# Mysql非select部分（超哥）

## 前言

## 什么是数据库？

数据库就是存储数据的仓库。

数据是对客观事物的观察结果，是对现实世界的总结和归纳。

数据库是长期存储在计算机内的、大量的、可共享的数据集合，数据按照一定的数学模型进行存储和管理，有较小的冗余度、较高的独立性和安全性。

通常提到数据库都不是指存储数据的位置（硬盘空间），而是指整个数据库系统(dbs)

数据库系统是由数据库、数据库管理系统（dbms），操作系统，服务器，客户端，硬件设备，开发人员，测试人员，用户，数据库管理员等组成。

## 学习数据库的侧重点

数据库管理员：学习数据库的所有知识，如原理、结构、安全、基本操作，数据的优化、效率提升，备份恢复等等

开发/测试：对数据基本的操作，增删改查。

数据库知识在测试工作中的具体应用：

1. 搭建测试环境
2. 结合后台数据对软件进行测试
3. 测试数据结构的合理性
4. 准备测试数据

## 关系数据模型

数据模型就是数据存储、操作和管理的方式，由3部分组成：数据结构、数据的操作以及数据的约束。数据库的发展过程中出现过树状结构的模型、网状结构的模型，关系数据模型和面向对象数据模型。树状和网状结构的数据库已经被淘汰，面向对象结构的数据库目前应用的范围很小，市场上主流的是关系数据库。如mysql，oracle，sql server等大部分主流数据库管理系统都是关系数据库。

1. 数据结构是数据存储的方式和数据之间的关系，是对数据的静态描述。

关系数据模型的数据结构就是一张二维表，由行和列组成。

1. 实体：是客观存在的可以相互区分的每个事物。
2. 关系：就是一张表，用来存储一类实体的数据。
3. 属性：就是列，一类实体是通过多个属性来进行描述和刻画的。也叫做表头或者字段。
4. 元组：就是行，记录具体某个实体的属性值，也叫做一条记录。
5. 分量：就是具体的数据，某个实体的某个属性值。
6. 域：属性的取值范围
7. **键：超键，候选键，主键，外键**
8. 数据的操作，定义了允许的操作和操作的规则，是对数据的动态描述

关系数据模型允许对数据的操作是增删改查，使用sql语句完成。

关系数据模型的操作过程对用户隐藏，用户只需要发出指令说明“干什么”，由数 据库管理系统来完成“如何做”。

关系数据模型的操作对象和操作结果都是一张表。

1. 数据的约束，数据应该满足的条件或者规则，为了保证数据的有效性和完整性。

在关系数据模型中，对数据的约束叫做关系完整性，一共有3条规则：

1. 实体完整性：主键的取值不能为空，空值(NULL)是指无意义，不存在，不知道的值，不是0或者空格。
2. 参照完整性：外键的取值要么为空值，要么为它所对应的主键的值，不能是主键之外的值。

学生表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 学号 | 身份证号 | 姓名 | 年龄 | 专业号 |
| 1001 | 10011 |  |  | 1 |
| 1002 | 10012 |  |  | 2 |
| 1003 | 10013 |  |  |  |
| 1004 | 10014 |  |  | 3 |
| 1005 | 10015 |  |  | 4 |
| 1006 | 10016 |  |  | 5 |

专业表

|  |  |
| --- | --- |
| 专业号 | 专业名 |
| 1 | 数学 |
| 2 | 计算机 |
| 3 | 会计 |
| 4 | 英语 |

学生表的外键对应专业表的主键，表示学生属于专业。学生表中的外键专 业号可以空，表示还没有分配专业。但是，如果要给学生分配专业，必须 分配到一个已存在的专业。

1. 用户定义的完整性：站在用户的角度，数据应该满足的条件或者规则，不满足这些条件的数据同样没有意义。

课程表

|  |  |  |
| --- | --- | --- |
| 课程编号 | 课程名称 | 学分 |
| 20045 | 高数 | 4 |
| 20046 | 体育 | 2 |
| 20047 | 大学物理 | 3 |

