Python开发之运维架构

讲师：王晓春

MYSQL数据库

本章内容

- 数据库介绍
- 关系型数据库
- MySQL介绍
- 安装MySQL
- 管理数据库
- 管理表
- 管理用户
- 权限管理
- 性能优化
- 备份和还原
- MYSQL集群

数据的时代

- 涉及的数据量大
- 数据不随程序的结束而消失
- 数据被多个应用程序共享
- 大数据

数据库的发展史

- 萌芽阶段：文件系统
  使用磁盘文件来存储数据
- 初级阶段：第一代数据库
  出现了网状模型、层次模型的数据库
- 中级阶段：第二代数据库
  关系型数据库和结构化查询语言
- 高级阶段：新一代数据库
  “关系-对象”型数据库

文件管理系统的缺点

- 编写应用程序不方便
- 数据冗余不可避免
- 应用程序依赖性
- 不支持对文件的并发访问
- 数据间联系弱
- 难以按用户视图表示数据
- 无安全控制功能

数据库管理系统的优点

- 相互关联的数据的集合
- 较少的数据冗余
- 程序与数据相互独立
- 保证数据的安全、可靠
- 最大限度地保证数据的正确性
- 数据可以并发使用并能同时保证一致性

数据库管理系统

- 数据库是数据的汇集，它以一定的组织形式存于存储介质上
- DBMS是管理数据库的系统软件，它实现数据库系统的各种功能。是数据库系统的核心
- DBA：负责数据库的规划、设计、协调、维护和管理等工作
- 应用程序指以数据库为基础的应用程序

数据库管理系统的基本功能

- 数据定义
- 数据处理
- 数据安全
- 数据备份

数据库系统的架构

- 单机架构
- 大型主机/终端架构
- 主从式架构（C/S）
- 分布式架构

关系型数据库

- 关系 ：关系就是二维表。并满足如下性质：
  表中的行、列次序并不重要
- 行row：表中的每一行，又称为一条记录
- 列column：表中的每一列，称为属性，字段
- 主键（Primary key）：用于惟一确定一个记录的字段
- 域domain：属性的取值范围，如，性别只能是‘男’和‘女’两个值

关系数据库

- RDBMS：
  - MySQL: MySQL, MariaDB, Percona Server
  - PostgreSQL: 简称为pgsql，EnterpriseDB
  - Oracle：
  - MSSQL：
  - DB2:
- 事务transaction：多个操作被当作一个整体对待
  - ACID:
    - A: 原子性
    - C：一致性
    - I: 隔离性
    - D：持久性

联系类型

- 联系的类型
  - 一对一联系(1:1)
  - 一对多联系(1:n)
  - 多对多联系(m:n)

数据三要素

- 数据结构：包括两类
  - 一类是与数据类型、内容、性质有关的对象，比如关系模型中的域、属性和关系等；
  - 另一类是与数据之间联系有关的对象，它从数据组织层表达数据记录与字段的结构
- 数据的操作：
  - 数据提取：在数据集合中提取感兴趣的内容。SELECT
  - 数据更新：变更数据库中的数据。INSERT、DELETE、UPDATE
- 数据的约束条件 ：是一组完整性规则的集合
  - 实体（行）完整性 Entity integrity
  - 域（列）完整性 Domain Integrity
  - 参考完整性 Referential Integrity

简易数据规划流程

- 第一阶段：收集数据，得到字段
  - 收集必要且完整的数据项
  - 转换成数据表的字段
- 第二阶段：把字段分类，归入表，建立表的关联
  - 关联：表和表间的关系
  - 分割数据表并建立关联的优点
  - 节省空间
  - 减少输入错误
  - 方便数据修改
- 第三阶段：
  - 规范化数据库

数据库的正规化分析

- RDMBS设计范式基础概念

设计关系数据库时，遵从不同的规范要求，设计出合理的关系型数据库，这些不同的规范要求被称为不同范式，各种范式呈递次规范，越高的范式数据库冗余越小

- 目前关系数据库有六种范式：第一范式（1NF）、第二范式（2NF）、第三范式（3NF）、巴德斯科范式（BCNF）、第四范式(4NF）和第五范式（5NF，又称完美范式）。满足最低要求的范式是第一范式（1NF）。在第一范式的基础上进一步满足更多规范要求的称为第二范式（2NF），其余范式以次类推。一般说来，数据库只需满足第三范式(3NF）即可

范式

- 1NF：无重复的列，每一列都是不可分割的基本数据项，同一列中不能有多个值，即实体中的某个属性不能有多个值或者不能有重复的属性。除去同类型的字段，就是无重复的列

说明：第一范式（1NF）是对关系模式的基本要求，不满足第一范式（1NF）的数据库就不是关系数据库

- 2NF：属性完全依赖于主键，第二范式必须先满足第一范式，要求表中的每个行必须可以被唯一地区分。通常为表加上一个列，以存储各个实例的唯一标识PK，非PK的字段需要与整个PK有直接相关性
- 3NF：属性不依赖于其它非主属性，满足第三范式必须先满足第二范式。第三范式要求一个数据库表中不包含已在其它表中已包含的非主关键字信息，非PK的字段间不能有从属关系

SQL概念

- SQL: Structure Query Language
  - 结构化查询语言
  - SQL解释器：
  - 数据存储协议：应用层协议，C/S
- S：server, 监听于套接字，接收并处理客户端的应用请求
- C：Client
  - 客户端程序接口
    - CLI
    - GUI
  - 应用编程接口
    - ODBC：Open Database Connectivity
    - JDBC：Java Data Base Connectivity

约束

- 约束：constraint，表中的数据要遵守的限制
  - 主键：一个或多个字段的组合，填入的数据必须能在本表中唯一标识本行；必须提供数据，即NOT NULL，一个表只能有一个
  - 惟一键：一个或多个字段的组合，填入的数据必须能在本表中唯一标识本行；允许为NULL，一个表可以存在多个
  - 外键：一个表中的某字段可填入的数据取决于另一个表的主键或唯一键已有的数据
  - 检查：字段值在一定范围内

基本概念

- 索引：将表中的一个或多个字段中的数据复制一份另存，并且此些需要按特定次序排序存储
- 关系运算：

  选择：挑选出符合条件的行

  投影：挑选出需要的字段

  连接：表间字段的关联

MySQL历史

- 1979年：TcX公司 Monty Widenius，Unireg
- 1996年：发布MySQL1.0，Solaris版本，Linux版本
- 1999年：MySQL AB公司，瑞典
- 2003年：MySQL 5.0版本，提供视图、存储过程等功能
- 2008年：Sun 收购
- 2009年：Oracle收购sun
- 2009年：Monty成立MariaDB

MySQL和MariaDB

- 官方网址：
  https://www.mysql.com/
  http://mariadb.org/
- 官方文档
  https://dev.mysql.com/doc/
  https://mariadb.com/kb/en/
- 版本演变：
  MySQL：5.1 --> 5.5 --> 5.6 --> 5.7
  MariaDB：5.5 -->10.0--> 10.1 --> 10.2 --> 10.3

MYSQL的特性

- 插件式存储引擎：也称为“表类型”，存储管理器有多种实现版本，功能和特性可能均略有差别；用户可根据需要灵活选择,Mysql5.5.5开始innoDB引擎是MYSQL默认引擎

  MyISAM ==> Aria

  InnoDB ==> XtraDB

- 单进程，多线程
- 诸多扩展和新特性
- 提供了较多测试组件
- 开源

安装MYSQL

- Mariadb安装方式：
- 1、源代码：编译安装
- 2、二进制格式的程序包：展开至特定路径，并经过简单配置后即可使用
- 3、程序包管理器管理的程序包
  - CentOS安装光盘
  - 项目官方：
    https://downloads.mariadb.org/mariadb/repositories/

RPM包安装MySQL

- RPM包安装
  - CentOS 7：安装光盘直接提供
    - mariadb-server　　服务器包
    - mariadb　　客户端工具包
  - CentOS 6
- 提高安全性
  - mysql_secure_installation
    - 设置数据库管理员root口令
    - 禁止root远程登录
    - 删除anonymous用户帐号
    - 删除test数据库

MariaDB程序

- 客户端程序：
  - mysql: 交互式的CLI工具
  - mysqldump：备份工具，基于mysql协议向mysqld发起查询请求，并将查得的所有数据转换成insert等写操作语句保存文本文件中
  - mysqladmin：基于mysql协议管理mysqld
  - mysqlimport：数据导入工具
- MyISAM存储引擎的管理工具：
  - myisamchk：检查MyISAM库
  - myisampack：打包MyISAM表，只读
- 服务器端程序
  - mysqld_safe
  - mysqld
  - mysqld_multi：多实例 ，示例：mysqld_multi --example

用户账号

- mysql用户账号由两部分组成：
  'USERNAME'@'HOST ‘
- 说明：
  HOST限制此用户可通过哪些远程主机连接mysql服务器
  支持使用通配符：
  % 匹配任意长度的任意字符
  172.16.0.0/255.255.0.0 或 172.16.%.%
  _ 匹配任意单个字符

Mysql 客户端

- mysql使用模式：
- 交互式模式：
  - 可运行命令有两类：
    - 客户端命令：
      - \h, help
      - \u，use
      - \s，status
      - \!，system
    - 服务器端命令：
      - SQL, 需要语句结束符；
- 脚本模式：

```
mysql –uUSERNAME -pPASSWORD < /path/somefile.sql
mysql> source /path/from/somefile.sql
```

Mysql客户端

- mysql客户端可用选项：

```
-A, --no-auto-rehash 禁止补全
-u, --user=          用户名,默认为root
-h, --host=          服务器主机,默认为localhost
-p, --passowrd=      用户密码,建议使用-p,默认为空密码
-P, --port=          服务器端口
-S, --socket=        指定连接socket文件路径
-D, --database=      指定默认数据库
-C, --compress       启用压缩
-e    “SQL“          执行SQL命令
-V, --version        显示版本
-v  --verbose        显示详细信息
--print-defaults     获取程序默认使用的配置
```

socket地址

- 服务器监听的两种socket地址：

  ip socket: 监听在tcp的3306端口，支持远程通信

  unix sock: 监听在sock文件上，仅支持本机通信

  如：/var/lib/mysql/mysql.sock)

  说明：host为localhost,127.0.0.1时自动使用unix sock

执行命令

- 运行mysql命令：默认空密码登录

  mysql>use mysql

  mysql>select user();查看当前用户

  mysql>SELECT User,Host,Password FROM user;

- 登录系统：mysql –uroot –p
- 客户端命令：本地执行

  mysql> help

  每个命令都完整形式和简写格式

  mysql> status 或 \s

- 服务端命令：通过mysql协议发往服务器执行并取回结果

  每个命令都必须命令结束符号；默认为分号

  SELECT VERSION();

服务器端配置

- 服务器端(mysqld)：工作特性有多种配置方式
- 1、命令行选项：
- 2、配置文件：类ini格式

集中式的配置，能够为mysql的各应用程序提供配置信息

[mysqld]
[mysqld_safe]
[mysqld_multi]
[mysql]
[mysqldump]
[server]
[client]

