**数据类型简介**

浮点数DECIMAL (M, D): M表示数据总长度(不包括小数点)；D表示小数位。

例如：DECIMAL (5, 2) 123.45

|  |  |  |  |
| --- | --- | --- | --- |
| 日期与时间类型 | | | |
| 类型 | 字节数 | 取值范围 | 零值 |
| YEAR | 1 | 1910~2155 | 0000 |
| DATE | 4 | 1000-01-01~9999-12-31 | 00 00:00:00 |
| TIME | 3 | -838:59:59~838:59:59 | 00:00:00 |
| DATETIME | 8 | 1000-01-01 00:00:00  ~9999-12-31 23:59:59 | 0000-00-00 00:00:00 |
| TIMESTAMP | 4 | 19700101080001~20370119111407 | 00000000000000 |

字符串类型

CHAR：固定长度字符串。Char(10)：实际只用2位还是10位。

VARCHAR：可变长度字符串。Varchar(10)：实际只用2位就是2位。

TEXT：大文本

ENUM：枚举类型（只能取一个元素）

SET：集合类型（能取多元素）

**数据库基本操作**

显示所有数据库: Show databases.

创建数据库: Create database.

删除数据库: Drop database.

**数据库表的基本操作**

1.创建表: CREATE TABLE 表名(属性名 数据类型 [完整约束条件],

属性名 数据类型 [完整约束条件],

…

属性名 数据类型 [完整约束条件]

);

2.查看表

查看基本表结构:DESCRIBE(DESC)表名;(出现表)

查看表详细结构:SHOW CREATE TABLE 表名;(出现表的详细语句)

3.修改表

修改表名 ALTER TABLE 旧表名 RENAME 新表名;

修改字段 ALTER TABLE 表名 CHANGE 旧属性名 新属性名 新数据类型;

增加字段 ALTER TABLE 表名 ADD 属性名1 数据类型 [完整性约束条件][FIRST/AFTER 属性名2];

删除字段 ALTER TABLE 表名 DROP 属性名;

4.删除表 DROP TABLE 表名;

**查询数据**

查询所有字段

SELECT 字段1，字段2，… FROM 表名;（可以调整字段顺序）

SELECT \* FROM 表名;

查询指定字段

SELECT 字段1，字段2，… FROM 表名;

Where条件查询

SELECT 字段1，字段2，… FROM 表名 WHERE 条件表达式;(……WHERE id = 1/age > 22)

带IN关键字查询

SELECT 字段1，字段2，… FROM 表名 WHERE 字段 [NOT] IN (元素1，元素2，元素3);

(……WHERE age IN (20,23))--所有20和23岁的。

带BETWEEN AND的范围查询

SELECT 字段1，字段2，… FROM 表名 WHERE 字段 [NOT] BETWEEN 取值1 AND 取值2;

(……WHERE age BETWEEN 20 AND 23)--所有20、21、22、23岁的。

带LIKE的模糊查询

SELECT 字段1，字段2，… FROM 表名 WHERE 字段 [NOT] LIKE 字符串;

%表示任意字符;\_表示单一字符。

(……WHERE Name LIKE `张三%`)--只要是张三什么的的都可以

(……WHERE Name LIKE `张三\_`)—代表张三X(……WHERE Name LIKE `张三\_\_`)—代表张三XX

(……WHERE Name LIKE `%张三%`)--只要是什么张三什么的都可以

空值查询

SELECT 字段1，字段2，… FROM 表名 WHERE 字段 IS [NULL];

带AND的多条件查询

SELECT 字段1，字段2，… FROM 表名 WHERE 条件表达式1 AND 条件表达式2;

带OR的多条件查询

SELECT 字段1，字段2，… FROM 表名 WHERE 条件表达式1 OR 条件表达式2;

去重复查询

SELECT DISTINCT 字段名 FROM 表名;

对查询结果排序

SELECT 字段1，字段2，… FROM 表名 ORDER BY 属性名 [ASC/DESC];

