



第一章

1.1

DB - DataBase 数据库

DBMS - DataBase Management System 数据库管理系统

SQL - Structured Query Language 结构化查询语言

1.2 MySQL 基础事实

- ① MySQL 是关系型数据库 (数据有值有来格开列)
- ② MySQL 是开源软件
- ③ 默认端口号: 3306
- ④ 退出命令: exit 或 quit
- ⑤ 登录命令: `mysql -u root -p` (u为用户名, p输密码)

1.3 容易混淆坑

`mysqld.exe` (服务进程) $\neq$ `mysql.exe` (客户端)

配置文件: `my.ini` (Windows) `my.cnf` (Linux)

第二章

① 创建数据库: `CREATE DATABASE` 数据库名

如果不存在则创建 `CREATE DATABASE IF NOT EXISTS` 数据库名

`CREATE DATABASE IF NOT EXISTS library`

`DEFAULT CHARACTER SET utf8mb4`

`DEFAULT COLLATE utf8mb4_general_ci;`

完整版.

② 查看数据库

SHOW DATABASES

查看数据库

SHOW CREATE DATABASE 库名

查看某个库的创建语句

③ 选择/使用数据库

USE 数据库名

选择要操作的数据库

④ 修改数据库 (不能改数据库的名字, 只能改字符集与属性)

ALTER DATABASE 库名 DEFAULT CHARACTER SET utf8mb4;

字符集

⑤ 删除数据库

DROP DATABASE 数据库名

DROP DATABASE IF EXISTS 数据库名 — 安全写法

字符集 (Character set): 决定能存进哪些字符 常用 utf8mb4

校对规则: 决定字符怎么比较和排序。

命名规则: 字符集 - 语言 - 后缀

2.3 存储引擎

		事务	外键
InnoDB	MySQL 默认引擎, 支持事务和外键	✓	✓
MyISAM	速度快但不支持事务	×	×
MEMORY	数据存在内存 速度极快 关机数据消失	×	×

第三章

一、表

表：数据库里所有数据都有在表里

Reader ID	Reader Name	Sex	Phone
1129	李二	女	13XXXX
12003	王五	男	1X12XX

表名：reader

列名/字段名：ReaderID、ReaderName

行/记录：每一横行是一条记录

列/属性：每一竖列是一种属性

数据项：每个格子里的具体值

完整性约束：保证数据合法的规则

最长35

5类常用数据类型

数...

4 artifacts

分类	类型	用途	例子
整型	INT	整数	年龄、数
浮点型	FLOAT(m,n)	不精确小数	普通价格
精确小数	DECIMAL(m,n)	精确小数 ★	工资、金
定长字符串	CHAR(n)	长度固定的字符串	学号、身
变长字符串	VARCHAR(n)	长度变化的字符串	姓名、地
大文本	TEXT	超长文本	商品详情
日期	DATE	只存日期	2026-06
日期时间	DATETIME	日期+时间	2026-06
年份	YEAR	只存年	2026
位	BIT	0/1	性别、是

1. CHAR VS VARCHAR

CHAR(10)存'abc'实际占10字节(用空格补满)

VARCHAR(10)存'abc'实际占3字节(实际长度+1字节记录长度)

2. FLOAT VS DECIMAL

FLOAT是不精确的
可能出現 0.1+0.2=0.3000000
DECIMAL是精确的

完整性约束

完整性

① 实体完整性

约束方式

PRIMARY KEY (主键)
UNIQUE KEY (唯一键)

通俗解释

每一行都有6位唯一识别, 不能重复

② 参照完整性

FOREIGN KEY ...
REFERENCES (外键)

表与表之间关联不能乱

③ 域完整性

NOT NULL, DEFAULT
CHECK

列的值必须合法
(不能为空, 有默认值, 范围限制)

④ 用户自定义完整性

由业务规则决定

业务上的特殊规则

主键的两大铁律

1. 不能为 NULL

2. 不能重复 (每行主键值都唯一)

3. 一张表只能有一个主键 (但主键可以由多列组成 → 复合主键)

建表语句详解

创建表的语法骨架:

CREATE TABLE 表名 [

列名1 数据类型 [列级约束]

列名2 数据类型 [列级约束],

...