课程表中，课程编号是主键，它不能为空并且唯一。

课程名称不是键，仍然不能重复而且不能为空。

学分必须是正数，并且是0.5的整数倍，也不能为空

## 范式

在设计数据库的结构时，表结构以及表和表之间的数据关系，需要遵循一些规则，这些规则就是范式。目前数据库的范式有6条：第一范式，第二范式，第三范式，BC范式，第四范式，第五范式。范式之间是递进关系，如要满足第二范式，则必须先满足第一范式。一般情况下了解前4个范式就能够满足数据设计的需要。范式的要求主要是为了保证数据的独立性和有效性，减少数据之间相互的干扰以及减少冗余数据。

参考三范式表：

第一范式：同一张表中不能有重复的列。数据库管理系统默认支持第一范式

第二范式：非主属性完全依赖于主属性，同一张表只能存储一类实体的数据。不能将多类实体的数据放到一张表中。

第三范式：非主属性不能出现在其他表里。要引用其他表的数据，只能引用主键不能引用其他属性。数据库管理系统默认支持第三范式；

BC范式：如果多张表之间数据有关系，必须分析清楚数据的关系，用主键外键对应。不能将某两张表的关系放到第三张表表示

## 键

超键：可以唯一标识一行的属性的集合



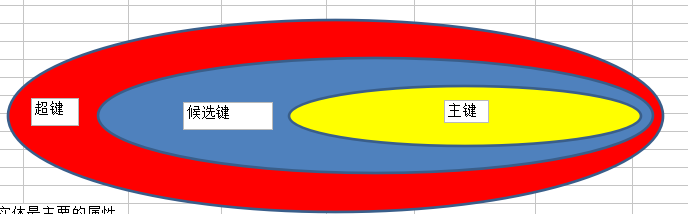
候选键：不包含多余属性的超键的集合



主键：在候选键中选择一个作为主键； ----一张表中主键只能有一个，通常选择该类实体最主要的属性

外键：引用的另一张表的主键； -----通常表和表之间的数据是有联系的，要将这样的联系体现出来使用主键和外键的对应





### --索引

##### --什么是索引？

索引（INDEX）类似书的目录，是供服务器在表中快速查找和操作行的数据库结构（索引需要占用存储空间）索引是数据和存储位置的列表，索引是以列为单位建立的、索引之所以能提高查询速度，是因为他们是按照查询条件存储数据的，数据量少而已排列有序，另外还提供了一个指向内容的指针，即记录的ROWID（类似书的页码）。

##### --索引的设计原则

1：经常出现在where，order by，group by后面的列可以创建索引；

2：有大量空值的列，数据频繁改动的列，不经常作为查询条件的列不要创建索引 （每次变动都会导致索引存储的数据更新）；

3：当某个数据具有唯一性特征时，可以指定唯一索引，以确保数据的完整性，提高查询效率；

##### --索引的有点

1：可以大大价款数据的查询速度，这是建索引的主要原因；

2：索引的内部结构由系统自动创建，每次对数据的修改索引也会自动更新；

3：在创建主键约束和唯一约束时，会自动创建主键索引和唯一索引，保证每行数据的唯一性；

##### --索引的不足

1：索引虽然提高了查询速度，同时却会降低更新表的速度，因为更新表时，MySQL不仅要保存数据，还要保存一下索引文件:；

2：建立索引会占用磁盘空间的索引文件，每个表至少支持16个索引（依据存储引擎的不同而有所区别）；

##### --查看索引

Show index from emp ; //// show indexes from emp (不同老师的不同写法，我也不知道那个是对的，但是好像都可以实现一样）；

##### --创建索引

Create index 索引名称 on 表名（列名（length））;--如果是 BLOB、TEXT的类型则需要制定length

Create index ind1 on from emp(ename)给EMP表的ename创建索引;

-MySQL中的索引类型主要是B树索引;

