MySQL复习

1. 数据库基础

1.1 核心概念 ⭐

> 最基础也最重要的概念：

- **数据库(DB)**：长期存储在计算机内、有组织、可共享的数据集合
- **数据库管理系统(DBMS)**：用于管理数据库的软件系统
- **数据库系统(DBS)**：包含数据库、数据库管理系统、数据库管理员和用户的完整系统
- **重要关系**：DBS包含DB和DBMS ⭐

1.2 数据库三级模式 ⭐ ⭐

> 数据库的三级模式结构是数据库的框架：

- **外模式**：用户视图级别，一个数据库可以有多个外模式
- **模式**：数据库逻辑级别，一个数据库只有一个模式
- **内模式**：数据库物理级别，一个数据库只有一个内模式
- **两级映像**：
  - 外模式/模式映像：保证数据的逻辑独立性
  - 模式/内模式映像：保证数据的物理独立性

1.3 数据模型 ⭐

> 数据模型是对现实世界数据特征的抽象：

- **层次模型**：树形结构，只能表示1:n关系
- **网状模型**：网络结构，可以表示m:n关系
- **关系模型**：二维表（最常用） ⭐
- **对象关系模型**：面向对象
- 数据模型的三要素：
  1. 数据结构：描述数据类型、内容
  2. 数据操作：增删改查
  3. 数据约束：保证数据的正确性、有效性、相容性

1.4 关系数据库基础 ⭐

> 理解这些概念对学习MySQL至关重要：

- **关系**：一个关系就是一张表
- **元组**：表中的一行数据
- **属性**：表中的一列
- **键的概念** ⭐ ⭐
  - 超键：能唯一标识记录的属性集合

- 候选键：最小的超键
- 主键：被选作记录标识的候选键
- 外键：关联其他表的主键的字段

1.5 数据库完整性 ⭐⭐

> 确保数据的正确性、有效性和一致性：

1. **实体完整性**：
   - 主键非空且唯一
   - 例：学号是主键，不能重复且不能为空
2. **参照完整性**：
   - 外键值必须存在于主表中或为空
   - 例：成绩表的学号必须存在于学生表中
3. **用户定义完整性**：
   - CHECK约束：age > 0
   - 默认值：gender DEFAULT 'M'
   - 非空约束：name NOT NULL

1.6 基本表与视图的区别 ⭐

> 经常考察的重点：

1. **基本表**：
   - 实际存储数据的表
   - 独立存在
   - 可以直接更新
2. **视图**：
   - 虚拟表，不存储数据
   - 依赖基本表
   - 一般用于查询，更新有限制

1.7 关系型vs非关系型数据库 ⭐

> 数据库的两大阵营：

- **关系型(SQL)**：
  - MySQL、Oracle、SQL Server
  - 使用表格存储数据
  - 支持ACID特性
  - 适合事务处理
- **非关系型(NoSQL)**：
  - MongoDB、Redis、Cassandra
  - 使用多种数据结构存储
  - 高性能、高可扩展性

- 适合大数据应用

2. MySQL基本操作

2.1 数据库操作 ⭐

```sql
-- 基本的数据库操作
CREATE DATABASE db_name;  -- 创建数据库
SHOW DATABASES;           -- 显示所有数据库
USE db_name;              -- 使用数据库
DROP DATABASE db_name;    -- 删除数据库
```

2.2 表操作 ⭐

```sql
-- 创建表（重要）
CREATE TABLE student (
    id INT PRIMARY KEY,          -- 主键约束
    name VARCHAR(50) NOT NULL,   -- 非空约束
    email VARCHAR(100) UNIQUE,   -- 唯一约束
    age INT CHECK (age > 0),     -- 检查约束
    gender CHAR(1) DEFAULT 'M', -- 默认值约束
    class_id INT,                -- 外键字段
    FOREIGN KEY (class_id) REFERENCES class(id)  -- 外键约束
);

-- 修改表
ALTER TABLE table_name ADD column_name datatype;    -- 添加列
ALTER TABLE table_name DROP column_name;           -- 删除列
ALTER TABLE table_name MODIFY column_name new_datatype;  -- 修改列定义
```