[表级约束]

) ENGINE = InnoDB DEFAULT CHARSET=utf8

-- ① 图书表

```
CREATE TABLE book (  
    BookID          CHAR(13)      NOT NULL,  
    BookName        VARCHAR(30)   NOT NULL,  
    Author          VARCHAR(20),  
    PublishingHouse VARCHAR(30),  
    Price           FLOAT(10,2),  
    BookNumber      INT,  
    PRIMARY KEY (BookID)  
) ENGINE=InnoDB DEFAULT CHARSET=
```

-- ② 读者表

```
CREATE TABLE reader (  
    ReaderID      CHAR(6)   NOT NU  
    ReaderName    CHAR(20)  NOT NU  
    Sex           CHAR(2)   NOT NU  
    Phone         CHAR(14),  
    PRIMARY KEY (ReaderID)  
) ENGINE=InnoDB DEFAULT CHARSET=
```

-- ③ 借阅表 (注意复合主键!)

```
CREATE TABLE borrow (  
    ReaderID      CHAR(6)   NOT NU  
    BookID        CHAR(13)  NOT NU  
    BorrowDate    DATETIME,  
    RefundDate    DATETIME,  
    PRIMARY KEY (ReaderID, BookID)  
) ENGINE=InnoDB DEFAULT CHARSET=gbk;
```

4.3 两种约束写法对比 数据... 4 artifacts

列级约束 (写在列定义后面) :

sql

```
BookID CHAR(13) PRIMARY KEY -- 主键写在列后  
ReaderID CHAR(6) UNIQUE KEY -- 唯一键  
Sex CHAR(2) NOT NULL DEFAULT '男' -- 非空+默认值
```

表级约束 (写在所有列定义之后, 单独一行) :

sql

```
CREATE TABLE borrow (  
    ReaderID CHAR(6),  
    BookID   CHAR(13),  
    ...,  
    PRIMARY KEY (ReaderID, BookID),  
    FOREIGN KEY (ReaderID) REFERENCES reader(ReaderID),  
    FOREIGN KEY (BookID) REFERENCES book(BookID)  
);
```

★ 铁律: 复合主键 (多列组成的主键) 必须用表级约束写!

五、修改表结构

5.1 加列 ADD

加一列: (默认加在最后)

```
ALTER TABLE book ADD COLUMN PublicationDate DATE;
```

加在某一列之后

```
ALTER TABLE book ADD COLUMN PublicationDate DATE AFTER....
```

加在最前

```
ALTER TABLE book ADD COLUMN BookCode VARCHAR(10) FIRST;
```

删列

```
ALTER TABLE book DROP COLUMN BookISBN
```

删除

改列 (MODIFY 和 CHANGE)

MODIFY	只改类型/约束, 不改列名	MODIFY	列名	新类型 (新的约束)
CHANGE	只改类型 又改列名	CHANGE	旧列名	新列名 新类型 (新的约束)

eg: 只改类型

```
ALTER TABLE book MODIFY Price DECIMAL(10,2)
```

改类型 + 列名

```
ALTER TABLE book CHANGE PublicationDate PublicationYear YEAR  
NULL DEFAULT 2022
```

加/删默认值

ALTER TABLE book SET DEFAULT 'ISBN-978-7-121'
加默认值

ALTER TABLE book DROP DEFAULT
删默认值

5.5 加/删主键、外键

sql

-- 加主键

ALTER TABLE book ADD PRIMARY KEY (BookID);

-- 删主键

ALTER TABLE book DROP PRIMARY KEY;

-- 加外键

ALTER TABLE borrow

ADD CONSTRAINT fk_reader

FOREIGN KEY (ReaderID) REFERENCES reader(ReaderID);

-- 删外键

ALTER TABLE borrow DROP FOREIGN KEY fk_reader;

重命名表

ALTER TABLE book RENAME TO books
= RENAME TABLE book TO books

删表

DROP TABLE book

DROP TABLE IF EXISTS book

六、外键的“删除/更新”策略 (★高频考点)

当外键关联的“父表”中数据被删除时，“子表”怎么办？4种策略：

sql

```
FOREIGN KEY (ReaderID) REFERENCES reader(ReaderID)
ON DELETE {CASCADE | RESTRICT | SET NULL | NO ACTION}
ON UPDATE {CASCADE | RESTRICT | SET NULL | NO ACTION}
```