格式：parameter = value

说明：_和- 相同

1，ON，TRUE意义相同， 0，OFF，FALSE意义相同

配置文件

◆配置文件：

后面覆盖前面的配置文件，顺序如下：

- /etc/my.cnf　　Global选项
- /etc/mysql/my.cnf　　Global选项
- *SYSCONFDIR/my.cnf*　　Global选项
- $MYSQL_HOME/my.cnf　　Server-specific 选项
- --defaults-extra-file=*path*
- ~/.my.cnf　　User-specific 选项

MairaDB配置

- 侦听3306/tcp端口可以在绑定有一个或全部接口IP上
- vim /etc/my.cnf

  [mysqld]

  skip-networking=1 关闭网络连接，只侦听本地客户端， 所有和服务器的交互都通过一个socket实现，socket的配置存放在/var/lib/mysql/mysql.sock ） 可在/etc/my.cnf修改

通用二进制格式安装过程

- 二进制格式安装过程
- (1) 准备用户
  - groupadd -r -g 306 mysql
  - useradd -r -g 306 -u 306 –m –d /app/data mysql
- (2) 准备数据目录

  以/app/data为例,建议使用逻辑卷
  - chown mysql:mysql /app/data
- (3) 准备二进制程序
  - tar xf mariadb-VERSION-linux-x86_64.tar.gz -C /usr/local
  - cd /usr/local;ln -sv mariadb-VERSION mysql
  - chown -R root:mysql /usr/local/mysql/

通用二进制格式安装过程

- (4) 准备配置文件

  mkdir /etc/mysql/

  cp support-files/my-huge.cnf /etc/mysql/my.cnf

  [mysqld]中添加三个选项：

  datadir = /app/data

  innodb_file_per_table = on

  skip_name_resolve = on　禁止主机名解析，建议使用

通用二进制格式安装过程

- (5)创建数据库文件

```
cd /usr/local/mysql/
./scripts/mysql_install_db --datadir=/app/data --user=mysql
```

- (6)准备日志文件

```
touch  /var/log/mysqld.log
chown mysqld /var/log/mysqld.log
```

- (7)准备服务脚本，并启动服务

```
cp ./support-files/mysql.server /etc/rc.d/init.d/mysqld
chkconfig --add mysqld
service mysqld start
```

- (8)安全初始化

```
/user/local/mysql/bin/mysql_secure_installation
```

源码编译安装mariadb

- 安装包

  yum install bison bison-devel zlib-devel libcurl-devel libarchive-devel boost-devel gcc gcc-c++ cmake ncurses-devel gnutls-devel libxml2-devel openssl-devel libevent-devel libaio-devel

  提示：如果出错，执行rm -f CMakeCache.txt

- 做准备用户和数据目录

  mkdir /data

  useradd –r –s /bin/false –m –d /data/mysqldb/ mysql

  tar xvf mariadb-10.2.15.tar.gz

- cmake 编译安装

  cd mariadb-10.2.15/

  编译选项:

  https://dev.mysql.com/doc/refman/5.7/en/source-configuration-options.html

源码编译安装mariadb


```
cmake . \
-DCMAKE_INSTALL_PREFIX=/app/mysql \
-DMYSQL_DATADIR=/data/mysqldb/ \
-DSYSCONFDIR=/etc \
-DMYSQL_USER=mysql \
-DWITH_INNOBASE_STORAGE_ENGINE=1 \
-DWITH_ARCHIVE_STORAGE_ENGINE=1 \
-DWITH_BLACKHOLE_STORAGE_ENGINE=1 \
-DWITH_PARTITION_STORAGE_ENGINE=1  \
-DWITHOUT_MROONGA_STORAGE_ENGINE=1 \
-DWITH_DEBUG=0 \
-DWITH_READLINE=1 \
-DWITH_SSL=system \
-DWITH_ZLIB=system \
-DWITH_LIBWRAP=0 \
-DENABLED_LOCAL_INFILE=1  \
-DMYSQL_UNIX_ADDR=/app/mysql/mysql.sock \
-DDEFAULT_CHARSET=utf8 \
-DDEFAULT_COLLATION=utf8_general_ci
make && make install
```

源码编译安装mariadb

- 准备环境变量

```
echo 'PATH=/app/mysql/bin:$PATH' > /etc/profile.d/mysql.sh
.    /etc/profile.d/mysql.sh
```

- 生成数据库文件

```
cd /app/mysql/
scripts/mysql_install_db --datadir=/data/mysqldb/ --user=mysql
```

- 准备配置文件

```
cp  /app/mysql/support-files/my-huge.cnf  /etc/my.cnf
```

- 准备启动脚本

```
cp /app/mysql/support-files/mysql.server  /etc/init.d/mysqld
```

- 启动服务

```
chkconfig --add mysqld ;service mysqld start
```

关系型数据库的常见组件

- 数据库：database
- 表：table
  - 行：row
  - 列：column
- 索引：index
- 视图：view
- 用户：user
- 权限：privilege
- 存储过程：procedure，无返回值
- 存储函数：function，有返回值
- 触发器：trigger
- 事件调度器：event scheduler，任务计划

SQL语言的兴起与语法标准

- 20世纪70年代，IBM开发出SQL，用于DB2
- 1981年，IBM推出SQL/DS数据库
- 业内标准微软和Sybase的T-SQL，Oracle的PL/SQL
- SQL作为关系型数据库所使用的标准语言，最初是基于IBM的实现在1986年被批准的。1987年，“国际标准化组织(ISO)”把ANSI(美国国家标准化组织)SQL作为国际标准。
- SQL：ANSI SQL

  SQL-86, SQL-89, SQL-92, SQL-99, SQL-03

SQL语言规范

- 在数据库系统中，SQL语句不区分大小写(建议用大写)
- 但字符串常量区分大小写
- SQL语句可单行或多行书写，以“;”结尾
- 关键词不能跨多行或简写
- 用空格和缩进来提高语句的可读性
- 子句通常位于独立行，便于编辑，提高可读性
- 注释：
  - SQL标准：
    /*注释内容*/　多行注释
    -- 注释内容　　单行注释，注意有空格
  - MySQL注释：
    #

数据库对象

- 数据库的组件(对象)：
  数据库、表、索引、视图、用户、存储过程、函数、触发器、事件调度器等
- 命名规则：
  - 必须以字母开头
  - 可包括数字和三个特殊字符（# _ $）
  - 不要使用MySQL的保留字
  - 同一database(Schema)下的对象不能同名

SQL语句分类

- SQL语句分类：
  - DDL: Data Defination Language
    CREATE, DROP, ALTER
  - DML: Data Manipulation Language
    INSERT, DELETE, UPDATE
  - DCL：Data Control Language
    GRANT, REVOKE
  - DQL：Data Query Language
    SELECT

SQL语句构成

- SQL语句构成：

  Keyword组成clause

  多条clause组成语句

- 示例：

  SELECT *　　SELECT子句

  FROM products　　FROM子句

  WHERE price>400　　WHERE子句

  说明：一组SQL语句，由三个子句构成，SELECT,FROM和WHERE是关键字

数据库操作

- 创建数据库：

```
CREATE DATABASE|SCHEMA [IF NOT EXISTS] 'DB_NAME';
CHARACTER SET 'character set name'
COLLATE 'collate name'
```

- 删除数据库

```
DROP DATABASE|SCHEMA [IF EXISTS] 'DB_NAME';
```

- 查看支持所有字符集：SHOW CHARACTER SET;
- 查看支持所有排序规则：SHOW COLLATION;
- 获取命令使用帮助：

```
mysql> HELP KEYWORD;
```

- 查看数据库列表：

```
mysql> SHOW DATABASES;
```

表

- 表：二维关系
- 设计表：遵循规范
- 定义：字段，索引

  字段：字段名，字段数据类型，修饰符

  约束，索引：应该创建在经常用作查询条件的字段上

创建表

- 创建表：CREATE TABLE
- (1) 直接创建
- (2) 通过查询现存表创建；新表会被直接插入查询而来的数据

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name [(create_definition,...)] [table_options] [partition_options] select_statement

- (3) 通过复制现存的表的表结构创建，但不复制数据

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name { LIKE old_tbl_name | (LIKE old_tbl_name) }

- 注意：
  - Storage Engine是指表类型，也即在表创建时指明其使用的存储引擎，同一库中不同表可以使用不同的存储引擎
  - 同一个库中表建议要使用同一种存储引擎类型

创建表

- CREATE TABLE [IF NOT EXISTS] ‘tbl_name’ (col1 type1 修饰符, col2 type2 修饰符, ...)
- 字段信息
  - col type1
  - PRIMARY KEY(col1,...)
  - INDEX(col1, ...)
  - UNIQUE KEY(col1, ...)
- 表选项：
  - ENGINE [=] engine_name
    SHOW ENGINES;查看支持的engine类型
  - ROW_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}
- 获取帮助：mysql> HELP CREATE TABLE;

表操作

- 查看所有的引擎：SHOW ENGINES
- 查看表：SHOW TABLES [FROM db_name]
- 查看表结构：DESC [db_name.]tb_name
- 删除表：DROP TABLE [IF EXISTS] tb_name
- 查看表创建命令：SHOW CREATE TABLE tbl_name
- 查看表状态：SHOW TABLE STATUS LIKE 'tbl_name’
- 查看库中所有表状态：SHOW TABLE STATUS FROM db_name

数据类型

- 数据类型：
  - 数据长什么样？
  - 数据需要多少空间来存放？
- 系统内置数据类型和用户定义数据类型
- MySql支持多种列类型：
  - 数值类型
  - 日期/时间类型
  - 字符串(字符)类型
  - https://dev.mysql.com/doc/refman/5.5/en/data-types.html
- 选择正确的数据类型对于获得高性能至关重要，三大原则：
  - 更小的通常更好，尽量使用可正确存储数据的最小数据类型
  - 简单就好，简单数据类型的操作通常需要更少的CPU周期
  - 尽量避免NULL，包含为NULL的列，对MySQL更难优化

数据类型

- 1、整型
  - tinyint(m)　　1个字节 范围(-128~127)
  - smallint(m)　　2个字节 范围(-32768~32767)
  - mediumint(m)　　3个字节 范围(-8388608~8388607)
  - int(m)　　4个字节 范围(-2147483648~2147483647)
  - bigint(m)　　8个字节 范围(+-9.22*10的18次方)

取值范围如果加了unsigned，则最大值翻倍，如tinyint unsigned的取值范围为(0~255)

int(m)里的m是表示SELECT查询结果集中的显示宽度，并不影响实际的取值范围，规定了MySQL的一些交互工具（例如MySQL命令行客户端）用来显示字符的个数。对于存储和计算来说，Int(1)和Int(20)是相同的

- BOOL，BOOLEAN：布尔型，是TINYINT(1)的同义词。zero值被视为假。非zero值视为真

数据类型

- 2、浮点型(float和double)，近似值
  - float(m,d) 单精度浮点型 8位精度(4字节) m总个数，d小数位
  - double(m,d) 双精度浮点型16位精度(8字节) m总个数，d小数位
  - 设一个字段定义为float(6,3)，如果插入一个数123.45678,实际数据库里存的是123.457，但总个数还以实际为准，即6位

