（关系型）数据库MySQL

1. 简要介绍关系型数据库

1. 作用：存数据
2. 优点：
   更安全
   利于数据查询和管理
   利于存储海量数据
   便于和编程语言结合进行控制
3. 存储介质：
   磁盘
   内存

2. 数据库的 CURD SQL 语句

Create（创建），Update（更新），Retrieve（读取），Delete（删除）

1. 创建数据库

```
CREATE DATABASE [IF NOT EXISTS] db_name [create_specification [, create_specification] ...]
create_specification:
[DEFAULT] CHARACTER SET charset_name
[DEFAULT] COLLATE collation_name
```

大写表示关键字
[] 是可选项
CHARACTER SET: 指定数据库采用的字符集
COLLATE: 指定数据库字符集的校验规则

> 查看系统默认字符集及校验规则

```
SHOW variables like 'character_set_database';
SHOW variables like 'collation_database';
```

> 查看数据库支持的字符集及校验规则

```
SHOW charset;
SHOW collation;
```

> 查看数据库创建语句

```
SHOW CREATE DATABASE db_name;
```

> 使用数据库

```
USE db_name;
```

> 修改数据库（字符集&校验规则）

```
ALTER DATABASE db_name [create_specification [, create_specification] ...]
```

> 删除数据库

```
DROP DATABASE [IF EXISTS] db_name
```

2. 创建表（最轻量化）

```
CREATE TABLE table_name (
    field1 datatype,
    field2 datatype,
    field3 datatype
) character set 字符集类型 collate 校验规则 engine 存储引擎;
```

不同的存储引擎，创建表的文件不一样

> 数据类型

一、数值类型
用于存储整数、小数等数值，分为**整数类型**、**浮点数类型**、**定点数类型**。
（超范围报错）

整数类型（精确整数）

类型	字节数	有符号范围（默认）	无符号范围（UNSIGNED）	适用场景
TINYINT	1	-128 ~ 127	0 ~ 255	状态（0/1）、年龄（小范围）等
SMALLINT	2	-32768 ~ 32767	0 ~ 65535	小范围计数（如订单状态码）
MEDIUMINT	3	-8388608 ~ 8388607	0 ~ 16777215	中等范围数值（如用户等级）
INT（INTEGER）	4	-2147483648 ~ 2147483647	0 ~ 4294967295	常用整数（如ID、数量）
BIGINT	8	-9223372036854775808 ~ 9223372036854775807	0 ~ 18446744073709551615	大整数（如超大ID、时间戳）

- 特点：存储精确整数，可通过 `UNSIGNED` 关键字指定无符号（仅正整数）。

浮点数类型（近似小数）

类型	字节数	范围（近似）	说明
FLOAT	4	约 ±3.402823466E+38（单精度）	可指定精度： `FLOAT(M,D)` ，M总位数，D小数位
DOUBLE	8	约 ±1.7976931348623157E+308（双精度）	精度高于FLOAT，同样支持 `DOUBLE(M,D)`

- 特点：存储近似小数（可能有精度丢失），适合非精确场景（如科学计算）。

定点数类型（精确小数）

类型	格式	说明	适用场景
DECIMAL	DECIMAL(M,D)	M：总位数（1_{65}），D：小数位数（$_{0}30$）	货币、精确计数

- 特点：存储精确小数（无精度丢失），内部以字符串形式存储，适合金额等需精确计算的场景。

二、字符串类型
用于存储文本、字符等数据，分为**固定长度**、**可变长度**、**长文本**等。

短字符串（CHAR / VARCHAR）

类型	存储方式	长度范围	特点	适用场景
CHAR	固定长度（不足补空格）	0 ~ 255 字符	查询速度快，浪费空间（固定长度）	长度固定的短文本（手机号、性别）
VARCHAR	可变长度（仅存实际内容）	0 ~ 65535 字符	节省空间，查询稍慢（需计算长度）	长度不固定的文本（姓名、描述）