策略	含义
CASCADE	级联：父表删/改，子表跟着删/改
RESTRICT	拒绝：子表还在用，父表禁止删/改（默认）
SET NULL	置空：父表删/改后，子表外键设为 NULL
NO ACTION	与 RESTRICT 类似



七、查看表结构的命令

SHOW TABLES 查看数据库中所有表

DESC 表名 查看表结构

DESCRIBE 表名 同上

SHOW CREATE TABLE 表名 查看完整CREATE语句

复制表：① 完全复制结构+数据

CREATE TABLE 新表名 AS SELECT * FROM 旧表名
(只复制表定义和数据，主键 外键 索引全去)

② 复制完整结构 (保留所有约束，但不复制数据)

CREATE TABLE 新表名 LIKE 旧表名

③ 只写结构不写数据

CREATE TABLE 新表名 AS SELECT * FROM 旧表名 WHERE 1=0;
(WHERE 1=0 永远为假 所以只复制结构没有数据)

2. AUTO_INCREMENT 自增

```
sql
CREATE TABLE student (
  StuID INT PRIMARY KEY AUTO_INCREMENT, --
  StuName VARCHAR(20) NOT NULL
);

INSERT INTO student(StuName) VALUES('张三'); -- Stu:
INSERT INTO student(StuName) VALUES('李四'); -- Stu:
```

StuID 自动=1
StuID 自动=2

铁律:

- 只能用在主键或唯一键列上
- 一张表只能有一个自增列
- 必须是整型

3. CHECK 约束

```
sql
CREATE TABLE student (
  Age INT CHECK (Age ≥ 0 AND Age ≤ 150),
  Sex CHAR(2) CHECK (Sex IN ('男', '女'))
);
```

★ DELETE / TRUNCATE / DROP

命令	删什么	删结构	能回收	速度	自增重置
DROP TABLE	整张表	X	X	快	
TRUNCATE TABLE	清空所有数据	✓	X	极快	归为1
DELETE FROM	数据 (可带WHERE)	✓	✓ (有事务)	慢	X

DROP 桌子搬走
TRUNCATE TABLE 桌子底下东西全扫
DELETE 一件件挑着扔

第四章

★ INSERT 插入数据

① 标准写法 (指定列)

```
INSERT INTO book (BookID, BookName, Author, Price, BookNumber)
VALUES ('987695321', 'MySQL数据库', '张伟', 68, 10)
```

往book表中插入一行 BookID=...

② 不指定列名 (要按照建表顺序写全所有值)

```
INSERT INTO book
VALUES ('987695321', 'MySQL数据库', '张伟', 68, 10)
```

(顺序必须和建表时一致, 少一个值就报错)

③ 插入多条记录 (一次插入多行)

```
INSERT INTO reader (ReaderID, ReaderName, Sex, Phone) VALUES
('1', '李一', '男', '11231597'),
('2', '王五', '男', '112314878'),
('3', '刘三', '男', '112354897');
```

④ SET 语法 (不常考)

```
INSERT INTO book SET BookID='978712432', BookName=''
```

PS: 字符串日期必须用单引号'' 数字不用

NULL不加引号 INSERT ... VALUES (... , NULL)
VALUES更标准

2. UPDATE ☆

基本语法 UPDATE 表名

SET 列1=值1 ; 列2=值2,

WHERE 条件

eg:

例子: 修改某本书的价格

sql

```
UPDATE book
SET Price = 69
WHERE BookID = '9787517071333';
```

例子: 同时改多列

sql

```
UPDATE book
SET Price = 69, BookNumber = 15
WHERE BookID = '9787517071333';
```

☆☆ UPDATE 的坑

不写 WHERE 会取整张表

UPDATE student SET s_name = '张三';

结果所有人姓名都叫张三

例子: 使用函数动态赋值

sql

```
-- 读者归还书, 把归还日期改为今天
UPDATE borrow
SET RefundDate = CURDATE()
WHERE ReaderID = '112219' AND BookID = '978730253155';
```