数据类型

- 3、定点数
  - 在数据库中存放的是精确值,存为十进制
  - decimal(m,d) 参数m<65 是总个数，d<30且 d<m 是小数位
  - MySQL5.0和更高版本将数字打包保存到一个二进制字符串中（每4个字节存9个数字）。例如，decimal(18,9)小数点两边将各存储9个数字，一共使用9个字节：小数点前的数字用4个字节，小数点后的数字用4个字节，小数点本身占1个字节
  - 浮点类型在存储同样范围的值时，通常比decimal使用更少的空间。float使用4个字节存储。double占用8个字节
  - 因为需要额外的空间和计算开销，所以应该尽量只在对小数进行精确计算时才使用decimal——例如存储财务数据。但在数据量比较大的时候，可以考虑使用bigint代替decimal

数据类型

- 4、字符串(char,varchar,_text)
  - char(n)　　固定长度，最多255个字符
  - varchar(n)　　可变长度，最多65535个字符
  - tinytext　　可变长度，最多255个字符
  - text　　可变长度，最多65535个字符
  - mediumtext　可变长度，最多2的24次方-1个字符
  - longtext　　可变长度，最多2的32次方-1个字符
  - BINARY(M)　固定长度，可存二进制或字符，长度为0-M字节
  - VARBINARY(M) 可变长度，可存二进制或字符，允许长度为0-M字节
  - 内建类型：ENUM枚举, SET集合

数据类型

◆ char和varchar：

- 1.char(n) 若存入字符数小于n，则以空格补于其后，查询之时再将空格去掉。所以char类型存储的字符串末尾不能有空格，varchar不限于此。
- 2.char(n) 固定长度，char(4)不管是存入几个字符，都将占用4个字节，varchar是存入的实际字符数+1个字节（n< n>255)，所以varchar(4),存入3个字符将占用4个字节。
- 3.char类型的字符串检索速度要比varchar类型的快

◆ varchar和text：

- 1.varchar可指定n，text不能指定，内部存储varchar是存入的实际字符数+1个字节（n< n>255)，text是实际字符数+2个字节。
- 2.text类型不能有默认值
- 3.varchar可直接创建索引，text创建索引要指定前多少个字符。varchar查询速度快于text

数据类型

- 5.二进制数据：BLOB
  - BLOB和text存储方式不同，TEXT以文本方式存储，英文存储区分大小写，而Blob是以二进制方式存储，不分大小写
  - BLOB存储的数据只能整体读出
  - TEXT可以指定字符集，BLOB不用指定字符集
- 6.日期时间类型
  - date　　日期 '2008-12-2'
  - time　　时间 '12:25:36'
  - datetime　　日期时间 '2008-12-2 22:06:44'
  - timestamp　　自动存储记录修改时间
  - YEAR(2), YEAR(4)：年份

timestamp字段里的时间数据会随其他字段修改的时候自动刷新，这个数据类型的字段可以存放这条记录最后被修改的时间

修饰符

◆所有类型：

- NULL　　数据列可包含NULL值
- NOT NULL　　数据列不允许包含NULL值
- DEFAULT　　默认值
- PRIMARY KEY　　主键
- UNIQUE KEY　　唯一键
- CHARACTER SET name　　指定一个字符集

◆数值型

- AUTO_INCREMENT　　自动递增，适用于整数类型
- UNSIGNED　　无符号

示例

- CREATE TABLE students (id int UNSIGNED NOT NULL PRIMARY KEY,name VARCHAR (20) NOT NULL,age tinyint UNSIGNED);
- DESC students;
- CREATE TABLE students2 (id int UNSIGNED NOT NULL ,name VARCHAR(20) NOT NULL,age tinyint UNSIGNED,PRIMARY KEY(id,name));

表操作

- DROP TABLE [IF EXISTS] 'tbl_name';
- ALTER TABLE 'tbl_name'
  - 字段：
    - 添加字段：add
      ADD col1 data_type [FIRST|AFTER col_name]
    - 删除字段：drop
    - 修改字段：
      alter（默认值）, change（字段名）, modify（字段属性）
  - 索引:
    - 添加索引：add index
    - 删除索引: drop index
  - 表选项
    - 修改:
- 查看表上的索引：SHOW INDEXES FROM [db_name.]tbl_name;
- 查看帮助：Help ALTER TABLE

修改表示例

- ALTER TABLE students RENAME s1;
- ALTER TABLE s1 ADD phone varchar(11) AFTER name;
- ALTER TABLE s1 MODIFY phone int;
- ALTER TABLE s1 CHANGE COLUMN phone mobile char(11);
- ALTER TABLE s1 DROP COLUMN mobile;
- Help ALTER TABLE 查看帮助

修改表示例

- ALTER TABLE students ADD gender ENUM('m','f')
- ALETR TABLE students CHANGE id sid int UNSIGNED NOT NULL PRIMARY KEY;
- ALTER TABLE students ADD UNIQUE KEY(name);
- ALTER TABLE students ADD INDEX(age);
- DESC students;
- SHOW INDEXES FROM students;
- ALTER TABLE students DROP age;

DML语句

- DML: INSERT, DELETE, UPDATE
- INSERT：

  一次插入一行或多行数据

  语法

  - INSERT [L OW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
    [ON DUPLICATE KEY UPDATE 如果重复更新之
    col_name=expr
    [, col_name=expr] ...]

  简化写法：

  INSERT tbl_name [(col1,...)] VALUES (val1,...), (val21,...)

DML语句

- INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
  [INTO] tbl_name
  SET col_name={expr | DEFAULT}, ...
  [ON DUPLICATE KEY UPDATE
    col_name=expr
      [, col_name=expr] ...]
- INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
  [INTO] tbl_name [(col_name,...)]
  SELECT ...
  [ON DUPLICATE KEY UPDATE
    col_name=expr
      [, col_name=expr] ...]

DML语句

- UPDATE：
- UPDATE [LOW_PRIORITY] [IGNORE] table_reference
  SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
  [WHERE where_condition]
  [ORDER BY ...]
  [LIMIT row_count]
- 注意：一定要有限制条件，否则将修改所有行的指定字段
  限制条件：
  WHERE
  LIMIT
- Mysql 选项：-U|--safe-updates| --i-am-a-dummy

DML语句

- DELETE:
- DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
  [WHERE where_condition]
  [ORDER BY ...]
  [LIMIT row_count]
    可先排序再指定删除的行数
- 注意：一定要有限制条件，否则将清空表中的所有数据
    限制条件：
      WHERE
      LIMIT
- TRUNCATE TABLE tbl_name; 清空表

DQL语句

- SELECT

```
[ALL | DISTINCT | DISTINCTROW ]
 [SQL_CACHE | SQL_NO_CACHE]
 select_expr [, select_expr ...]
    [FROM table_references
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [FOR UPDATE | LOCK IN SHARE MODE]
```

SELECT

- 字段显示可以使用别名：
  col1 AS alias1, col2 AS alias2, ...
- WHERE子句：指明过滤条件以实现“选择”的功能：
  过滤条件：布尔型表达式
  算术操作符：+, -, *, /, %
  比较操作符：=, !=, <>, >, >=, <, <=
  BETWEEN min_num AND max_num
  IN (element1, element2, ...)
  IS NULL
  IS NOT NULL

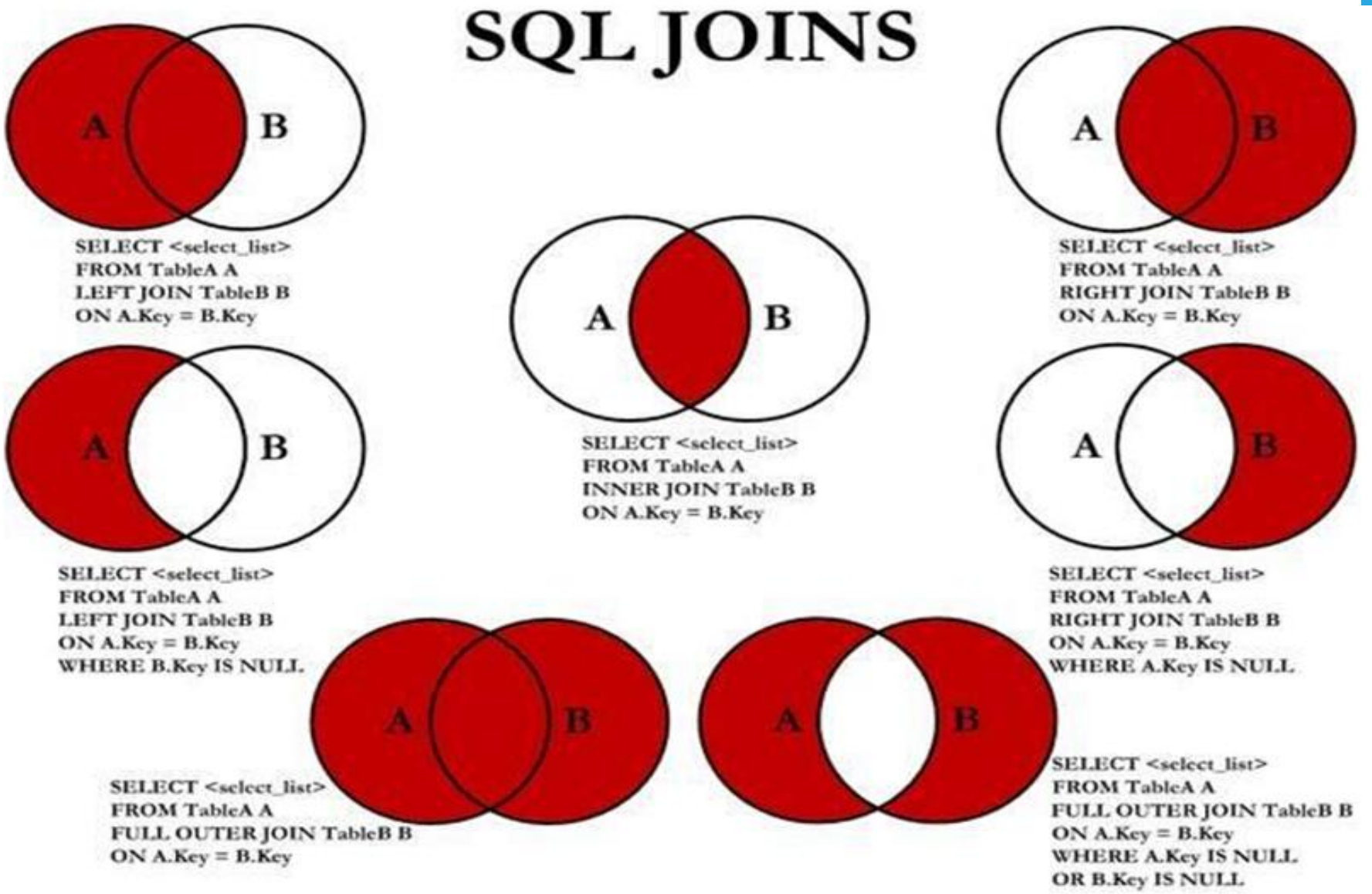
SELECT

- LIKE:
  - %: 任意长度的任意字符
  - _：任意单个字符
- RLIKE：正则表达式，索引失效，不建议使用
- REGEXP：匹配字符串可用正则表达式书写模式，同上
- 逻辑操作符：
  - NOT
  - AND
  - OR
  - XOR