- 注意： `VARCHAR` 实际存储上限受表的总字节限制（65535字节）。

长文本（TEXT / BLOB）
用于存储超长文本或二进制数据（如文章、图片）。

类型	存储内容	最大长度（约）	特点
TINYTEXT	短文本	255 字符	无默认值，适合短备注
TEXT	普通文本	65535 字符	适合文章段落
MEDIUMTEXT	中等长度文本	16MB	适合长文章、日志
LONGTEXT	超长文本	4GB	适合超大文本（如电子书）
TINYBLOB	二进制数据	255 字节	存储小二进制（如图标）
BLOB	二进制数据	65535 字节	存储普通二进制（如小图片）
MEDIUMBLOB	二进制数据	16MB	存储中等二进制（如图片、音频片段）
LONGBLOB	二进制数据	4GB	存储超大二进制（如视频、大型文件）

- 特点： `TEXT` 存储文本（字符）， `BLOB` 存储二进制（字节）；均无默认值，查询效率较低，建议避免频繁查询。

枚举与集合（ENUM / SET）

类型	格式	说明	适用场景
ENUM	ENUM('值1','值2',...)	只能从预定义列表中选**1个值**	性别（男/女）、状态（启用/禁用）
SET	SET('值1','值2',...)	可从预定义列表中选**多个值**（最多64个）	爱好（读书,运动,音乐）

- 特点：存储效率高（内部用数字编码），但修改预定义列表需ALTER TABLE。

三、日期和时间类型
用于存储日期、时间或时间戳。

类型	格式	范围	特点	适用场景
DATE	YYYY-MM-DD	1000-01-01 ~ 9999-12-31	仅存储日期	生日、注册日期
TIME	HH:MM:SS	-838:59:59 ~ 838:59:59	存储时间（可表示时间间隔，如负数）	时长、打卡时间
DATETIME	YYYY-MM-DD HH:MM:SS	1000-01-01 00:00:00 ~ 9999-12-31 23:59:59	日期+时间，**不受时区影响**	订单创建时间、事件发生时间
TIMESTAMP	YYYY-MM-DD HH:MM:SS	1970-01-01 00:00:01 ~ 2038-01-19 03:14:07	日期+时间，**受时区影响**（存储UTC）	时间戳（如最后修改时间）
YEAR	YYYY	1901 ~ 2155 或 0000	仅存储年份	出生年份、发布年份

- 关键区别： `DATETIME` 存储原始时间， `TIMESTAMP` 会随时区转换（插入时转UTC，查询时转当前时区）。

四、特殊类型

- **JSON**：MySQL 5.7+ 支持，用于存储JSON格式数据，可直接查询/修改JSON中的字段（如 `SELECT data->'$.name' FROM table` ），适合存储半结构化数据。
- **GEOMETRY**：用于存储空间数据（如点、线、多边形），支持空间索引和地理计算（较少用）。

五、选择原则

1. 最小化：选择能满足需求的最小类型（如年龄用 `TINYINT` 而非 `INT` ）。
2. 精确性：金额等需精确计算的场景用 `DECIMAL` ，避免 `FLOAT/DOUBLE` 。
3. 时区敏感：需跨时区的时间用 `TIMESTAMP` ，否则用 `DATETIME` 。
4. 字符串：短且固定长度用 `CHAR` ，长或可变用 `VARCHAR` ，超长用 `TEXT` 。

> 查看表结构

```
DESC table_name;
```

> 修改表结构

```
ALTER TABLE tablename ADD (column datatype [DEFAULT expr][,column datatype]...);
ALTER TABLE tablename MODIfy (column datatype [DEFAULT expr][,column datatype]...);
ALTER TABLE tablename DROP (column);
```

插入新字段后，对原来表中的数据没有影响

> 删除表

```
DROP TABLE [IF EXISTS] tablename;
```

> 表的常见约束
> 列描述： comment
> 空属性：允许为空 null （默认），不允许为空 not null
> 默认值： default value
> 主键： primary key （唯一的约束该字段里面的数据，不能重复，不能为空，一张表中最多只能有一个主键，主键所在的列通常是整数类型）
> 唯一键： unique （保证唯一性，可为空，空字段不进行唯一性比较）
> 自增： auto_increment
> 外键： foreign key （外键用于定义主表和从表之间的关系：外键约束主要定义在从表上，主表则必须是有主键约束或唯一键约束。定义外键后，外键列数据必须在主表的主键列存在或为null） -- 相关性绑定