★ CURDATE() 返回当前日期 (年月日); NOW() 返回当前日期+时间。

3. DELETE

基本语法

DELETE FROM 表名 WHERE 条件;

例子

sql

```
-- 删除"算法分析与设计"这本书
DELETE FROM book WHERE BookName = '算法分析与设计';

-- 删除所有价格低于50的书
DELETE FROM book WHERE Price < 50;

-- 删除整张表所有记录 (保留表结构)
DELETE FROM book;
```

☆和UPDATE一样 不写WHERE删除整张表

eg: DELETE FROM student
整张表数据都删了

2.4 三大操作语法对比表

操作	关键字	语法骨架
插入	INSERT INTO	INSERT INTO 表(列) VALUES(值);
修改	UPDATE	UPDATE 表 SET 列=值 WHERE 条件
删除	DELETE FROM	DELETE FROM 表 WHERE 条件;

2.5 必背的常用日期函数

函数	返回	例子
CURDATE()	当前日期	2026-06-11
NOW()	当前日期+时间	2026-06-11 14:30:00
CURTIME()	当前时间	14:30:00
YEAR(日期)	提取年份	2026

第五章

表记录的查询

一. SELECT 的核心结构

SELECT [DISTINCT]	列名	选什么列
FROM 表名		从哪个表
[WHERE 条件]		过滤行
[GROUP BY 分组列]		分组
[HAVING 分组条件]		分组后过滤
[ORDER BY 排序列 ASC DESC]		排序
[LIMIT 数量]		限制条数

顺序: (SELECT → FROM → WHERE → GROUP → HAVING → ORDER → LIMIT)

二. 最基础的查询

1.

SELECT * FROM book; 查所有列

SELECT BookName, Price FROM book 查指定列
列名

2. DISTINCT: 去掉重复行

SELECT DISTINCT Publishing House FROM book
(SELECT 默认值 ALL 不去重, DISTINCT 去重)

3. 给列起别名 AS

SELECT Bookname AS '书名', Price AS '价格' FROM book;
(AS 可省略)

4. 列做计算

SELECT BookName, Price, Price * 0.9 AS '折后价' FROM book
(将每本书打9折后价格)

三. WHERE 过滤条件

条件类型	写法	例子
比较	= < > < >= <=	WHERE Price > 60
范围	BETWEEN ... AND ...	WHERE Price BETWEEN
集合	IN (... , ...)	WHERE PublishingHouse
模糊	LIKE 配%和_	WHERE BookName LIKE
空值	IS NULL / IS NOT NULL	WHERE RefundDate IS
逻辑	AND OR NOT	WHERE A=1 AND B=2

★ 原题考点2: $\text{Score} \geq 60$ AND $\text{Score} \leq 69$ 等价于 Score BETWEEN 60 AND 69.

四. ORDER BY 排序 (ASC 升, DESC 降)

1. 单列升序

SELECT * FROM book ORDER BY Price 升序/ASC

2. 单列降序

SELECT * FROM book ORDER BY Price DESC;
降

3. 多列排序, 先按出版社升序再价格降序

SELECT * FROM book
ORDER BY PublishingHouse ASC, Price DESC.

五. LIMIT 分页

LIMIT 数量 - 取前几条

LIMIT 偏移, 数量 - 跳过几条, 再取几条

LIMIT 数量 OFFSET 偏移 - 内行写法

eg: SELECT * FROM city LIMIT 5, 10

跳过前5条, 取 6-15条

LIMIT m, n 中 m 是跳过的数量

六. 聚合函数 把多行汇总

六、聚合函数 (5. 数... 10 artifacts 考)

聚合 = “把多行汇总成一个值”

函数	作用	例子
COUNT(*)	总行数 (含 NULL)	SELECT COUNT(*) FROM
COUNT(列名)	该列非 NULL 的行数	SELECT COUNT(Refund
SUM(列名)	求和	SELECT SUM(Price) F
AVG(列名)	平均值	SELECT AVG(Price) F
MAX(列名)	最大值	SELECT MAX(Price) F
MIN(列名)	最小值	SELECT MIN(Price) F