SELECT

- GROUP：根据指定的条件把查询结果进行“分组”以用于做“聚合”运算
  avg(), max(), min(), count(), sum()
  HAVING: 对分组聚合运算后的结果指定过滤条件
- ORDER BY: 根据指定的字段对查询结果进行排序
  升序：ASC
  降序：DESC
- LIMIT [[offset,]row_count]：对查询的结果进行输出行数数量限制
- 对查询结果中的数据请求施加“锁”
  FOR UPDATE: 写锁，独占或排它锁，只有一个读和写
  LOCK IN SHARE MODE: 读锁，共享锁，同时多个读

SQL JOINS



多表查询

- 交叉连接：笛卡尔乘积
- 内连接：
  - 等值连接：让表之间的字段以“等值”建立连接关系；
  - 不等值连接
  - 自然连接:去掉重复列的等值连接
  - 自连接
- 外连接：
  - 左外连接：
    - FROM tb1 LEFT JOIN tb2 ON tb1.col=tb2.col
  - 右外连接
    - FROM tb1 RIGHT JOIN tb2 ON tb1.col=tb2.col

多表查询

- 子查询：在查询语句嵌套着查询语句，性能较差

  基于某语句的查询结果再次进行的查询

- 用在WHERE子句中的子查询：
  - 用于比较表达式中的子查询；子查询仅能返回单个值

    SELECT Name,Age FROM students WHERE Age>(SELECT avg(Age) FROM students);
  - 用于IN中的子查询：子查询应该单键查询并返回一个或多个值从构成列表

    SELECT Name,Age FROM students WHERE Age IN (SELECT Age FROM teachers);
  - 用于EXISTS

多表查询

- 用于FROM子句中的子查询

使用格式：SELECT tb_alias.col1,... FROM (SELECT clause) AS tb_alias WHERE Clause;

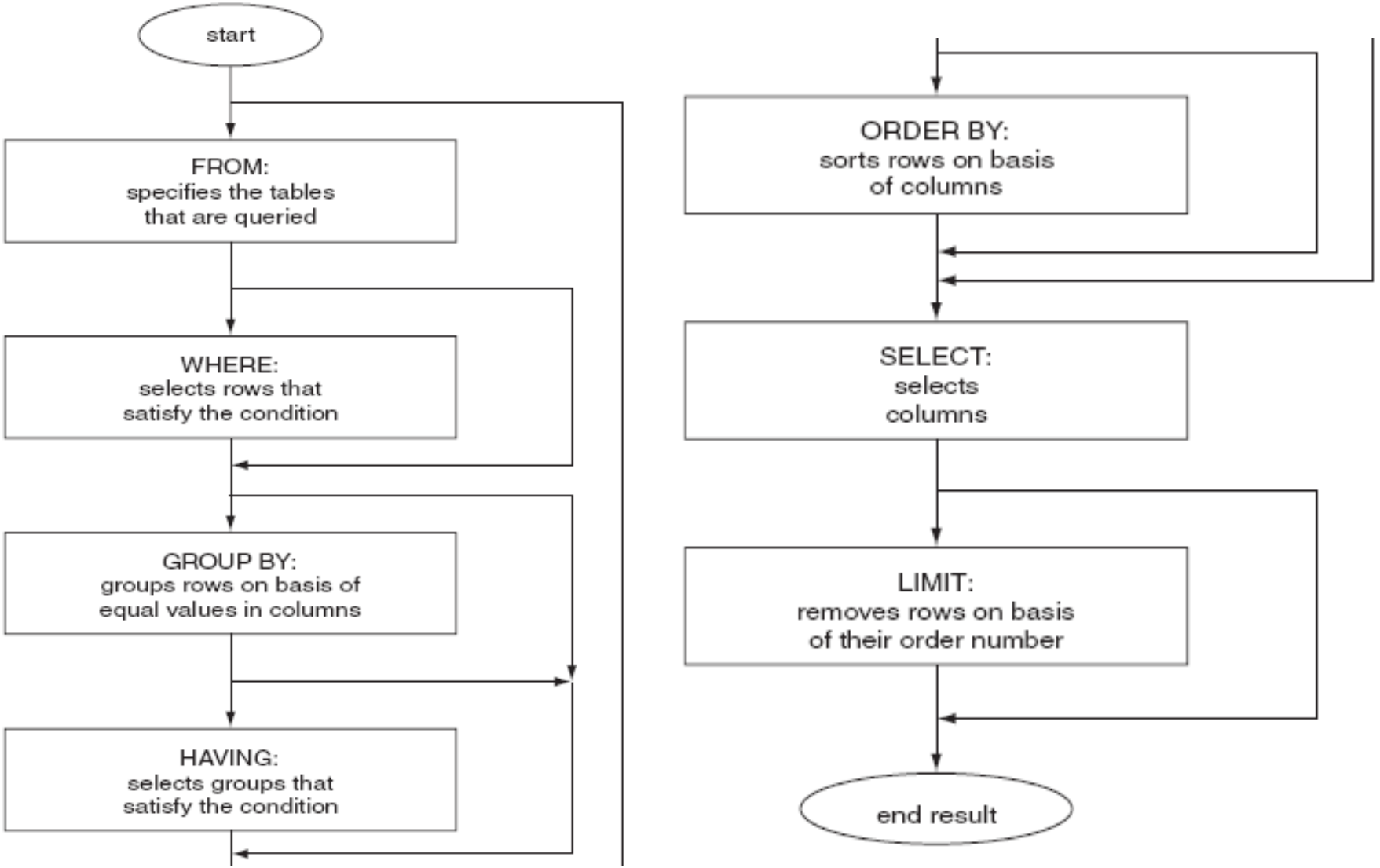
示例：

SELECT s.aage,s.ClassID FROM (SELECT avg(Age) AS aage,ClassID FROM students WHERE ClassID IS NOT NULL GROUP BY ClassID) AS s WHERE s.aage>30;

- 联合查询：UNION

SELECT Name,Age FROM students UNION SELECT Name,Age FROM teachers;

SELECT语句


start
FROM:
specifies the tables
that are queried
WHERE:
selects rows that
satisfy the condition
GROUP BY:
groups rows on basis of
equal values in columns
HAVING:
selects groups that
satisfy the condition
ORDER BY:
sorts rows on basis
of columns
SELECT:
selects
columns
LIMIT:
removes rows on basis
of their order number
end result

视图

- 视图：VIEW,虚表，保存有实表的查询结果
- 创建方法：

```
CREATE   VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

- 查看视图定义：SHOW CREATE VIEW view_name
- 删除视图：

```
DROP VIEW [IF EXISTS]
    view_name [, view_name] ...
    [RESTRICT | CASCADE]
```

- 视图中的数据事实上存储于“基表”中，因此，其修改操作也会针对基表实现；其修改操作受基表限制

函数

- 函数：系统函数和自定义函数

  系统函数:https://dev.mysql.com/doc/refman/5.7/en/func-op-summary-ref.html

- 自定义函数 (user-defined function UDF)
  - 保存在mysql.proc表中
  - 创建UDF:

    CREATE [AGGREGATE] FUNCTION function_name(parameter_name type,[parameter_name type,...])
    RETURNS {STRING|INTEGER|REAL}
     runtime_body
  - 说明：

    参数可以有多个,也可以没有参数

    必须有且只有一个返回值

自定义函数

- 查看函数列表：

  SHOW FUNCTIOIN STATUS;

- 查看函数定义

  SHOW CREATE FUNCTION function_name

- 删除UDF:

  DROP FUNCTION function_name

- 调用自定义函数语法:

  SELECT function_name(parameter_value,...)

自定义函数

- 示例:无参UDF

```
CREATE FUNCTION simpleFun() RETURNS VARCHAR(20) RETURN "Hello World! “;
```

- 示例：有参数UDF

```
DELIMITER //
CREATE FUNCTION deleteById(uid SMALLINT UNSIGNED) RETURNS VARCHAR(20)
BEGIN
DELETE FROM students WHERE stuid = uid;
RETURN (SELECT COUNT(uid) FROM students);
END//
DELIMITER ;
```

自定义函数

- 自定义函数中定义局部变量语法:

  DECLARE 变量1[,变量2,...]变量类型 [DEFAULT 默认值]

- 说明：局部变量的作用范围是在BEGIN...END程序中,而且定义局部变量语句必须在BEGIN...END的第一行定义

- 示例:

```
DELIMITER //
CREATE FUNCTION addTwoNumber(x SMALLINT UNSIGNED, Y SMALLINT
   UNSIGNED)
RETURNS SMALLINT
BEGIN
DECLARE a, b SMALLINT UNSIGNED DEFAULT 10;
SET  a = x, b = y;
RETURN a+b;
END//
DELIMITER ;
```

自定义函数

- 为变量赋值语法
  - SET parameter_name = value[,parameter_name = value...]
  - SELECT INTO parameter_name

- 示例:

```
...
DECLARE x int;
SELECT COUNT(id) FROM tdb_name INTO x;
RETURN x;
END//
```

存储过程

- 存储过程：存储过程保存在mysql.proc表中
- 创建存储过程

  CREATE PROCEDURE sp_name ([proc_parameter [,proc_parameter ...]])
  routime_body

  其中:proc_parameter : [IN|OUT|INOUT] parameter_name type

  其中IN表示输入参数，OUT表示输出参数，INOUT表示既可以输入也可以输出；
  param_name表示参数名称；type表示参数的类型
- 查看存储过程列表

  SHOW PROCEDURE STATUS

存储过程

- 查看存储过程定义

  SHOW CREATE PROCEDURE sp_name

- 调用存储过程:

  CALL sp_name ([proc_parameter [,proc_parameter ...]])

  CALL sp_name

  说明:当无参时,可以省略"()",当有参数时,不可省略"()"

- 存储过程修改:

  ALTER语句修改存储过程只能修改存储过程的注释等无关紧要的东西,不能修改存储过程体,所以要修改存储过程,方法就是删除重建

- 删除存储过程:

  DROP PROCEDURE [IF EXISTS] sp_name

存储过程示例

- 创建无参存储过程:

```
delimiter //

CREATE PROCEDURE showTime()
BEGIN
   SELECT now();
END//

delimiter ;