追加主键

```
ALTER TABLE tablename ADD PRIMARY KEY (column);
```

删除主键

```
ALTER TABLE tablename DROP PRIMARY KEY;
```

复合主键

```
ALTER TABLE tablename ADD PRIMARY KEY (column1, column2);
```

创建唯一键

```
ALTER TABLE tablename ADD UNIQUE (column);
```

创建外键

```
ALTER TABLE tablename ADD FOREIGN KEY (column) REFERENCES tablename(column);
```

3. 插入数据

单行数据全列插入

```
INSERT INTO table_name (column1, column2, column3) VALUES (value1, value2, value3);
```

多行数据指定列插入

```
INSERT INTO table_name (column1, column2, column3) VALUES (value1, value2, value3), (value1, value2, value3);
```

4. 查询数据

```
SELECT column1, column2, column3 FROM table_name;
```

> 查询字段为表达式

```
SELECT column1 + column2 FROM table_name;
```

> 给查询结果起别名

```
SELECT column1 AS alias1, column2 AS alias2 FROM table_name;
```

> 去重

```
SELECT DISTINCT column1, column2 FROM table_name;
```

> where

```
SELECT column1, column2 FROM table_name WHERE column1 = value1;
```

运算符

- 比较运算符： `=` （NULL 不安全）、 `!=` 、 `>` 、 `<` 、 `>=` 、 `<=` 、 `<>` （不等于）、 `<=>` （NULL）
- 逻辑运算符： `AND` 、 `OR` 、 `NOT`
- 模糊查询： `LIKE` 、 `NOT LIKE`
- 范围查询： `BETWEEN` 、 `NOT BETWEEN`
- 空值查询： `IS NULL` 、 `IS NOT NULL`
- 集合查询： `IN` 、 `NOT IN`

> 排序

```
SELECT column1, column2 FROM table_name ORDER BY column1 [ASC|DESC];
```

ASC：升序（默认）
DESC：降序

> 分页

```
SELECT column1, column2 FROM table_name LIMIT start, count;
```

start：起始行（从0开始）
count：行数

5. 更新数据

```
UPDATE table_name SET column1 = value1, column2 = value2 WHERE condition;
```

6. 删除数据

```
DELETE FROM table_name WHERE condition;
```

7. 聚合函数

- 统计查询数据数量： `COUNT([DISTINCT])`
- 求和： `SUM(column)`
- 平均值： `AVG(column)`

- 最大值： `MAX(column)`
- 最小值： `MIN(column)`

8. 分组

```
SELECT column1, column2, column3 FROM table_name GROUP BY column1, column2, column3;
```

将查询结果中指定字段值相同的行归为一组，然后对每个组使用聚合函数（如 COUNT() 统计数量、SUM() 求和等）进行分析，从而快速得到分组后的统计结果

9. 函数

- 数学函数
  数学函数主要用于数值计算，涵盖取整、绝对值、幂运算、随机数等常见场景，以下为高频函数列表：

函数名	功能描述	语法示例	执行结果
`ABS(x)`	返回数值 x 的绝对值	`SELECT ABS(-15.6);`	15.6
`ROUND(x,n)`	对 x 进行四舍五入，保留 n 位小数（n 默认为 0，即取整）	`SELECT ROUND(123.456,2);`	123.46
`CEIL(x)`	向上取整（返回大于等于 x 的最小整数，同 `CEILING(x)` ）	`SELECT CEIL(5.1);`	6
`FLOOR(x)`	向下取整（返回小于等于 x 的最大整数）	`SELECT FLOOR(5.9);`	5
`MOD(x,y)`	返回 x 除以 y 的余数（同 `x % y` ，余数符号与 x 一致）	`SELECT MOD(10,3);`	1
`POWER(x,y)`	返回 x 的 y 次幂（同 `POW(x,y)` ）	`SELECT POWER(2,3);`	8
`SQRT(x)`	返回 x 的平方根（x 需非负）	`SELECT SQRT(16);`	4
`RAND()`	返回 0~1 之间的随机浮点数（若加参数 `RAND(N)` ，N 为种子，可生成固定序列）	`SELECT RAND();`	0.87654（示例）
`TRUNCATE(x,n)`	对 x 截断保留 n 位小数（不四舍五入，直接截取）	`SELECT TRUNCATE(123.456,2);`	123.45