(……ORDER BY age ASC)--按年龄升序排列;DESC降序

GROUP BY分组查询

SELECT 字段1，字段2，… FROM 表名 GROUP BY 属性名 [HAVING 条件表达式][WITH ROLLUP]

1. 单独使用(……GROUP BY grade)，毫无意义
2. 与GROUP\_CONTACT()函数一起使用；

SELECT grade, GROUP\_CONTACT(Name) FROM 表名 GROUP BY grade;(出现每个grade的人的Name)

1. 与聚合函数一起使用；

每个年级学生总数：SELECT grade, COUNT(Name) FROM 表名 GROUP BY grade;

1. 与HAVING一起使用（起限制作用，查询结果筛选）

每个年级学生总数大于3的：SELECT grade, COUNT(Name) FROM 表名 GROUP BY grade HAVING COUNT(Name) > 3;

1. 与WITH ROLLUP一起使用(最后加入一个总和行);

每个年级学生总数(多一行总计)：SELECT grade, COUNT(Name) FROM 表名 GROUP BY grade WITH ROLLUP;

LIMIT分页查询

SELECT 字段1，字段2，… FROM 表名 LIMIT 初始位置，记录数;

SELECT \* FROM 表名 LIMIT 0,5；模拟第一页，五个数据

SELECT \* FROM 表名 LIMIT 5,5；模拟第二页，五个数据

……

**使用聚合函数查询**

COUNT()函数

用来统计记录的条数。常与GROUP BY关键字一起使用。

SUN()函数

求和函数。常与GROUP BY关键字一起使用。

AVG()函数

求平均值函数。常与GROUP BY关键字一起使用。

MAX()函数

求最大值函数。常与GROUP BY关键字一起使用。

MIN()函数

求最小值函数。常与GROUP BY关键字一起使用。

**连接查询**

——将两个或两个以上的表按照某个条件连接起来

SELECY \* FROM 表1，表2；两张表笛卡尔乘积的形式合并

内连接——查询两个或两个以上的表；

外连接——可以查出某一张表的所有信息

SELECT 属性名列表 FROM 表名1 LEFT/RIGHT JOIN 表名2 ON 表名1.属性名1=表名2.属性名2；(左连接：把第一张表的罗列出来；右连接：把第二张表的罗列出来)

多条件连接查询(加AND)

**子查询(在查询结果中查询)**

带IN关键字的子查询

一个查询语句的条件可能落在另一个SELECT语句的查询结果中。

SELECT id FROM t\_booktype;

SELECT \* FROM t\_book WHERE booktypeId (NOT) IN (SELECT id FROM t\_booktype);

带比较运算符的子查询

SELECT \* FROM t\_book WHERE price >= (SELECT price FROM t\_pricelevel WHERE priceLever = 1);

带Exists关键字的子查询

SELECT \* FROM t\_book WHERE (NOT) EXSISTS (SELECT \* FROM t\_booktype);

当后面括号中的SELECT为TRUE时执行前面的查询。

带Any关键字的子查询

SELECT \* FROM t\_book WHERE price >= ANY (SELECT price FROM t\_pricelevel);

只要大于等于任何一个pricelevel就好。

带ALL关键字的子查询

SELECT \* FROM t\_book WHERE price >= ALL (SELECT price FROM t\_pricelevel);

只要大于等于所有pricelevel就好。

**合并查询结果**

UNION

使用UNION关键字是，数据库系统会将所有的查询结果合并到一起，然后去除掉相同的记录；

SELECT id FROM t\_book UNION SELECT id FROM t\_booktype;

UNION ALL

使用UNION ALL，不会去除掉系统的记录；

**为表和字段取别名**

为表取别名 格式：表名 表的别名

为字段取别名 格式：属性名 AS 别名

**插入、更新和删除数据**

插入数据

1.给表的所有字段插入数据

格式：INSERT INTO 表名 VALUES(值1，值2，值3……);