CALL showTime;
```

存储过程示例

- 创建含参存储过程:只有一个IN参数

```
delimiter //
CREATE PROCEDURE seleById(IN id SMALLINT UNSIGNED)
BEGIN
   SELECT * FROM students WHERE stuid = uid;
END//
delimiter ;

call seleById(2);
```

存储过程示例

- 示例

```
delimiter //
CREATE PROCEDURE dorepeat(p1 INT)
BEGIN
  SET @x = 0;
   REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
END//

delimiter ;
CALL dorepeat(1000);
SELECT @x;
```

存储过程示例

- 创建含参存储过程:包含IN参数和OUT参数

```
delimiter //
CREATE PROCEDURE deleteById(IN id SMALLINT UNSIGNED, OUT num
    SMALLINT UNSIGNED)
BEGIN
DELETE FROM students WHERE stuid = id;
SELETE row_count() into num;
END//
delimiter ;
call seleById(2,@Line);
SELETE @Line;
```

- 说明:创建存储过程deleteById,包含一个IN参数和一个OUT参数.调用时,传入删除的ID和保存被修改的行数值的用户变量@Line,select @Line;输出被影响行数

存储过程

- 存储过程优势:
  - 存储过程把经常使用的SQL语句或业务逻辑封装起来,预编译保存在数据库中,当需要时从数据库中直接调用,省去了编译的过程
  - 提高了运行速度
  - 同时降低网络数据传输量
- 存储过程与自定义函数的区别:
  - 存储过程实现的过程要复杂一些,而函数的针对性较强
  - 存储过程可以有多个返回值,而自定义函数只有一个返回值
  - 存储过程一般独立的来执行,而函数往往是作为其他SQL语句的一部分来使用

流程控制

- 存储过程和函数中可以使用流程控制来控制语句的执行
- 流程控制：
  - IF：用来进行条件判断。根据是否满足条件，执行不同语句
  - CASE：用来进行条件判断，可实现比IF语句更复杂的条件判断
  - LOOP：重复执行特定的语句，实现一个简单的循环
  - LEAVE：用于跳出循环控制
  - ITERATE：跳出本次循环，然后直接进入下一次循环
  - REPEAT：有条件控制的循环语句。当满足特定条件时，就会跳出循环语句
  - WHILE：有条件控制的循环语句

触发器

- 触发器的执行不是由程序调用，也不是由手工启动，而是由事件来触发、激活从而实现执行
- 创建触发器

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    TRIGGER trigger_name
    trigger_time trigger_event
    ON tbl_name FOR EACH ROW
    trigger_body
```

  - 说明：

trigger_name：触发器的名称
trigger_time：{ BEFORE | AFTER }，表示在事件之前或之后触发
trigger_event:：{ INSERT |UPDATE | DELETE }，触发的具体事件
tbl_name：该触发器作用在表名

触发器示例


```
CREATE TABLE student_info (
  stu_no INT(11) NOT NULL AUTO_INCREMENT,
  stu_name VARCHAR(255) DEFAULT NULL,
  PRIMARY KEY (stu_no)
);
CREATE TABLE student_count (
  student_count INT(11) DEFAULT 0
);
INSERT INTO student_count VALUES(0);
```

触发器示例

- 示例：创建触发器，在向学生表INSERT数据时，学生数增加，DELETE学生时，学生数减少

```
CREATE TRIGGER trigger_student_count_insert
AFTER INSERT
ON student_info FOR EACH ROW
UPDATE student_count SET student_count=student_count+1;

CREATE TRIGGER trigger_student_count_delete
AFTER DELETE
ON student_info FOR EACH ROW
UPDATE student_count SET student_count=student_count-1;
```

触发器

- 查看触发器

  SHOW TRIGGERS

  查询系统表information_schema.triggers的方式指定查询条件，查看指定的触发器信息。

  ```
  mysql> USE information_schema;
  Database changed
  mysql> SELECT  * FROM triggers WHERE
          trigger_name='trigger_student_count_insert';
  ```

- 删除触发器

  DROP TRIGGER trigger_name;

MySQL用户和权限管理

- 元数据数据库：mysql
  - 系统授权表：
  - db, host, user
  - columns_priv, tables_priv, procs_priv, proxies_priv
- 用户账号：
  'USERNAME'@'HOST'：
  - @'HOST':
    - 主机名；
    - IP地址或Network;
    - 通配符：
      - %, _: 172.16.%.%

用户管理

- 创建用户：CREATE USER

  CREATE USER 'USERNAME'@'HOST' [IDENTIFIED BY 'password'] ;

  默认权限：USAGE

- 用户重命名：RENAME USER

  RENAME USER old_user_name TO new_user_name

- 删除用户：

  DROP USER 'USERNAME'@'HOST ‘

示例：删除默认的空用户

DROP USER ''@'localhost';

用户管理

- 修改密码：
  - mysql>SET PASSWORD FOR 'user'@'host' = PASSWORD(‘password');
  - mysql>UPDATE mysql.user SET password=PASSWORD('password') WHERE clause;

    此方法需要执行下面指令才能生效：

    mysql> FLUSH PRIVILEGES;
  - #mysqladmin -u root –poldpass password ‘newpass ‘
- 忘记管理员密码的解决办法：
  - 启动mysqld进程时，为其使用如下选项：

    --skip-grant-tables --skip-networking
  - 使用UPDATE命令修改管理员密码
  - 关闭mysqld进程，移除上述两个选项，重启mysqld

MySQL权限管理

- 权限类别：
  - 管理类
  - 程序类
  - 数据库级别
  - 表级别
  - 字段级别

MySQL用户和权限管理

- 管理类：
  - CREATE TEMPORARY TABLES
  - CREATE USER
  - FILE
  - SUPER
  - SHOW DATABASES
  - RELOAD
  - SHUTDOWN
  - REPLICATION SLAVE
  - REPLICATION CLIENT
  - LOCK TABLES
  - PROCESS

MySQL用户和权限管理

- 程序类： FUNCTION、PROCEDURE、TRIGGER
  - CREATE
  - ALTER
  - DROP
  - EXCUTE
- 库和表级别：DATABASE、TABLE
  - ALTER
  - CREATE
  - CREATE VIEW
  - DROP
  - INDEX
  - SHOW VIEW
  - GRANT OPTION：能将自己获得的权限转赠给其他用户

MySQL用户和权限管理

- 数据操作：
  - SELECT
  - INSERT
  - DELETE
  - UPDATE
- 字段级别：
  - SELECT(col1,col2,...)
  - UPDATE(col1,col2,...)
  - INSERT(col1,col2,...)
- 所有权限：ALL PRIVILEGES 或 ALL

授权

- 参考：https://dev.mysql.com/doc/refman/5.7/en/grant.html
- GRANT priv_type [(column_list)],... ON [object_type] priv_level TO 'user'@'host' [IDENTIFIED BY 'password'] [WITH GRANT OPTION];
  - priv_type: ALL [PRIVILEGES]
  - object_type:TABLE | FUNCTION | PROCEDURE
  - priv_level: *(所有库) | *.* | db_name.* | db_name.tbl_name | tbl_name(当前库的表) | db_name.routine_name(指定库的函数,存储过程,触发器)
  - with_option: GRANT OPTION

  | MAX_QUERIES_PER_HOUR count

  | MAX_UPDATES_PER_HOUR count

  | MAX_CONNECTIONS_PER_HOUR count

  | MAX_USER_CONNECTIONS count

  示例：GRANT SELECT (col1), INSERT (col1,col2) ON mydb.mytbl TO 'someuser'@'somehost';

授权

- 回收授权：REVOKE priv_type [(column_list)] [, priv_type [(column_list)]] ... ON [object_type] priv_level FROM user [, user] ...

  示例：

  REVOKE DELETE ON testdb.* FROM 'testuser'@'% ‘

- 查看指定用户获得的授权：

  Help SHOW GRANTS

  SHOW GRANTS FOR 'user'@'host';

  SHOW GRANTS FOR CURRENT_USER[()];

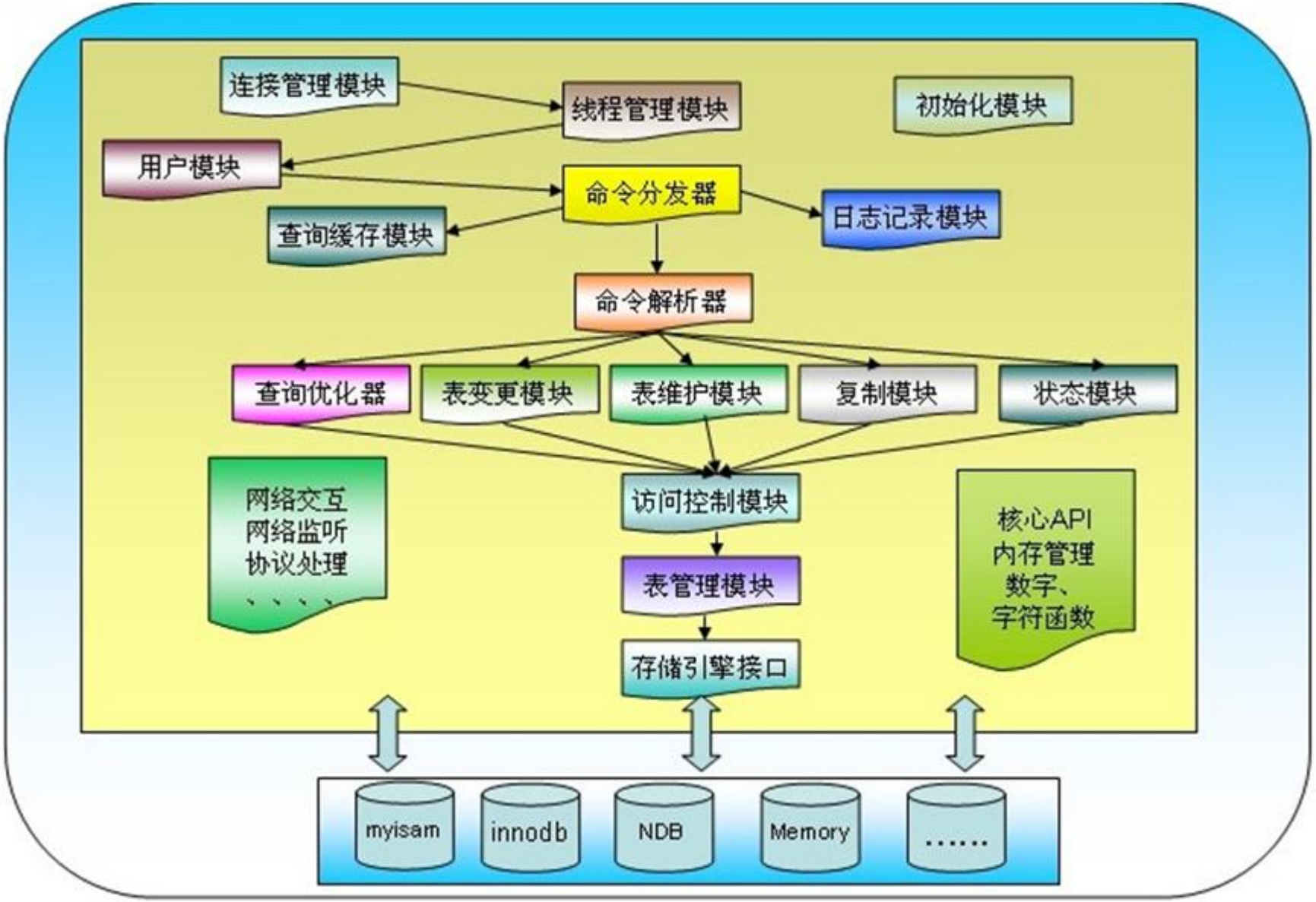
- 注意：MariaDB服务进程启动时会读取mysql库中所有授权表至内存

  (1) GRANT或REVOKE等执行权限操作会保存于系统表中，MariaDB的服务进程通常会自动重读授权表，使之生效

  (2) 对于不能够或不能及时重读授权表的命令，可手动让MariaDB的服务进程重读授权表：

  mysql> FLUSH PRIVILEGES;

MySQL体系结构


连接管理模块
线程管理模块
初始化模块
用户模块
命令分发器
日志记录模块
查询缓存模块
命令解析器
查询优化器
表变更模块
表维护模块
复制模块
状态模块
网络交互
网络监听
协议处理
、、、、
访问控制模块
表管理模块
存储引擎接口
核心API
内存管理
数字、
字符函数
myisam
innodb
NDB
Memory
.....