##### --添加普通索引

Alter table 表名 add index 索引名称（列名）；

Alter table emp add index ind1(mgr); 为表emp 的mgr列添加索引；

##### -- 创建唯一索引

Create unique index 索引名称 on 表名（列名（length））; unique / ju: ni:k/ 唯一的、

--添加唯一索引

Alter table 表名 add unique 索引名称（列名）

##### --组合索引

按职位确定部门编号(job，deptno);

CREATE INDEX ind\_job\_deptno ON emp(job,deptno);

如果查询条件只有job，那么该索引生效（以左边的列为准）;

如果查询条件包含job，deptno，该索引依然生效;

如果查询条件只有deptno 则该索引不生效;

例如：CREATE INDEX my\_job\_deptno ON emp (job,deptno)

##### --删除索引

DROP INDEX 索引名称 ON 表名;

DROP INDEX ind\_ename ON emp;

DROP INDEX ind\_mgr ON emp;

DROP INDEX ind\_job\_deptno ON emp;

##### --注意点

如果想对某个列使用索引，那么在SELECT语句中就不要对该列使用<>,!=

SELECT \* FROM emp

WHERE empno<>7369; -- 导致empno的索引不生效

SELECT \* FROM emp

WHERE empno>7369 OR empno<7369; -- empno的索引生效

使用LIKE进行查询，不能用%开头

SELECT \* FROM emp

WHERE ename LIKE '%S'; -- ename的索引不生效

## --视图

视图view，是一种数据库对象，本身不存储数据，但是可以帮助我们更好更方便的使用数据，视图包含一系列带有名称的列和行的数据；视图主要出于两种原因（1、安全原因，视图可以隐藏一些数据、2、可以使复杂的查询易于理解和使用）；

视图保持的是一个查询语句，可以理解为是查询语句的快捷方式

查询语句原本的数据来源成为基表，对视图的访问和操作相当于对基表中数据的操作

将复杂的查询语句保持成视图，可以快速访问基表中特定的内容；

视图可以针对不同的角色区分不同的权限；

视图可以简化查询语句

Create （or replace）view 视图名称

[别名1，别名2......别名N]

As

Select --- 完整的查询语句

--如果创建视图时，视图名称已经存在，不加or replace 会直接报错，，如果加上 or replace 则会替换已有的视图，可以理解为修改已有的视图；

--区分权限

举例：公司的BOSS和HR可以访问所有数据并且修改；直接访问emp表

部门经理只能访问本部门的数据并且修改；给部门经理创建部门视图；create view dept20 （ # 视图名称）as select \* from EMP where deptno=20；

举例： HR BOSS 想要了解每个员工的薪资情况

SELECT ename,sal FROM emp;

---部门经理只能查看自己部门的员工薪水

SELECT ename,sal FROM emp WHERE deptno=20;

-- 视图

CREATE VIEW v\_deptno20（#视图名称）

AS

(SELECT ename,sal FROM emp WHERE deptno=20); # 至于到底加不加（）两个老师写法不一样感觉差不多一样都可以实现耶....

--对表的增删改查操作，同样可以对视图操作，，对视图中数据的修改，相当于是对基表中数据的修改；

举例：-- 视图的使用和表的使用是相似的（把视图当做表来操作）

SELECT \* FROM v\_deptno20;

-- 修改SMITH的薪水，涨薪200

UPDATE v\_deptno20

SET sal=sal+200 WHERE ename='SMITH' ;

原表中即emp表中SMITH薪水一样涨了200，对视图修改/删除实际上都是对原表数据操作，视图本身不存储数据的；

--简化复杂的查询语句

使用部门名称来查询该部门的平均工资

SELECT AVG(e.sal) FROM emp e,dept d

WHERE e.deptno=d.deptno

AND d.dname='SALES'

-- 视图简化

CREATE VIEW v\_emp\_dept #（视图名称）

AS

(SELECT e.\*,d.dname,d.loc FROM emp e,dept d

WHERE e.deptno=d.deptno)