3. SQL语句分类 ⭐ ⭐

> 必须掌握的SQL分类：

1. **DDL（数据定义）**：
   - CREATE：创建
   - DROP：删除
   - ALTER：修改
   - TRUNCATE：清空（删除所有行）
2. **DML（数据操作）**：
   - INSERT：插入
   - UPDATE：更新
   - DELETE：删除（可带WHERE条件）
3. **DQL（数据查询）**：
   - SELECT：查询
   - WHERE：条件
   - GROUP BY：分组
   - HAVING：分组后过滤

- ORDER BY：排序

4. **DCL（数据控制）**：
   - GRANT：授权
   - REVOKE：撤销权限

4. 查询专题 ⭐⭐⭐

4.1 基础查询

```sql
-- 基本SELECT语句结构（记住这个顺序）⭐
SELECT [DISTINCT] 列名    -- DISTINCT去重
FROM 表名
[WHERE 条件]              -- 行过滤
[GROUP BY 分组]           -- 分组
[HAVING 分组后条件]       -- 组过滤
[ORDER BY 排序]           -- 排序
[LIMIT 限制行数];         -- 限制结果数量

-- WHERE子句条件
age > 20                  -- 比较运算
age BETWEEN 20 AND 30     -- 范围
name LIKE '张%'           -- 模糊匹配
id IN (1,2,3)             -- 集合
score IS NULL             -- 空值判断
```

4.2 连接查询 ⭐⭐

```sql
-- 内连接（最常用）
SELECT s.name, c.class_name
FROM student s
INNER JOIN class c ON s.class_id = c.id;

-- 左外连接（保留左表所有数据）
SELECT s.name, c.class_name
FROM student s
LEFT JOIN class c ON s.class_id = c.id;

-- 右外连接（保留右表所有数据）
SELECT s.name, c.class_name
FROM student s
RIGHT JOIN class c ON s.class_id = c.id;

-- 全外连接（MySQL通过UNION实现）
SELECT s.name, c.class_name
FROM student s
LEFT JOIN class c ON s.class_id = c.id
UNION
SELECT s.name, c.class_name
FROM student s
RIGHT JOIN class c ON s.class_id = c.id;
```

4.3 子查询 ⭐⭐

```sql
-- IN子查询
SELECT name
FROM student
WHERE class_id IN (
    SELECT id
    FROM class
    WHERE grade = '高一'
);

-- EXISTS子查询
SELECT name
FROM student s
WHERE EXISTS (
    SELECT 1
    FROM score sc
    WHERE sc.student_id = s.id
    AND sc.score > 90
);

-- 相关子查询（重要）
SELECT name, score
FROM student s
WHERE score > (
    SELECT AVG(score)
    FROM student
    WHERE class_id = s.class_id
);
```

4.4 复杂查询示例 ⭐⭐

```sql
-- 查询每个班级的前三名
SELECT *
FROM (
    SELECT
        name,
        class_id,
        score,
        DENSE_RANK() OVER (
            PARTITION BY class_id
            ORDER BY score DESC
        ) as rank
    FROM student
) t
WHERE rank <= 3;

-- 查询连续三天都有登录的用户
SELECT DISTINCT a.user_id
FROM login_logs a
JOIN login_logs b ON a.user_id = b.user_id
    AND DATEDIFF(b.login_date, a.login_date) = 1
JOIN login_logs c ON b.user_id = c.user_id
    AND DATEDIFF(c.login_date, b.login_date) = 1;
```

5. 锁和事务 ⭐⭐⭐

5.1 锁的分类

1. **按操作分**：
   - 读锁（共享锁）：允许多个事务同时读
   - 写锁（排他锁）：只允许一个事务写
2. **按粒度分**：
   - 表锁：锁定整张表
   - 行锁：锁定单行数据
   - 页锁：介于表锁和行锁之间

