users\25849>mysqldump bookshop>D:\bookshop.sql -uroot -p123456
  ```

- 导出指定表

  ```
  C:\Users\25849>mysqldump bookshop emp> D:\ bookshop_emp.sql -uroot -
  p123456
  ```

7.3.2数据的导入

```
C:\Users\25849>mysql -uroot -p123456
mysql>create database bookshop;
mysql>use bookshop;
mysql>source D:\bookshop.sql
```

8 数据库设计的三规范

数据表的设计依据，对数据库表的设计

- 第一范式（1NF）：要求任何一张表必须要有主键，每一个字段为原子性，字段不可以再分。

- 第二范式（2NF）：建立在第一范式基础上，要求所有非主键字段完全依赖于主键，不存在部份依赖
- 第三范式（3NF）：建立在第二范式基础上，要求所有非主键字段直接依赖于主键，不存在传递依赖
- 减少数据表的冗余

9 SQL高级应用

8.1 T-SQL程序设计

8.1.1 变量

1. 全局变量

```
@@ -- @@开头的变量
```

   - 系统定于和维护，用户无法进行修改或管理

2. 局部变量

```
declare @i int  -- 使用关键declare 声明变量 i，数据类型为：int
```

   - 局部变量赋值

```
-- 使用 set 或 select
set @i = 123
select @i = 1
```

8.1.2 流程控制语句

1 if 语句

```
# 类似 java 中的 if ··· else if ··· else
if <条件表达式>
  <命令行或程序块>
else
  <命令行或程序块>
```

2 begin ··· end语句

```
BEGIN
  <命令行或程序块>
END
```

3 IF [NOT] EXISTS 语句

```
IF [NOT] EXISTS(select 子查询)
  <命令行或程序块>
else
  <命令行或程序块>
```

4 CASE 语句

格式一:

```
CASE <表达式>
  WHEN <表达式> THEN <表达式>
  ...
  WHEN <表达式> THEN <表达式>
  [ELSE  <表达式>]
END
```

格式二:

```
CASE <表达式>
  WHEN <表达式> THEN <表达式>
  ...
  WHEN <表达式> THEN <表达式>
  [ELSE  <表达式>]
END
```

```
SELECT SNo,CNo,Score =
  CASE
    WHEN Score is null THEN '未考'
    WHEN Score < 60  THEN '不及格'
    WHEN Score >= 60 AND Score <= 90 THEN '良好'
    WHEN Score >= 90 THEN '优秀'
  END
FROM SC
```

5 WHILE 语句

```
WHILE (条件)
  <命令行或程序块>
```

8.2存储过程

8.2.1 存储过程 的有点

1. 模块化的程序设计
2. 高效率的执行
3. 较少网络流量
4. 可以作为安全机制使用

8.2.2 存储过程的分类

1. 系统存储的过程
2. 用户自定义存储过程
3. 扩展存储过程

8.2.3 创建存储过程

```
DROP TABLE IF EXISTS t_student;

CREATE TABLE t_student
(
    id INT(11) PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  age  INT(11) NOT NULL
);

INSERT INTO t_student VALUES(NULL,'懿',22),(NULL,'小懿',18);
```

8.3触发器

8.3.1 分类

1. DML 触发器
2. DDL 触发器

8.3.2 创建DML触发器

格式:

```
CREATE TRIGGER 触发器名称
ON { table | view }
{ FOR | AFTER | INSTEAD OF }
{ [INSERT] | [UPDATE] | [DELETE] }
AS
SQL语句[,...n]
```

例子：修改student 表数据，修改之后查询修改后的数据。

```
--创建修改之后的触发器
CREATE TRIGGER trig_student_After
ON student
FOR  UPDATE
AS
  PRINT 'THE TRIGGER IS AFTER'
  SELECT * FROM student
```

- FOR 和AFTER 作用一样

8.3.3 创建DDL触发器

```
CREATE TRIGGER 触发器名称
 ON { ALL SERVER | DATABASE }
{ FOR | AFTER }
{ 事件类型|事件组}[,...n]
AS
SQL语句[,...n]
```

例子：插入数据库后输入‘创建数据库’

```
CREATE TRIGGER trig_create
ON ALL SERVER
 AFTER CREATE_DATABASE
AS
  PRINT '创建数据库'
```