通过视图简化了复杂的查询语句，就可以快速访问数据；--将复杂的多表查询语句简化成了视图。

SELECT AVG(sal) FROM v\_emp\_dept WHERE dname='SALES'; #然后再通过视图直接快速查询

Emp表和dept表关系比较密切，经常互为条件查询，可以提前创建好视图，将这一类的查询语句可以通过视图完成

直接创建多表的视图；

--删除视图

删除视图等于删除对应的SELECT语句，不会对原表产生影响

DROP VIEW 视图名称;

DROP VIEW v\_emp\_dept; 接上例子，删除后再使用SELECT AVG(sal) FROM v\_emp\_dept WHERE dname='SALES';查询则会报错，因为视图已经删除了；

## --表的约束

约束是从表的层面对列中的数据进行限制，让数据满足某种规则或条件，给列创建约束，定义列的属性或者保证数据的有效性和完整性。

DEFAULT -- 默认值

PRIMARY KEY -- 主键约束 简称PK -- primary /prai mei ri/

FOREIGN KEY -- 外键约束 简称FK --foreign /fo ren /

NOT NULL -- 非空约束

UNIQUE -- 唯一约束 -- unique /ju: ni:k /

--数据类型：enum(value1,valu2......valueN) 限定取值的列表，列表中的值

--约束可以在创建表(create）的同时定义，也可以在创建表之后

推荐在create语句中使用

--约束的定义方式有两种，列级约束和表级约束

Create table 表名

(

列名1 数据类型[default 默认值][列级约束1][列级约束2]......，

列名2 数据类型[default 默认值][列级约束1][列级约束2]......，

.......，

列名N 数据类型[default 默认值][列级约束1][列级约束2]......

);

--列级约束直接将关键字写在列后面，外键只能用表级约束定义

表级约束的定义方式：

--约束类型（列名）

Constraint 约束命名 约束类型（列名）

--非空约束只能使用列级约束

--enum 严格来说是数据类型，不是约束

实际例子：

创建一个表，studentinfo

CREATE TABLE studentinfo

(

学号 INT(5) PRIMARY KEY,

姓名 VARCHAR(20) NOT NULL, -- varchar 可变长字符

年龄 TINYINT(2) DEFAULT 20,

性别 ENUM('m','f') DEFAULT'f',

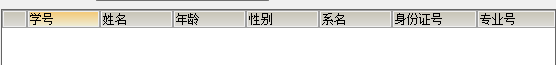
系名 VARCHAR(10),

身份证号 VARCHAR(18),

专业号 INT(4)

)

查询表：select \* from studentinfo



再插入数据：

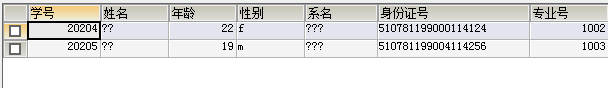
INSERT INTO studentinfo

VALUES('20205','省帅',19,'m','管理系','510781199004114256','1003')

INSERT INTO studentinfo

VALUES('20204','花花','22','f','经济系','510781199000114124','1002')

查询表：select \* from studentinfo



至于为什么名字和系别是？？我也不知道，反正操作方法是这样的，至于为什么是？？再等待解决。

--建表后增加约束，，不推荐，在建表时就应该把约束考虑清楚（）

为studentinfo增加非空约束

ALTER TABLE studentinfo MODIFY 系名 VARCHAR(10) NOT NULL;

DESC studentinfo

-- 删除表中的所有数据

TRUNCATE TABLE studentinfo;

-- 查看表结构

DESC studentinfo；

-- 查看创建表的结构和约束

SHOW CREATE TABLE 表名;

SHOW CREATE TABLE studentinfo;

--添加主键：

Alter table 表名 add constraint 约束命名primary key 列名；

--删除主键：

Alter table 表名 drop primary key;