5.2 ACID特性

- **原子性(Atomicity)**：事务是不可分割的工作单位 ⭐
- **一致性(Consistency)**：事务前后数据保持一致
- **隔离性(Isolation)**：事务之间互不干扰
- **持久性(Durability)**：事务一旦提交永久保存

5.3 事务隔离级别 ⭐

> 解决并发访问问题：

隔离级别	脏读	不可重复读	幻读
读未提交	是	是	是
读已提交	否	是	是
可重复读	否	否	是
串行化	否	否	否

5.4 事务控制

```sql
-- 开启事务
START TRANSACTION;

-- 创建保存点
SAVEPOINT point1;

-- 回滚到保存点
ROLLBACK TO SAVEPOINT point1;

-- 提交事务
COMMIT;

-- 回滚事务
ROLLBACK;
```

6. 索引优化 ⭐⭐

6.1 索引分类

- **普通索引**：加速查询
- **唯一索引**：加速查询 + 唯一性约束
- **主键索引**：聚簇索引，数据按主键顺序存储
- **复合索引**：多列组成，遵循最左前缀原则
- **全文索引**：用于文本搜索

6.2 索引使用原则

1. **建索引**：
   - WHERE、ORDER BY、GROUP BY的列
   - 外键列
   - 经常作为范围查询的列
2. **不建索引**：
   - 数据量小的表
   - 经常更新的列
   - 取值很少的列（如性别）

6.3 SQL优化 ⭐⭐

1. **索引优化**：

```sql
-- 使用索引列作为条件
SELECT * FROM user WHERE id = 1;  -- 走索引
SELECT * FROM user WHERE name LIKE 'zhang%';  -- 走索引
SELECT * FROM user WHERE name LIKE '%zhang';  -- 不走索引
```

2. **JOIN优化**：

```sql
-- 小表驱动大表
SELECT * FROM small_table s
JOIN big_table b ON s.id = b.id;

-- 使用索引列作为连接条件
SELECT * FROM table1 t1
JOIN table2 t2 ON t1.indexed_col = t2.indexed_col;
```

3. **子查询优化**：

```sql
-- 使用JOIN替代IN子查询
SELECT * FROM table1
WHERE id IN (SELECT id FROM table2);
-- 优化为
SELECT DISTINCT table1.*
FROM table1
JOIN table2 ON table1.id = table2.id;
```

7. 视图和存储过程 ⭐

7.1 视图

```sql
-- 创建视图
CREATE VIEW v_student AS
SELECT s.*, c.class_name
FROM student s
JOIN class c ON s.class_id = c.id;

-- 视图的优点：
-- 1. 简化复杂查询
-- 2. 提供数据安全性
-- 3. 保持数据独立性
```

7.2 存储过程

```sql
DELIMITER //
CREATE PROCEDURE update_score(
    IN student_id INT,     -- 输入参数
    IN course_id INT,
    IN new_score INT,
    OUT result INT         -- 输出参数
)
BEGIN
    -- 变量声明
    DECLARE old_score INT;

    -- 异常处理
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    SET result = -1;

    -- 业务逻辑
    SELECT score INTO old_score
    FROM scores
    WHERE student_id = student_id
    AND course_id = course_id;

    IF old_score IS NULL THEN
        SET result = 0;
    ELSE
        UPDATE scores SET score = new_score
        WHERE student_id = student_id
        AND course_id = course_id;
        SET result = 1;
```

```
    END IF;
END //
DELIMITER ;
```

8. 数据库设计 ⭐⭐⭐

8.1 设计步骤

1. **需求分析**：理解业务需求
2. **概念设计**：绘制E-R图
3. **逻辑设计**：转换为关系模式
4. **物理设计**：定义数据库表结构
5. **实施**：创建数据库和表