存储引擎

Feature	MyISAM	Memory	InnoDB	Archive	NDB
Storage limits	256TB	RAM	64TB	None	384EB
Transactions	No	No	Yes	No	Yes
Locking granularity	Table	Table	Row	Row	Row
MVCC	No	No	Yes	No	No
Geospatial data type support	Yes	No	Yes	Yes	Yes
Geospatial indexing support	Yes	No	Yes [a]	No	No
B-tree indexes	Yes	Yes	Yes	No	No
T-tree indexes	No	No	No	No	Yes
Hash indexes	No	Yes	No [b]	No	Yes
Full-text search indexes	Yes	No	Yes [c]	No	No
Clustered indexes	No	No	Yes	No	No
Data caches	No	N/A	Yes	No	Yes
Index caches	Yes	N/A	Yes	No	Yes
Compressed data	Yes [d]	No	Yes [e]	Yes	No
Encrypted data [f]	Yes	Yes	Yes	Yes	Yes
Cluster database support	No	No	No	No	Yes
Replication support [g]	Yes	Limited [h]	Yes	Yes	Yes
Foreign key support	No	No	Yes	No	Yes [i]
Backup / point-in-time recovery [j]	Yes	Yes	Yes	Yes	Yes
Query cache support	Yes	Yes	Yes	Yes	Yes
Update statistics for data dictionary	Yes	Yes	Yes	Yes	Yes

InnoDB support for FULLTEXT indexes is available in MySQL 5.6.4 and later.

存储引擎比较：https://docs.oracle.com/cd/E17952_01/mysql-5.5-en/storage-engines.html

存储引擎

- MyISAM引擎特点：
  - 不支持事务
  - 表级锁定
  - 读写相互阻塞，写入不能读，读时不能写
  - 只缓存索引
  - 不支持外键约束
  - 不支持聚簇索引
  - 读取数据较快，占用资源较少
  - 不支持MVCC（多版本并发控制机制）高并发
  - 崩溃恢复性较差
  - MySQL5.5.5前默认的数据库引擎

存储引擎

- 适用场景：只读（或者写较少）、表较小（可以接受长时间进行修复操作）
- MyISAM引擎文件：

  tbl_name.frm: 表格式定义

  tbl_name.MYD: 数据文件

  tbl_name.MYI: 索引文件

存储引擎

- InnoDB引擎特点：
  - 支持事务，适合处理大量短期事务
  - 行级锁
  - 读写阻塞与事务隔离级别相关
  - 可缓存数据和索引
  - 支持聚簇索引
  - 崩溃恢复性更好
  - 支持MVCC高并发
  - 从MySQL5.5后支持全文索引
  - 从MySQL5.5.5开始为默认的数据库引擎

存储引擎

- InnoDB数据库文件
  - 所有InnoDB表的数据和索引放置于同一个表空间中
    表空间文件：datadir定义的目录下
    数据文件：ibddata1, ibddata2, ...
  - 每个表单独使用一个表空间存储表的数据和索引
    启用：innodb_file_per_table=ON
    两类文件放在数据库独立目录中
      数据文件(存储数据和索引)：tb_name.ibd
      表格式定义：tb_name.frm

其它存储引擎

- Performance_Schema:Performance_Schema数据库
- Memory ：将所有数据存储在RAM中，以便在需要快速查找参考和其他类似数据的环境中进行快速访问。适用存放临时数据。引擎以前被称为HEAP引擎
- MRG_MyISAM：使MySQL DBA或开发人员能够对一系列相同的MyISAM表进行逻辑分组，并将它们作为一个对象引用。适用于VLDB(Very Large Data Base)环境，如数据仓库
- Archive ：为存储和检索大量很少参考的存档或安全审核信息，只支持SELECT和INSERT操作；支持行级锁和专用缓存区
- Federated联合：用于访问其它远程MySQL服务器一个代理，它通过创建一个到远程MySQL服务器的客户端连接，并将查询传输到远程服务器执行，而后完成数据存取，提供链接单独MySQL服务器的能力，以便从多个物理服务器创建一个逻辑数据库。非常适合分布式或数据集市环境

其它数据库引擎

- BDB：可替代InnoDB的事务引擎，支持COMMIT、ROLLBACK和其他事务特性
- Cluster/NDB：MySQL的簇式数据库引擎，尤其适合于具有高性能查找要求的应用程序，这类查找需求还要求具有最高的正常工作时间和可用性
- CSV：CSV存储引擎使用逗号分隔值格式将数据存储在文本文件中。可以使用CSV引擎以CSV格式导入和导出其他软件和应用程序之间的数据交换
- BLACKHOLE ：黑洞存储引擎接受但不存储数据，检索总是返回一个空集。该功能可用于分布式数据库设计，数据自动复制，但不是本地存储
- example：“stub”引擎，它什么都不做。可以使用此引擎创建表，但不能将数据存储在其中或从中检索。目的是作为例子来说明如何开始编写新的存储引擎

管理存储引擎

- 查看mysql支持的存储引擎:
  show engines;
- 查看当前默认的存储引擎:
  show variables like '%storage_engine%';
- 设置默认的存储引擎：
  vim /etc/my.conf
  [mysqld]
  default_storage_engine= InnoDB;

管理存储引擎

- 查看库中所有表使用的存储引擎

  Show table status from db_name;

- 查看库中指定表的存储引擎

  show table status like ' tb_name ';

  show create table tb_name;

- 设置表的存储引擎：

  CREATE TABLE tb_name(...) ENGINE=InnoDB;

  ALTER TABLE tb_name ENGINE=InnoDB;

MySQL中的系统数据库

- mysql数据库：是mysql的核心数据库，类似于sql server中的master库，主要负责存储数据库的用户、权限设置、关键字等mysql自己需要使用的控制和管理信息
- PERFORMANCE_SCHEMA:MySQL 5.5开始新增的数据库，主要用于收集数据库服务器性能参数,库里表的存储引擎均为PERFORMANCE_SCHEMA，用户不能创建存储引擎为PERFORMANCE_SCHEMA的表
- information_schema数据库:MySQL 5.0之后产生的，一个虚拟数据库，物理上并不存在。information_schema数据库类似与“数据字典”，提供了访问数据库元数据的方式，即数据的数据。比如数据库名或表名，列类型，访问权限（更加细化的访问方式）

服务器配置

- mysqld选项，服务器系统变量和服务器状态变量

  https://dev.mysql.com/doc/refman/5.7/en/mysqld-option-tables.html

  https://mariadb.com/kb/en/library/full-list-of-mariadb-options-system-and-status-variables/

- 注意：其中有些参数支持运行时修改，会立即生效；有些参数不支持，且只能通过修改配置文件，并重启服务器程序生效；有些参数作用域是全局的，且不可改变；有些可以为每个用户提供单独（会话）的设置
- 获取mysqld的可用选项列表：

  mysqld --help –verbose

  mysqld --print-defaults 获取默认设置

服务器配置

- 获取运行中的mysql进程使用各服务器参数及其值

  mysql> SHOW GLOBAL VARIABLES;

  mysql> SHOW [SESSION] VARIABLES;

- 设置服务器系统变量三种方法：
  - 在命令行中设置:

    shell> ./mysqld_safe --aria_group_commit="hard “
  - 在配置文件my.cnf中设置：

    aria_group_commit = "hard"
  - 在mysql客户端使用SET命令：

    SET GLOBAL aria_group_commit="hard";

服务器端设置

- 修改服务器变量的值：

```
mysql> help SET
```

- 修改全局变量：仅对修改后新创建的会话有效；对已经建立的会话无效

```
mysql> SET GLOBAL system_var_name=value;
mysql> SET @@global.system_var_name=value;
```

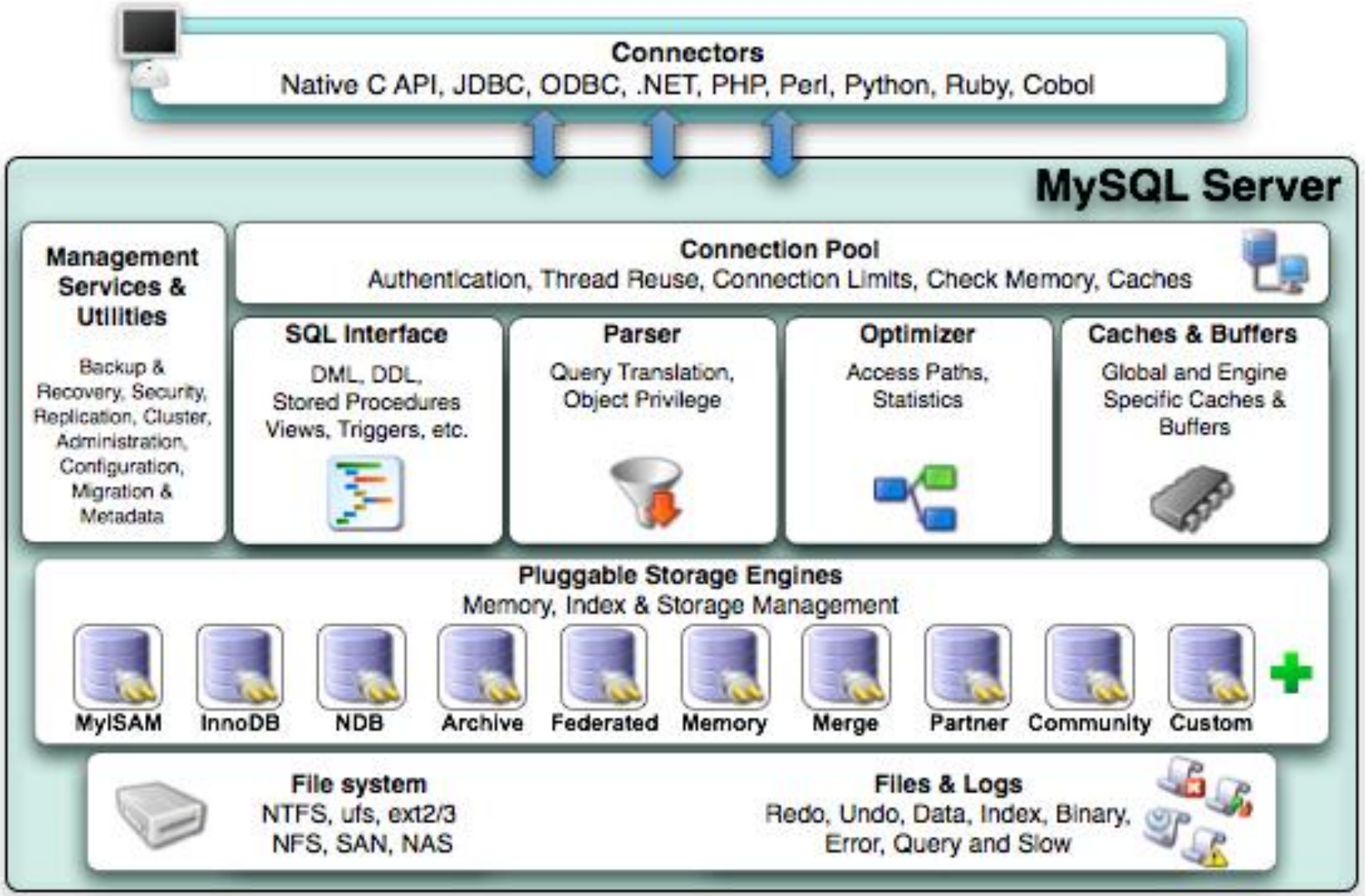
- 修改会话变量：

```
mysql> SET [SESSION] system_var_name=value;
mysql> SET @@[session.]system_var_name=value;
```