⚠ COUNT 的坑 (必考)

sql

```
-- borrow 表里有 RefundDate 为 NULL 的 (还没归还的书)
SELECT COUNT(*) FROM borrow; -- 所有借阅记
SELECT COUNT(RefundDate) FROM borrow; -- 已归还的记
```

} FROM book

七、GROUP BY 分组 (★★★ 必考)

用法：按某列分组后做聚合

sql

```
-- 按性别分组，统计男女各多少人
SELECT Sex, COUNT(*) FROM reader GROUP BY Sex;
-- 结果：
-- 男 | 3
-- 女 | 3
```

sql

每个出版社的平均价格

```
ouse, AVG(Price) FROM book GROUP BY PublishingHouse;
```

🎯 黄金搭档：GROUP BY + HAVING

HAVING 用于分组后的过滤（WHERE 是分组前过滤）。

sql

```
-- 统计每个出版社的图书数，只看图书数 > 2 的
SELECT PublishingHouse, COUNT(*) AS '数量'
FROM book
GROUP BY PublishingHouse
HAVING COUNT(*) > 2;
```

⚠️ WHERE 和 HAVING 的区别（高频考点）

	WHERE	HAVING
过滤时机	分组前	分组后
能用聚合函数吗	❌ 不能	✅ 能
位置	FROM 后面	GROUP BY 后面

CURRENT() 当前日期

NOW() 当前日期 + 时间

DATE_DIFF(后, 前) 相差天数

TIMESTAMPDIFF(单位, 前, 后) 两个日期相差多少

CURTIME() 当前时间

YEAR(NOW()), MONTH(NOW()), DAY(NOW()) 提取年月日

DATE_FORMAT(NOW(), %Y-%m-%d) 时间转为字符串

STR_TO_DATE('2024-01-01', %Y-%m-%d) 字符串转时间

DAY 5 多表连接

四种连接类型

1. 交叉连接 CROSS JOIN (笛卡尔积)

最暴力的连接: 每张表的每行 $\times$ 另一张表的每行

```
SELECT * FROM reader CROSS JOIN borrow;
```

2. 内连接 INNER JOIN ☆☆

只保留两边都能匹配的行。

sql

-- 查"借阅记录 + 对应的书名"

```
SELECT borrow.ReaderID, borrow.BookID, book.BookName  
FROM borrow INNER JOIN book  
ON borrow.BookID = book.BookID;
```

翻译: 把 borrow 和 book 拼起来, 条件是两张表的 BookID 相同。

INNER JOIN 的核心语法

sql

```
SELECT 列  
FROM 表1 INNER JOIN 表2  
ON 表1. 外键列 = 表2. 主键列;
```

(INNER 可选) ON后面是连接条件 (通常外键 = 主键)
(列名重复写成表名.列名) eg: borrow.BookID

3. 外连接: LEFT / RIGHT JOIN ★★

外连接 = 内连接 + 一边未匹配的所有行 (没匹配上的)

LEFT JOIN (左外连接): 保留左表所有行

sql

```
-- 查所有读者, 以及他们的借阅记录 (没借过书的读者也要显示)
SELECT reader.ReaderName, borrow.BookID
FROM reader LEFT JOIN borrow
ON reader.ReaderID = borrow.ReaderID;
```

结果示意:

ReaderName	BookID
刘雨轩	9787121419111
李嘉欣	9787111636222
白浩杰	NULL ← 这个人没借过书, 但也显示

RIGHT JOIN (右外连接): 保留右表所有行

sql

```
SELECT reader.ReaderName, borrow.BookID
FROM reader RIGHT JOIN borrow
ON reader.ReaderID = borrow.ReaderID;
```

199%都用 LEFT JOIN, 把没匹配上的显示出来

三种连接对比图

4. 三种连接对比图

假设 reader 有 A、B、C 三个读者, borrow 里只有 A、B 的借阅记录:

Code

```
INNER JOIN:  A, B          ← 只保留两边都有的
LEFT JOIN:   A, B, C       ← 左表所有, C没匹配显示 NULL
RIGHT JOIN:  A, B          ← 右表所有
CROSS JOIN:  A×全部, B×全部, C×全部 ← 暴力组合
```


三、三张表连接 (★原题考点)