8.2 E-R图 ⭐⭐

> 实体-联系图，描述数据的静态特征

- **矩形**：实体
- **椭圆**：属性（主键带下划线）
- **菱形**：联系
- **连线**：表示关系类型
  - 一对一(1:1)：—┼—
  - 一对多(1:n)：—┼—<
  - 多对多(m:n)：>—┼—<

8.3 关系模式转换规则

1. **一对一**：
   - 任选一个实体作为主表
   - 在另一个表中加入外键
2. **一对多**：
   - 在"多"的一方加入外键
3. **多对多**：
   - 创建中间关系表
   - 包含两个实体的主键作为联合主键

9. 规范化 ⭐⭐⭐

> 数据库设计的重要理论：

9.1 函数依赖

- **完全函数依赖**：学号+课程号 → 成绩
- **部分函数依赖**：学号+课程号 → 姓名（只依赖学号）
- **传递函数依赖**：学号 → 系名 → 系主任

9.2 范式

1. **第一范式(1NF)**:
   - 属性不可分
   - 例：地址应拆分为省、市、区
2. **第二范式(2NF)**:
   - 消除部分函数依赖
   - 例：拆分学生表和成绩表
3. **第三范式(3NF)**:
   - 消除传递函数依赖
   - 例：系名和系主任应该放在单独的表中

9.3 范式例子

原表:

```
学生选课表(学号，姓名，年龄，课程号，课程名，成绩，学院，院长)
```

函数依赖:

- 学号 → 姓名，年龄，学院
- 课程号 → 课程名
- 学号，课程号 → 成绩
- 学院 → 院长

转换为3NF:

```sql
-- 学生表
CREATE TABLE student (
    学号 CHAR(10) PRIMARY KEY,
    姓名 VARCHAR(20),
    年龄 INT,
    学院 VARCHAR(20)
);

-- 课程表
CREATE TABLE course (
    课程号 CHAR(10) PRIMARY KEY,
    课程名 VARCHAR(50)
);

-- 成绩表
CREATE TABLE score (
    学号 CHAR(10),
    课程号 CHAR(10),
    成绩 INT,
    PRIMARY KEY (学号, 课程号),
    FOREIGN KEY (学号) REFERENCES student(学号),
    FOREIGN KEY (课程号) REFERENCES course(课程号)
);
```

```sql

-- 学院表
CREATE TABLE college (
    学院 VARCHAR(20) PRIMARY KEY,
    院长 VARCHAR(20)
);
```

10. 常见考点 ⭐ ⭐ ⭐

10.1 查询类型

1. **基础查询**：
   - 单表条件查询
   - 分组统计查询
   - 排序和限制查询
2. **多表查询**：
   - 内连接、外连接
   - 子查询（IN、EXISTS）
   - 组合查询（UNION）
3. **复杂查询**：
   - 分组后过滤
   - 多表联合统计
   - 行转列查询

10.2 考试重点

1. **概念题**：
   - 三级模式两级映像
   - ACID特性
   - 数据库完整性
   - 规范化理论
2. **设计题**：
   - E-R图设计
   - 关系模式转换
   - 规范化设计
3. **SQL题**：
   - 多表连接查询
   - 复杂统计查询
   - 视图和存储过程

10.3 易错点

1. **完整性约束**：
    - 主键和唯一键的区别
    - 外键的参照完整性
2. **事务特性**：
    - ACID含义
    - 隔离级别的区别
3. **规范化**：
    - 函数依赖的判断
    - 范式的判断和转换

注意事项 ⭐

1. 答题技巧：
    - 概念题注意关键词
    - SQL题先写框架后填充
    - 设计题多画图理清关系
2. 实践建议：
    - 动手实践每个SQL例子
    - 理解概念间的联系
    - 多做例题和真题
3. 避免误区：
    - 不要死记硬背
    - 理解原理更重要
    - 注意实际应用场景