- 日期函数
  日期函数用于处理 `DATE` 、 `TIME` 、 `DATETIME` 类型数据，包括获取当前时间、格式化、增减日期、计算间隔等，是业务中处理时间逻辑的核心：

函数名	功能描述	语法示例	执
`CURDATE()`	返回当前日期（格式： `YYYY-MM-DD` ，同 `CURRENT_DATE()` ）	`SELECT CURDATE();`	2024
`CURTIME()`	返回当前时间（格式： `HH:MM:SS` ，同 `CURRENT_TIME()` ）	`SELECT CURTIME();`	14:3
`NOW()`	返回当前日期时间（格式： `YYYY-MM-DD HH:MM:SS` ，同 `SYSDATE()` ）	`SELECT NOW();`	2024 14:3
`DATE_FORMAT(d,f)`	将日期 d 按格式 f 转换为字符串（f 为格式符，如 `%Y` =4位年， `%m` =2位月）	`SELECT DATE_FORMAT(NOW(),'%Y年%m月%d日');`	2024
`DATE_ADD(d,INTERVAL v unit)`	给日期 d 增加时间（v 为数值，unit 为单位：YEAR/MONTH/DAY/HOUR 等）	`SELECT DATE_ADD('2024-01-01', INTERVAL 1 MONTH);`	2024
`DATE_SUB(d,INTERVAL v unit)`	给日期 d 减少时间（用法同 DATE_ADD，方向相反）	`SELECT DATE_SUB('2024-01-01', INTERVAL 7 DAY);`	2023
`DATEDIFF(d1,d2)`	计算 d1 与 d2 的日期差（结果为 `d1 - d2` ，仅算日期部分，忽略时间）	`SELECT DATEDIFF('2024-05-20','2024-05-10');`	10

函数名	功能描述	语法示例	
`TIMESTAMPDIFF(unit,d1,d2)`	按 unit 单位计算 d2 与 d1 的时间差（unit：SECOND/MINUTE/HOUR/DAY/YEAR 等）	`SELECT TIMESTAMPDIFF(HOUR,'2024-05-20 10:00','2024-05-20 15:30');`	5.5
`YEAR(d)` / `MONTH(d)`	提取日期 d 中的年份/月份（类似还有 `DAY(d)`、`HOUR(t)`、`MINUTE(t)` 等）	`SELECT YEAR('2024-05-20');`	2024