要查“读者姓名 + 书名 + 借阅日期”，需要连接 3 张表：

```
sql

SELECT
    reader.ReaderID,
    reader.ReaderName,
    book.BookName,
    borrow.BorrowDate
FROM borrow
INNER JOIN book    ON borrow.BookID = book.BookID
INNER JOIN reader  ON borrow.ReaderID = reader.ReaderID
```

- 三表：1. 选定表：通常是带外键的表（这里 borrow 连 reader (ReaderID) 和 book (BookID)）
2. 一次连一张，每次写一个 ON 条件
3. 列名用表名列名消歧义

四、表别名 (让 SQL 更短)

```
sql

-- 不用别名：又长又啰嗦
SELECT borrow.BookID, book.BookName FROM borrow INNER JOIN book ON borrow.BookID = book.BookID;

-- 用别名 AS：简洁清爽
SELECT b.BookID, k.BookName
FROM borrow AS b INNER JOIN book AS k
ON b.BookID = k.BookID;

-- AS 可以省略
SELECT b.BookID, k.BookName
FROM borrow b INNER JOIN book k
ON b.BookID = k.BookID;
```

★ 表别名后，全句都要用别名（不能再写完整表名）。

子查询 (SQL 套路)

1. 标量子查询 (返回 1 个值)

```
sql
-- 查询高于平均定价的图书
SELECT * FROM book
WHERE Price > (SELECT AVG(Price) FROM book);
--           |               子查询返回一个数           |
```

工作机制:

1. 先算子查询 `SELECT AVG(Price) FROM book` → 返回比如 70.5
2. 再执行外层 → `WHERE Price > 70.5`

2. IN 子查询 (返回多个值的列表)

```
sql
-- 查询没有借阅过任何图书的读者
SELECT * FROM reader
WHERE ReaderID NOT IN (SELECT DISTINCT ReaderID FROM borrow);
--           |               子查询返回所有借过书的读者号           |
```

从 reader 表 从 borrow 表找

翻译: 找出 reader 表里那些 ReaderID 不在 borrow 表中出现过的人。

★ IN 和 NOT IN 必考——子查询返回多个值时用。

3. EXISTS 子查询 (判断是否存在)

返回 TRUE/FALSE

```
sql
-- 如果存在书号为 9787121198666 的书, 就查询借阅这本书的所有读者
SELECT * FROM borrow
WHERE EXISTS (SELECT * FROM book WHERE BookID = '9787121198666');
--           |               子查询返回一个布尔值           |
```

TRUE/FALSE

子查询位置三大类

位置

WHERE 子句中

FROM 子句中 (当临时表)

SELECT 子句中

例子

WHERE Price > (SELECT AVG(Price) ...)

FROM (SELECT * FROM book) AS a

SELECT name, (SELECT MAX(score) ...) FROM ...

六、子查询 vs 连接查询 (很多题两者都行)

例: 查询借阅过 BookID=9787121198666 的读者信息

方法1: 用 IN 子查询

sql

```
SELECT * FROM reader
WHERE ReaderID IN (
    SELECT ReaderID FROM borrow WHERE BookID = '9787121198666'
);
```

方法2: 用 INNER JOIN

sql

```
SELECT reader.* FROM reader
INNER JOIN borrow ON reader.ReaderID = borrow.ReaderID
WHERE borrow.BookID = '9787121198666';
```



★ 两种都对, 考试推荐用 JOIN (更高级、更清晰)。

二、数据类型 (Data Type)

1. 五大类数据类型

MySQL 数据类型按用途分 5 大类:

大类	代表类型	用途
1 整型	INT TINYINT BIGINT	整数 (年龄、数量)
2 浮点/定点	FLOAT DOUBLE DECIMAL	小数 (价格、工资)
3 字符串	CHAR VARCHAR TEXT	文字
4 日期时间	DATE TIME DATETIME YEAR	时间
5 复合类型	SET ENUM	枚举/集合

Day 7

一. 索引

索引 = 书的目录

作用: 加快查询速度

优点: ① 加快 SELECT 查询速度
② 让排序更快

缺点: ① 占用额外的存储空间
② 降低 INSERT / UPDATE / DELETE 速度

3. 索引的 4 种类型

类型	关键字	特点
普通索引	INDEX	没有限制, 单纯加速
唯一索引	UNIQUE INDEX	<u>索引列的值不能重复</u>
主键索引	PRIMARY KEY	<u>主键自动创建</u> (不需要手动)
全文索引	FULLTEXT INDEX	<u>用于长文本搜索</u>