2.给表的指定字段插入数据

格式：INSERT INTO 表名(属性1，属性2，属性3……) VALUES(值1，值2，值3……);

3.同时插入多条记录

格式：INSERT INTO 表名[(属性列表)]

VALUES (取值列表1),(取值列表2)

…,

(取值列表n);

更新数据

UPDATE 表名

SET 属性名1=取值1，属性名2=取值2，

…,

属性名n=取值n

WHERE 条件表达式；

UPDATE t\_book SET bookName=’Java编程思想’,price=120 WHERE id=1;

UPDATE t\_book SET bookName=’我’ WHERE bookName LIKE’%我爱我家%’;

删除数据

DEKETE FROM 表名 WHERE 条件表达式;

**索引**

索引引入

索引是由数据库表中一列或者多列组合而成，其作用是提高对表中数据的查询速度；类似于图书的目录，方便快速定位，寻找指定的内容；

索引的优缺点

优点：提高查询数据的速度；

缺点：创建和维护索引的时间增加了；

索引的分类

1. 普通索引

可以创建在任何数据类型中。

1. 唯一性索引

使用UNIQUE参数可以设置。在创建唯一性索引时，限制该索引的值必须是唯一的。

1. 全文索引

使用FULLTEXT参数可以设置，全文索引只能创建在CHAR,VARCHAR,TEXT类型的字段上。主要作用就是提高查询较大字符串类型的速度；只有MyISAM引擎支持该索引，Mysql引擎默认不支持。

1. 单列索引

在表中可以给单个字段创建索引，单列索引可以是普通索引，也可以是唯一性索引，还可以是全文索引。

1. 多列索引

多列索引是在表上的多个字段上创建一个索引。

1. 空间索引

使用SPATIAL蚕食可以设置空间索引。空间索引只能建立在空间数据类型上，这样可以提高系统获取空间数据的效率。

创建索引

CREATE TABLE 表名(属性名 数据类型[完整性约束条件],

属性名 数据类型[完整性约束条件],

……

属性名 数据类型

[UNIQUE/FULLTEXT/SPATIAL/] INDEX/KEY [别名](属性名1[(长度)][ASC/DESC])

);

2.在已经存在的表上创建索引

CREATE [UNION/FULLTEXT/SPATIAL] INDEX 索引名**ON** 表名 (属性名 [(长度)][ASC/DESC])；

3.用ALTER TABLE 语句来创建索引

ALTER TABLE 表名ADD [UNION/FULLTEXT/SPATIAL] INDEX 索引名ON 表名 (属性名 [(长度)][ASC/DESC])；

删除索引

DROP INDEX 索引名 ON 表名;

**视图**

**视图的引入**

1.视图是一种虚拟的表，是从数据库中一个或多个表中导出来的表。

2.数据库中只存放了视图的定义，而没有存放视图中的数据，这些数据存放在原来的表中。

3.使用视图查询数据时，数据库系统会从原来的表中取出对应的数据。

**视图的作用**

1. 使操作简便化；
2. 增加数据的安全性；
3. 提高表的逻辑独立性；

**创建视图**

CREATE [ALGORITHM = {UNDEFIEND/MERGE/TEMPTABLE}]

VIEW 视图名 [(属性清单)]

AS SELECT 语句

[WITH [CASCADED/LOCAL] CHECK OPTION];

单表上创建视图

CREATE VIEW v1 AS SELECT \* FROM t\_book;

CREATE VIEW v2 AS SELECT bookName, price FROM t\_book;(只显示bookName 和 price)

CREATE VIEW v3(b, p) AS SELECT bookName, price FROM t\_book;(改了表头名字为b和p)

多表上创建视图

CREATE VIEW v4 AS SELECT bookName, bookType FROM t\_book, t\_bookType where t\_book.bookTypeId = t\_bookType.id;