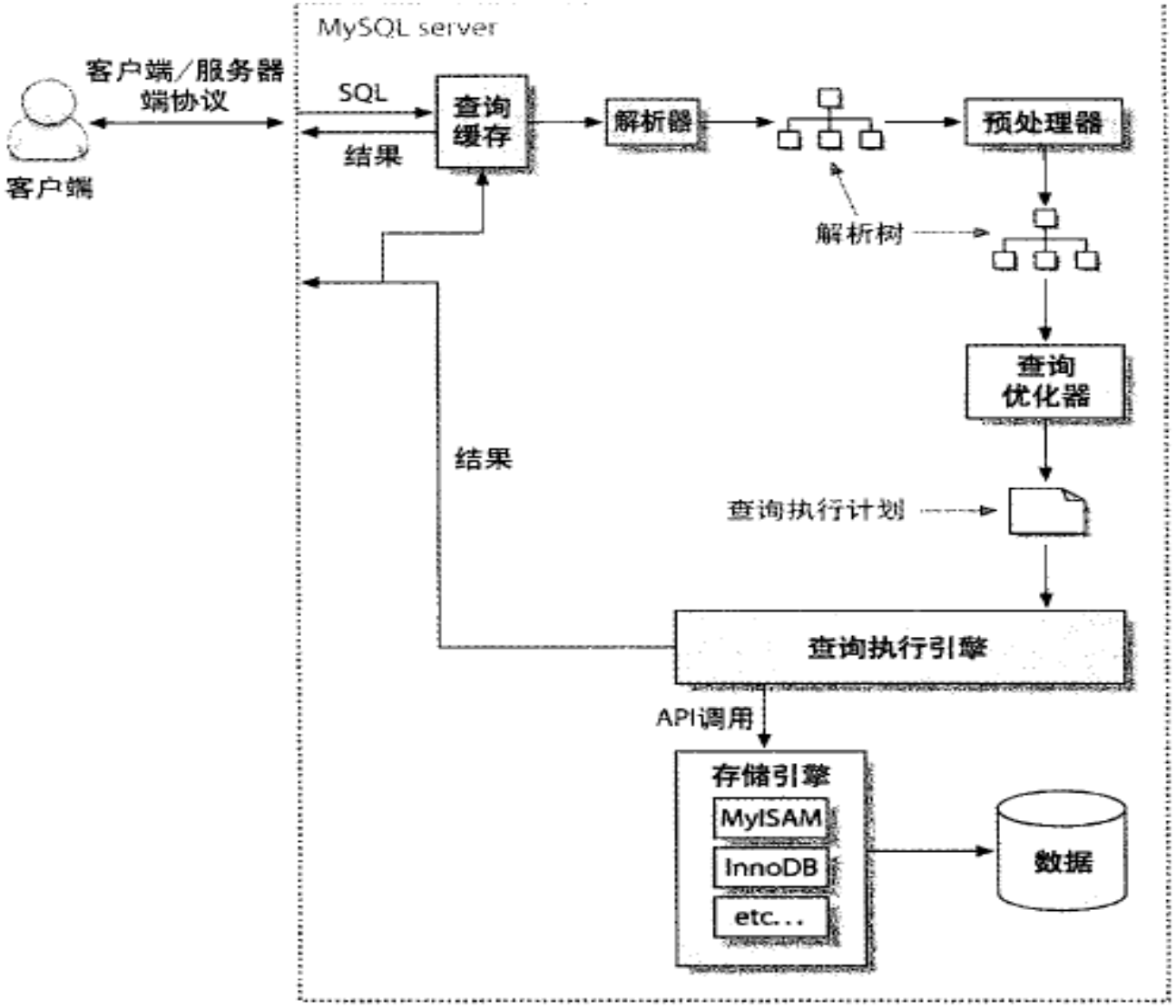
- 状态变量（只读）：用于保存mysqld运行中的统计数据的变量，不可更改

```
mysql> SHOW GLOBAL STATUS;
mysql> SHOW [SESSION] STATUS;
```

MySQL架构



查询的执行路径



查询缓存

- 查询缓存（ Query Cache ）原理：

缓存SELECT操作或预处理查询的结果集和SQL语句，当有新的SELECT语句或预处理查询语句请求，先去查询缓存，判断是否存在可用的记录集，判断标准：与缓存的SQL语句，是否完全一样，区分大小写

- 优缺点

不需要对SQL语句做任何解析和执行，当然语法解析必须通过在先，直接从Query Cache中获得查询结果，提高查询性能

查询缓存的判断规则，不够智能，也即提高了查询缓存的使用门槛，降低其效率；

查询缓存的使用，会增加检查和清理Query Cache中记录集的开销

查询缓存

- 哪些查询可能不会被缓存
  - 查询语句中加了SQL_NO_CACHE参数
  - 查询语句中含有获得值的函数，包含自定义函数，如：NOW()
    CURDATE()、GET_LOCK()、RAND()、CONVERT_TZ()等
  - 对系统数据库的查询：mysql、information_schema 查询语句中使用SESSION级别变量或存储过程中的局部变量
  - 查询语句中使用了LOCK IN SHARE MODE、FOR UPDATE的语句，查询语句中类似SELECT ...INTO 导出数据的语句
  - 对临时表的查询操作；存在警告信息的查询语句；不涉及任何表或视图的查询语句；某用户只有列级别权限的查询语句
  - 事务隔离级别为Serializable时，所有查询语句都不能缓存

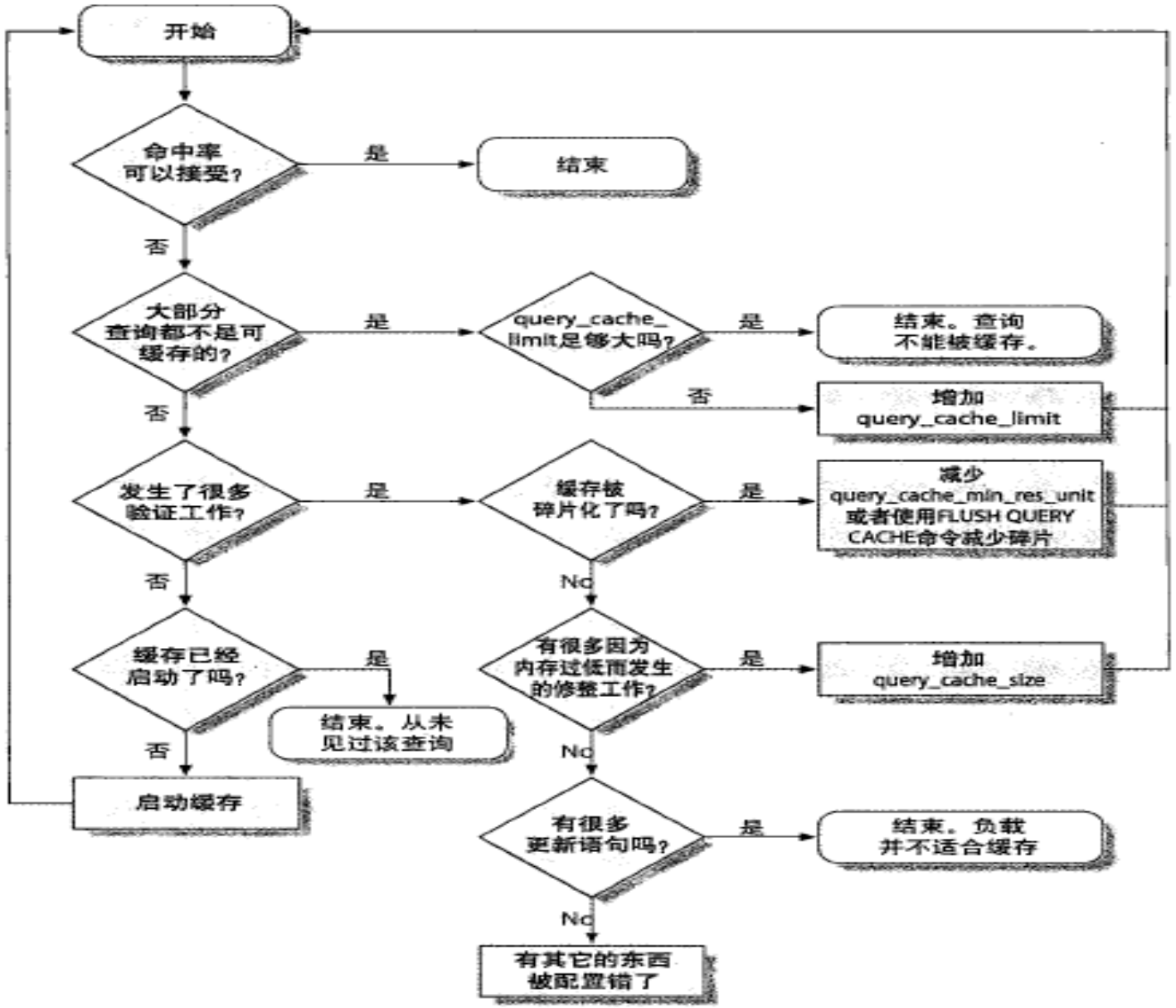
查询缓存

- 查询缓存相关的服务器变量
  - query_cache_min_res_unit: 查询缓存中内存块的最小分配单位，默认4k，较小值会减少浪费，但会导致更频繁的内存分配操作，较大值会带来浪费，会导致碎片过多，内存不足
  - query_cache_limit：单个查询结果能缓存的最大值，默认为1M，对于查询结果过大而无法缓存的语句，建议使用SQL_NO_CACHE
  - query_cache_size：查询缓存总共可用的内存空间；单位字节，必须是1024的整数倍，最小值40KB，低于此值有警报
  - query_cache_wlock_invalidate：如果某表被其它的会话锁定，是否仍然可以从查询缓存中返回结果，　默认值为OFF，表示可以在表被其它会话锁定的场景中继续从缓存返回数据；ON则表示不允许
  - query_cache_type: 是否开启缓存功能，取值为ON, OFF, DEMAND