- 字符串函数
  字符串函数用于处理 `CHAR`、`VARCHAR` 类型数据，包括拼接、截取、替换、大小写转换等，是文本数据处理的常用工具：

函数名	功能描述	语法示例	执行结果
`CONCAT(s1,s2,...)`	拼接多个字符串（若有一个参数为 NULL，结果整体为 NULL）	`SELECT CONCAT('My','SQL','教程');`	MySQL教程
`CONCAT_WS(sep,s1,s2,...)`	用分隔符 sep 拼接字符串（sep 不为 NULL，忽略 NULL 的参数）	`SELECT CONCAT_WS('-','2024','05','20');`	2024-05-20
`LENGTH(s)`	返回字符串 s 的字节数（注意：UTF8 编码下，1个中文占3字节，1个英文占1字节）	`SELECT LENGTH('MySQL教程');`	8（5英文+2中文=5+6=11？哦不对，MySQL是5字符，教程是2中文，UTF8下5*1* + 2*3*=11，之前示例错了，纠正：`SELECT LENGTH('MySQL教程');` 结果11）
`CHAR_LENGTH(s)`	返回字符串 s 的字符数（无论中英文，1个字符算1）	`SELECT CHAR_LENGTH('MySQL教程');`	7（5+2）
`SUBSTRING(s,pos,len)`	从字符串 s 的 pos 位置开始，截取 len 个字符（pos 从1开始，len 可选，默认到末尾，同 `SUBSTR(s,pos,len)`）	`SELECT SUBSTRING('MySQL教程',1,5);`	MySQL
`REPLACE(s,old,new)`	将字符串 s 中的 old 子串替换为 new 子串	`SELECT REPLACE('MySQL教程','MySQL','数据库');`	数据库教程
`UPPER(s)` / `LOWER(s)`	将字符串 s 转换为全大写/全小写（同 `UCASE(s)`、`LCASE(s)`）	`SELECT UPPER('mysql');`	MYSQL
`TRIM(s)`	去除字符串 s 两端的空格（`LTRIM(s)` 去左空格，`RTRIM(s)` 去右空格）	`SELECT TRIM('  MySQL  ');`	MySQL
`INSTR(s,sub)`	返回子串 sub 在字符串 s 中首次出现的位置（从1开始，未找到返回0）	`SELECT INSTR('MySQL教程','SQL');`	3
`LEFT(s,n)` / `RIGHT(s,n)`	从字符串 s 的左侧/右侧截取 n 个字符	`SELECT LEFT('MySQL教程',3);`	MyS
`LPAD(s,len,pad)` / `RPAD(s,len,pad)`	用 pad 字符将 s 填充到 len 长度（左填充/	`SELECT LPAD('123',5,'0');`	00123

函数名	功能描述	语法示例	执行结果
	右填充，s 超长则截断）		

- 其它常用函数
  此类函数为辅助性工具，涵盖条件判断、空值处理、系统信息获取等场景，在业务逻辑中高频使用：

函数名	功能描述	
`IF(expr,v1,v2)`	条件判断：若 expr 为 TRUE，返回 v1；否则返回 v2	`SELECT IF(10>5,'大于','小于');`
`IFNULL(v1,v2)`	空值处理：若 v1 不为 NULL，返回 v1；否则返回 v2（仅判断 v1 是否为 NULL）	`SELECT IFNULL(NULL,'空值');`
`CASE WHEN 条件1 THEN 结果1 WHEN 条件2 THEN 结果2 ... ELSE 默认结果 END`	多条件判断（类似编程语言的 if-else if-else）	`SELECT CASE WHEN score>=90 THEN 'A' WHEN s`
`ISNULL(expr)`	判断 expr 是否为 NULL：是则返回 1，否则返回 0（同 `expr IS NULL`）	`SELECT ISNULL(NULL);`
`USER()`	返回当前登录的 MySQL 用户名及主机（格式：`用户名@主机地址`）	`SELECT USER();`
`VERSION()`	返回当前 MySQL 服务器的版本号	`SELECT VERSION();`
`COUNT(expr)`	统计非 NULL 值的数量（聚合函数，常与 `GROUP BY` 搭配）	`SELECT COUNT(*) FROM student;`
`MAX(col) / MIN(col)`	求某列的最大值/最小值（聚合函数，适用于数值、日期、字符串类型）	`SELECT MAX(score) FROM student;`

10. 复合查询

- 多表查询（笛卡尔积）

```
SELECT column1, column2, column3 FROM table1, table2;
```

11. 事务

- 事务（Transaction）：数据库中一组不可分割的操作单元，需满足 ACID 特性（原子性、一致性、隔离性、持久性），例如 “转账”(扣钱 + 加钱）就是一个典型事务。
- 隔离性（Isolation）：多个事务并发执行时，一个事务的操作不能被其他事务 “干扰”。若隔离性不足，就会出现脏读、不可重复读、幻读。
- 事务满足属性：
  - 原子性：要么全部成功，要么全部失败
  - 一致性：事务执行前后，数据保持一致性
  - 隔离性：事务之间相互隔离，互不干扰
  - 持久性：事务一旦提交，对数据库的修改是永久的
- 并发场景与问题：
  - RR：无问题
  - RW：脏读、不可重复读、幻读
  - WW

- 脏读：读取“未提交的脏数据”
  一个事务读取了另一个事务**尚未提交（未最终确认）的数据**，而后续该事务因异常回滚（Rollback），导致之前读取的数据是“无效的脏数据”。