4. 索引语法 (三件套)

① 创建索引

CREATE [UNIQUE] INDEX 索引名 ON 表名 (列名 [ASC / DESC])

② 查看索引

SHOW INDEX FROM 表名

③ 删除索引

DROP INDEX 索引名 ON 表名

★ 关键记忆:

- UNIQUE INDEX = 唯一索引 (列值不能重复)
- INDEX 名 ON 表(列1, 列2) = 组合索引 (多列一起)
- (列名(N)) = 只为列的前 N 个字符建索引
- ASC/DESC = 升序/降序

二. 视图

视图 = 虚拟表

(它本身不存数据, 有的是一条 SELECT 语句, 每次查视图, 相当于执行那条 SELECT 语句)

2. 视图的核心特性 (★★★★ 必考)

🔴 5 个必考点:

1. ✓ 视图是虚表 (不存数据)
2. ✓ 数据来自基表或视图 (视图可以套视图)
3. ✓ 使用视图能简化查询 (复杂查询封装成视图, 下次直接查视图)
4. ✗ 视图不能加快查询速度 (陷阱! 很多人以为视图加速, 其实不会)
5. ✗ 不是所有视图都能 UPDATE (含聚合函数、GROUP BY 的视图不可更新)

3. 视图语法

① 创建视图

CREATE [OR REPLACE] VIEW 视图名 [列名1, 列名2, ...]

AS SELECT 语句

[WITH CHECK OPTION] — 每次插入数据时, 必须满足视图 WHERE 条件

② 查询视图 (像查表一样)

SELECT * FROM 视图名

③ 删除视图

DROP VIEW 视图名

如果视图已有, 就替换

CASCADDED (级联)
LOCAL (本地)

三、触发器

触发器 = 自动执行的 SQL "程序"

它"埋伏"在某张表上, 当你对该表做了 INSERT / UPDATE / DELETE 时
自动跑一段预先写好的代码

2. 触发器的核心知识 (★★★★ 必考)

考点	内容
3 种触发事件	INSERT / UPDATE / DELETE (不支持 SELECT)
触发时机	BEFORE (事件前) / AFTER (事件后)
临时表	NEW (新值, 用于 INSERT/UPDATE) / OLD (旧值, 用于 UPDATE / DELETE)
触发器不能修改	只能删了重建! (DROP + CREATE)
同一时机限制	同一张表的同一时机 (如 BEFORE UPDATE) 只能有一个触发器

3. NEW 和 OLD 临时表

操作	NEW (新值)	OLD (旧值)
INSERT	✓ 可用	✗ 不可用
UPDATE	✓ 可用	✓ 可用
DELETE	✗ 不可用	✓ 可用

记忆:

- INSERT 是新增 → 只有"新"值, 所以只能用 NEW
- DELETE 是删除 → 只有"旧"值, 所以只能用 OLD
- UPDATE 是修改 → 新旧都有, 都能用

4. 触发器 语法

```
CREATE TRIGGER 触发器名 { BEFORE | AFTER } { INSERT / UPDATE /  
DELETE }  
ON 表名 FOR EACH ROW  
BEGIN  
    — 触发器要执行的 SQL 语句  
END;
```

5. 实战例子（原题练习1）

需求：当 reader 表删除一条记录时，自动把这条记录复制到 de_reader 表（做备份）。

sql



```
CREATE TRIGGER tr_reader_delete AFTER DELETE
ON reader FOR EACH ROW
BEGIN
    INSERT INTO de_reader(ReaderID, ReaderName, Sex,
        VALUES(OLD.ReaderID, OLD.ReaderName, OLD.Sex, OL
END;
```

🎯 关键点：

- AFTER DELETE —— 删除之后触发
- OLD.ReaderID —— 引用被删除的旧记录的列
- 因为是 DELETE，所以只能引用 OLD（不能用 NEW）

6. 删除触发器

DROP TRIGGER IF EXISTS 触发器名

Day 8

神经网络基础

变量