--添加唯一约束

Alter table 表名 add unique 列名；

--删除唯一约束

Alter table 表名 drop index 列名；

--添加非空约束

Alter table 表名 change 原列名 新列名 数据类型；

--添加外键约束

Alter table 表名 add foreign key（列名）references 表名（列名）；

--删除外键约束，必须先知道外键的名字，

Alter table 表名 drop foreign key 外键名称

# DML操作语句

##### Insert 语句（增加）

-- 按照表结构插入数据

Insert into 表名

Values(value1,value2,value3........valueN)按照表的原有列放入数据，values的数量，顺序和数据类型都要和表的结构想对应。

事物（commit 提交），（rollback 回滚）

DML语句和DDL语句会生成一个事物，事物需要提交才能永久生效；

-- MySQL默认自动提交事务，可以用命令手动修改；

SET autocommit=0; -- 1表示自动提交事务；（修改后就可以回滚啦）；

-- 在部门表中插入一条记录

INSERT INTO dept

VALUES(60,'DEVELOPMENT','CHENGDU');



-- 再插入一条数据

INSERT INTO dept(deptno,loc)

VALUES(50,'mianyang');



---按照自定义的列和顺序放入数据，自定义的列必须和下面的values想对应，可以不写某些列不写的列默认放入空值；

-- 事务在提交之前，只会对本地的数据库产生影响，不会修改到数据库服务器的数据，事务一但提交，则对数据库服务器数据生效，无法再回滚。

-- 事务在提交之前，可以回滚，甚至可以回滚到指定位置 SAVEPOINT（回滚点设置）

-- 请为emp表添加一位新员工LUCY，员工编号8888，该员工的上级是SMITH ，入职日期是当天，薪水200，职位是'CLERK' 部门编号20

INSERT INTO emp

VALUES(8888,'LUCY','CLERK',7369,curdate(),200,NULL,20); ---放入的数据可以是手动输入也可以是函数的结果；

--insert into 表名 select ........；

将查询的结果直接放入另外一张表，相当于拷贝数据

--举个例子：

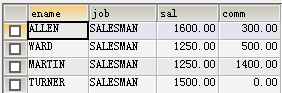
SELECT \* FROM bonus



INSERT INTO bonus

SELECT ename,job,sal,comm FROM emp

WHERE comm IS NOT NULL; ---选择emp 表中的列，ename,job,sal,comm, 限定comm不为空，添加到新的表bouns 中；



---Mysql 允许insert 语句一次放入多行数据；

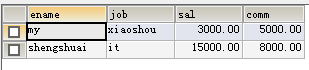
Insert into 表名

Values(values1,values2....valuesN),(values1,values2,.....valuesN)......;

例：

INSERT INTO bonus

VALUES('my','xiaoshou',3000,5000),('shengshuai','it',15000,8000);



---可设置回滚点

Savepoint A -- 设置回滚点

............

............

Rollback to savepoint A --回滚到指定回滚点A的位置；

##### Update 改/更新

Update 表名

Set 列名1=values1,列名2=values2 .......

[where .........]; -----限定条件，如果不写where 会修改所有行的数据；

-- 修改40号部门的地址为BEIJING

UPDATE dept

SET loc='BEIJING' WHERE deptno=40;

可以用当前表的数据作为修改的依据；

例：

- 给emp表中所有员工加薪200美金

UPDATE emp

SET sal=sal+200;

--where 和 set 后面可以使用子查询；

例如：

-- 将dept表中的RESEARCH部门的员工上级改为7839；

UPDATE emp

SET mgr=7839

WHERE deptno=(SELECT deptno FROM dept -----使用子查询

WHERE dname='research');

##### --delete 删除

DELETE (FROM) 表名

WHERE 限定条件 -- 如果不写where子句则表示删除所有数据；

-- 删除bouns表中的所有数据；

DELETE FROM bouns;