步骤	事务B（转账方：给A转100元）	事务A（查询方：查自己余额）
1	开始事务，查询A当前余额为1000元	-
2	执行更新：将A的余额改为1100元（未提交）	-
3	-	开始事务，查询自己余额为1100元（读取了B未提交的数据）
4	发现转账信息错误，**回滚事务**（A的余额恢复为1000元）	-
5	-	基于“1100元”的脏数据做决策（如消费），但实际余额仅1000元

读取的是“临时无效数据”，若依赖该数据做业务决策，会导致逻辑错误。

- 不可重复读（Non-Repeatable Read）：同一行数据多次读取结果不同
  一个事务内，**多次读取同一行数据**，但由于其他事务对该行数据进行了“修改并提交”，导致前后两次读取的结果不一致。

步骤	事务A（查询方：多次查余额）	事务B（修改方：给A转500元）
1	开始事务，第一次查余额：1000元	-
2	-	开始事务，将A的余额改为1500元，**提交事务**
3	第二次查余额：1500元（与第一次结果不同）	-
4	事务A困惑：“为什么同一笔查询，结果变了？”	-

“不可重复”的是**同一行数据的内容**（因update/delete导致），影响事务内数据读取的“一致性”（例如统计报表中途数据被修改，结果不准）。

- 幻读：同一查询多次执行“行数不同”
  一个事务内，**多次执行相同的查询条件（如“查余额>1000的账户”）**，但由于其他事务“插入/删除了符合条件的行并提交”，导致前后两次查询返回的“行数不同”（像出现了“幻影”一样）。

步骤	事务A（查询方：查“余额>1000的账户数”）	事务B（插入方：新增一个余额1200的账户）
1	开始事务，第一次查询：符合条件的账户有2个	-
2	-	开始事务，新增一个余额1200的账户，**提交事务**
3	第二次查询：符合条件的账户有3个（行数增加，出现“幻影行”）	-
4	事务A困惑：“同一查询，怎么多了一行数据？”	-

对比维度	脏读（Dirty Read）	不可重复读（Non-Repeatable Read）	幻读（Phantom Read）
读取的数据状态	读取**未提交**的数据	读取**已提交**的数据	读取**已提交**的数据
数据变化类型	其他事务未提交的修改	其他事务**修改/删除**同一行并提交	其他事务**插入/删除**符合条件的行并提交
结果差异表现	数据是“无效脏数据”	同一行数据内容不一致	符合条件的**行数不一致**
问题本质	读了“临时数据”	读了“已变更的旧数据”	读了“新增/消失的行”

“幻读” 的是符合条件的行的数量（因 insert/delete 导致），而非行内数据内容（与不可重复读的关键区别）

- 如何解决？—— 数据库隔离级别
  数据库通过“隔离级别”控制并发事务的干扰程度，不同隔离级别能解决不同的一致性问题。主流数据库（如MySQL、PostgreSQL）支持4种标准隔离级别，其解决能力如下表：

隔离级别	解决脏读?	解决不可重复读?	解决幻读?	并发性能（从高到低）
读未提交（Read Uncommitted）	❌ 不解决	❌ 不解决	❌ 不解决	最高（几乎无隔离）
读已提交（Read Committed）	✅ 解决	❌ 不解决	❌ 不解决	较高

隔离级别	解决脏读?	解决不可重复读?	解决幻读?	并发性能（从高到低）
可重复读（Repeatable Read）	✅ 解决	✅ 解决	部分解决[1]	中等
串行化（Serializable）	✅ 解决	✅ 解决	✅ 解决	最低（完全串行执行）

> 注[1]：MySQL的InnoDB引擎通过“Next-Key Locking”（间隙锁+行锁）机制，在“可重复读”级别下就能完全解决幻读，这是InnoDB的特性（其他数据库如PostgreSQL的可重复读仍可能出现幻读）。

12. 索引

- 索引的优点：
  - 提高查询速度
  - 降低数据库IO成本
  - 提高数据的安全性
- 索引的缺点：
  - 降低更新速度
  - 占用存储空间
  - 降低维护速度
- 原理：
  - 索引是存储在磁盘上的数据结构，以便于查找
  - 索引的存储结构：B+树