CREATE VIEW v5 AS SELECT tb.bookName, tby.bookTypeName FROM t\_book tb, t\_bookType tby where tb.bookTypeId = tby.id;(使用了别名)

**查看视图**

1. DESCRIBE 语句查看视图基本信息

DESC v5；

1. SHOW TABLE STATUS 语句查看视图基本信息

SHOW TABLE STATUS LIKE `v5`;

1. SHOW CREATE VIEW 语句查看视图详细信息

SHOW CREATE VIEW v5;

4.在views表中查看视图详细信息(可视化界面里)

**修改视图**

1. CREATE OR REPLACE VIEW语句修改视图(视图不存在CREATE，已经有了REPLACE)

CREATE OR REPLACE [ALGORITHM = {UNDEFIEND/MERGE/TEMPTABLE}]

VIEW 视图名 [(属性清单)]

AS SELECT 语句

[WITH [CASCADED/LOCAL] CHECK OPTION];

1. ALTER语句修改视图

ALTER [ALGORITHM = {UNDEFIEND/MERGE/TEMPTABLE}]

VIEW 视图名 [(属性清单)]

AS SELECT 语句

[WITH [CASCADED/LOCAL] CHECK OPTION];

例：CREATE OR REPLACE v1(bookName, price) AS SELECT bookName, price FROM t\_book;

还原：ALTER VIEW v1 AS SELECT \* FROM t\_book;

**更新视图**

更新视图是指通过视图来插入(INSERT)、更新(UPDATE)和删除(DELETE)表中的数据。因为视图是一个虚拟的表，其中没有数据。通过视图更新时，都是转换基本表来个更新。更新视图时，只能更新权限范围内的数据。超出了范围，就不能更新。

INSERT INTO v1 VALUES (NULL,`java good`,120,`wang`,1);

UPDATE v1 SET bookName = `java very good`, price = 200 WHERE id = 5;

DELETE FROM v1 WHERE id = 5;

**删除视图**

删除时只删除视图的定义，不会删除数据。

DROP VIEW [IF EXISTS]视图名列表 [RESTRICT/CASCADE];

**触发器**

触发器(TRIGGER)是由事件来触发某个操作。这些事件包括INSERT语句、UPDATE语句和DELETE语句。当数据库系统执行这些事件时，就会激活触发器执行相应的操作。

**创建和使用触发器**

1.创建只有一个执行语句的触发器

CREATE TRIGGER 触发器名 BEFORE/AFTER 触发事件

ON 表名 FOR EACH ROW 执行语句;

例：CREATE TRIGGER trig\_book AFTER INSERT

ON t\_book FOR EACH ROW

UPDATE t\_bookType SET bookNum + 1 WHERE new.bookTypeId = t\_booktype.id;

INSERT INTO t\_book VALUES (NULL,`JAVA GOOD`,100,`wang`,1);

2.创建多个执行语句的触发器

CREATE TRIGGER 触发器名 BEFORE/AFTER 触发事件

ON 表名 FOR EACH ROW

BEGIN

执行语句列表

END

例:

DELIMITER |

CREATE TRIGGER trig\_book2 AFTER DELETE

ON t\_book FOR EACH ROW

BEGIN

UPDATE t\_bookType SET bookNum - 1 WHERE old.bookTypeId = t\_booktype.id;

INSERT INTO t\_log VALUES (NULL, NOW(), `在book表里删除了一行数据`);

DELETE FROM t\_test WHERE old.bookTypeId = t\_test.id;

END

|

DELIMITER;

注：默认情况下，delimiter是分号;。在命令行客户端中，如果有一行命令以分号结束，那么回车后，MySQL将会执行该命令。DELIMITER(分隔符)是为了告诉MySQL解释器，该段命令是否已经结束了，MySQL是否可以执行了。

DELETE FROM t\_book WHERE id = 5;

**查看触发器**

SHOW TRIGGERS;

在系统表里看

**删除触发器**

DROP TRIGGERS 触发器名;

测试git