-- 删除SALES部门的所有员工信息

DELETE FROM emp

WHERE deptno=

(SELECT deptno FROM dept

WHERE dname='sales'); ------- 30号部门

# 表管理DDL语句

数据定义语句（DDL）create 创建，alter修改结构，drop删除，truncate删除..等语句

##### --创建表

Create table 表名

（

列名1 数据类型 [DEFAULT 默认值][列级约束1,列级约束2 ……],

列名2 数据类型 [DEFAULT 默认值][列级约束1,列级约束2 ……],

...,

列名n 数据类型 [DEFAULT 默认值][列级约束1,列级约束2 ……],

）

-- 表名的命名规则

1)必须以字母开头

2)可包括数字

3)只能包含A-Z, a-z, 0-9, \_, $, AND #

4)不要使用MySQL的保留字

5)同一用户的表对象不能同名

--数据类型

Char(n) 定义最大长度为n，固定长度字符串， n最大128个字符，如果存储的数据不到n个字符，会使用空格补齐到n个字节，保证所有的字符都是n个字符长度，浪费存储空间，查询效率高；

Varchar（n）定义最大长度为n，可变长字符串，n最大65535个字符，节省存储空间，查询效率低；

Int整数

Float(m,n)定义总位数为m,保留n位小数的浮点数；

Double(m,n)定义总位数为m，保留n位小数的双精度浮点数；

N会占用m的位数，例如：float(5,2)表示3位整数2位小数；

日期：

Date(日期）、time(时间）、datetime(日期时间）

Desc 表名 ： 查看表结构

--创建一张新表student :

CREATE TABLE student

(

学号 INT(5),

姓名 VARCHAR(20),

年龄 TINYINT(2), -----tinyint 这个老师没讲，好像是年龄的数据类型

性别 CHAR(1),

系别 VARCHAR(10),

身份证号 VARCHAR(18),

专业号 INT(4)

)

CHARACTER SET=utf8; ---创建的表数据支持中文字符样式

如果不加CHARACTER SET=utf8 则表中不支持中文字符。会以？？？的形式展示出来



-- 补充 修改表名

RENAME TABLE 原表名 TO 新表名;

--如果数字的范围超出则会使用最大值代替，如果给数字的列放入字符串则会转化为0，数字可以直接看着是字符，日期也可以看着是字符，日期类型的列放入的不是日期会产生一个错误日期，定义了相关的列即使输入的是空值也会让默认值失效，只有在insert语句中定义的某些列，未定义的某些列默认值才会生效；

##### --alter 修改表结构

-- 不推荐在建表后修改表结构，特别是在已有数据的表中,但是如果需要修改可以使用alter语句来完成；

--添加/删除列

Alter table 表名 add 列名 数据类型; ------添加列

Alter table 表名 drop 列名; ------删除列

---为表student添加一个列‘联系电话’

Alter table student add 联系电话 int(11);

删除该列;

Alter table student drop 联系电话;

--在语句最后可以添加插入表的位置，first表示添加到第一列，after列名 表示添加到某一列之后

ALTER TABLE student ADD 电话 INT(11)AFTER 姓名;



Delete 属于dml语句，需要提交才能生效，可以回滚撤销操作；

Drop 属于ddl语句，直接提交生效，不能回滚，直接删除整张表，不保留数据和表结构；

Truncate属于ddl语句，直接生效不能回滚。

--修改数据类型：

添加默认值：

Alter table 表名 alter 列名 set default(默认值）

删除默认值：

Alter table 表名 alter 列名 drop default

--修改表：

alter table 表名 change column 旧列名 新列名 新列名的数据类型; --修改列名和数据类型 column / ‘kao leim/ 列

Alter table 表名 modify column 列名 新的数据类型; --修改数据类型 modify/ ‘mao di fi/ 修改

重命名表（rename）:

Rename table 原表名 to 新表名； ---修改表名

--删除表(truncate drop)

Truncate table 表名; ---删除表中的所有数据（不能指定删除范围）

Drop table 表名; ---删除表结构以及表中的数据

--数据控制语言（DCL） GRANT 授权，REVOKE 解除授权，(作为了解）；

--事务处理语言（TCL） COMMIT 提交，ROLLBACK 回滚事务 等语句（前面已经提到这里不再详细说明）。