查询缓存

- SELECT语句的缓存控制
  - SQL_CACHE: 显式指定存储查询结果于缓存之中
  - SQL_NO_CACHE: 显式查询结果不予缓存
- query_cache_type参数变量：
  - query_cache_type的值为OFF或0时，查询缓存功能关闭
  - query_cache_type的值为ON或1时，查询缓存功能打开，SELECT的结果符合缓存条件即会缓存，否则，不予缓存，显式指定SQL_NO_CACHE，不予缓存，此为默认值
  - query_cache_type的值为DEMAND或2时，查询缓存功能按需进行，显式指定SQL_CACHE的SELECT语句才会缓存；其它均不予缓存
  - 参看：https://mariadb.com/kb/en/library/server-system-variables/#query_cache_type
    https://dev.mysql.com/doc/refman/5.7/en/query-cache-configuration.html

优化查询缓存



查询缓存

- 查询缓存相关的状态变量：SHOW GLOBAL STATUS LIKE ‘Qcache%';
  - Qcache_free_blocks：处于空闲状态 Query Cache中内存 Block 数
  - Qcache_free_memory：处于空闲状态的 Query Cache 内存总量
  - Qcache_hits：Query Cache 命中次数
  - Qcache_inserts：向 Query Cache 中插入新的 Query Cache 的次数，即没有命中的次数
  - Qcache_lowmem_prunes：当 Query Cache 内存容量不够，需要删除老的 Query Cache 以给新的 Cache 对象使用的次数
  - Qcache_not_cached：没有被 Cache 的 SQL 数，包括无法被 Cache 的 SQL 以及由于 query_cache_type 设置的不会被 Cache 的 SQL语句
  - Qcache_queries_in_cache：在 Query Cache 中的 SQL 数量
  - Qcache_total_blocks：Query Cache 中总的 Block

命中率和内存使用率估算

- 查询缓存中内存块的最小分配单位query_cache_min_res_unit ：(query_cache_size - Qcache_free_memory) / Qcache_queries_in_cache
- 查询缓存命中率 ：Qcache_hits / (Qcache_hits + Qcache_inserts) * 100%
- 查询缓存内存使用率：(query_cache_size – qcache_free_memory) / query_cache_size * 100%

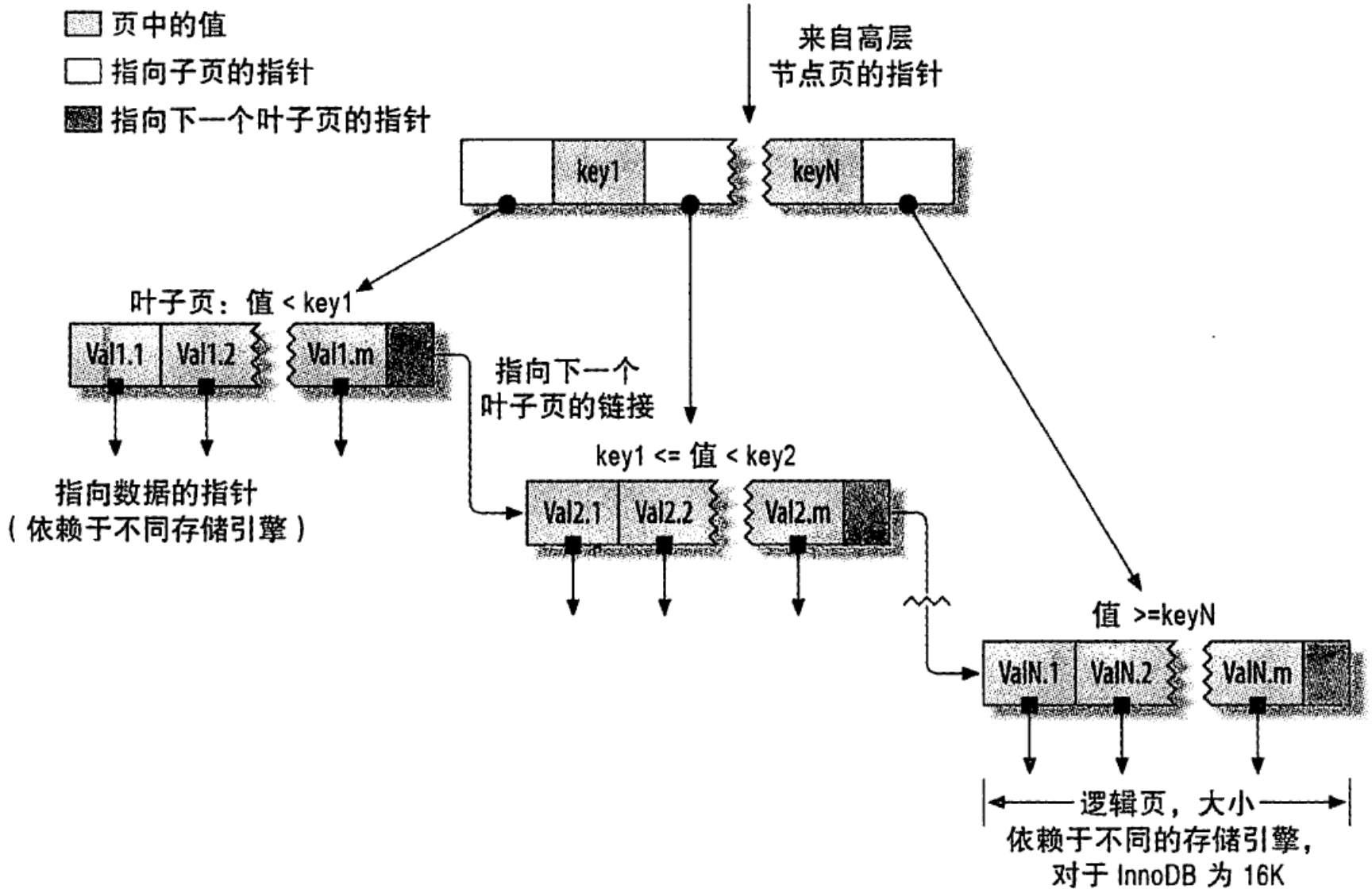
索引

- 索引是特殊数据结构：定义在查找时作为查找条件的字段
- 优点：提高查询速度，缺点：占用额外空间，影响插入速度
- 索引实现在存储引擎
- 索引类型：
  - B+ TREE、 HASH、 R TREE
  - 聚簇（集）索引、非聚簇索引：数据是否与索引存储在一起
  - 主键索引、辅助索引
  - 稠密索引、稀疏索引：是否索引了每一个数据项
  - 简单索引、组合索引

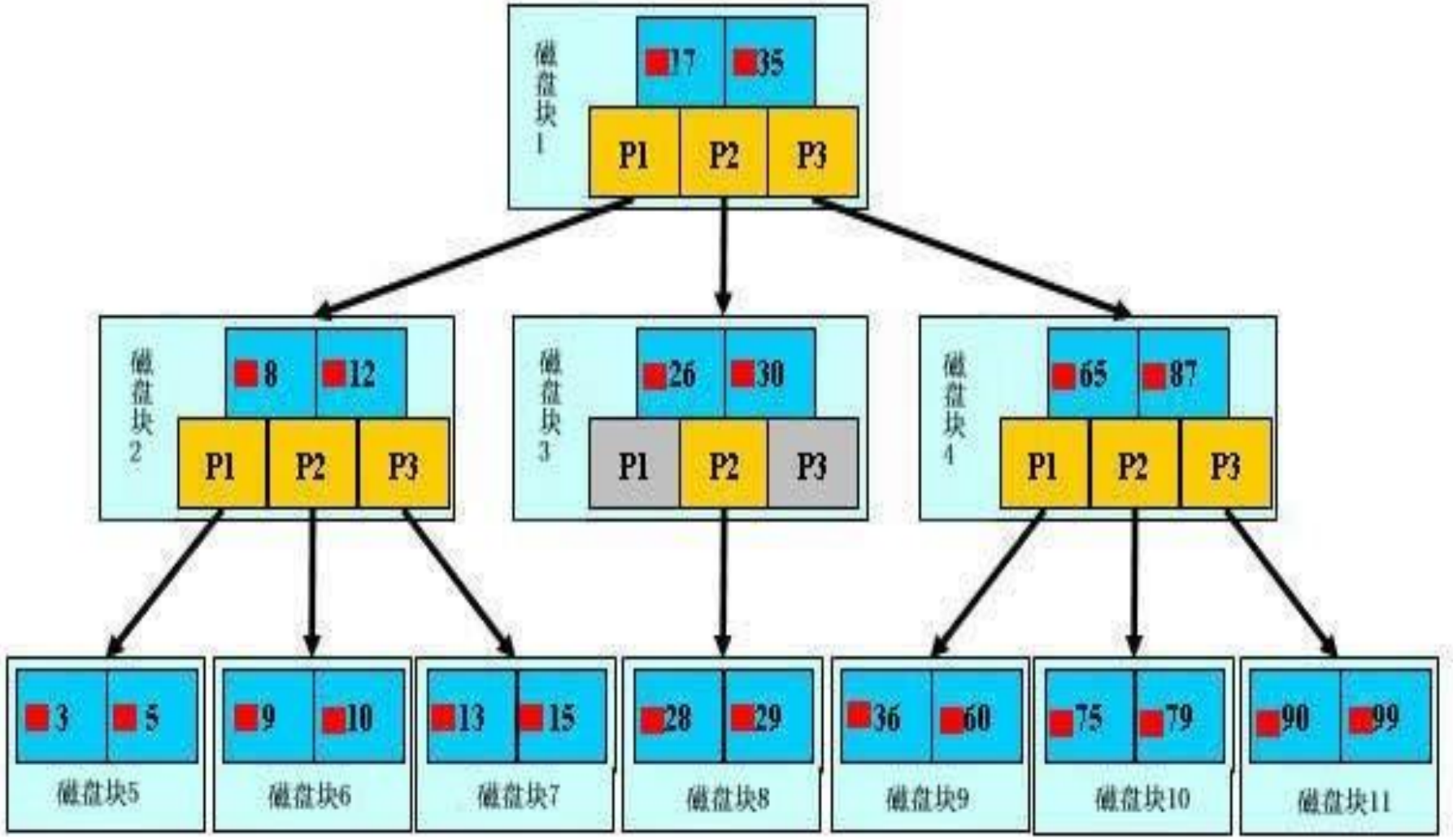
    左前缀索引：取前面的字符做索引

    覆盖索引：从索引中即可取出要查询的数据，性能高

索引B+TREE



索引B+tree



B-Tree索引

- B+ Tree索引：顺序存储，每一个叶子节点到根结点的距离是相同的；左前缀索引，适合查询范围类的数据
- 可以使用B-Tree索引的查询类型：

  全值匹配：精确所有索引列，如：姓wang，名xiaochun，年龄30

  匹配最左前缀：即只使用索引的第一列，如：姓wang

  匹配列前缀：只匹配一列值开头部分，如：姓以w开头的

  匹配范围值：如：姓ma和姓wang之间

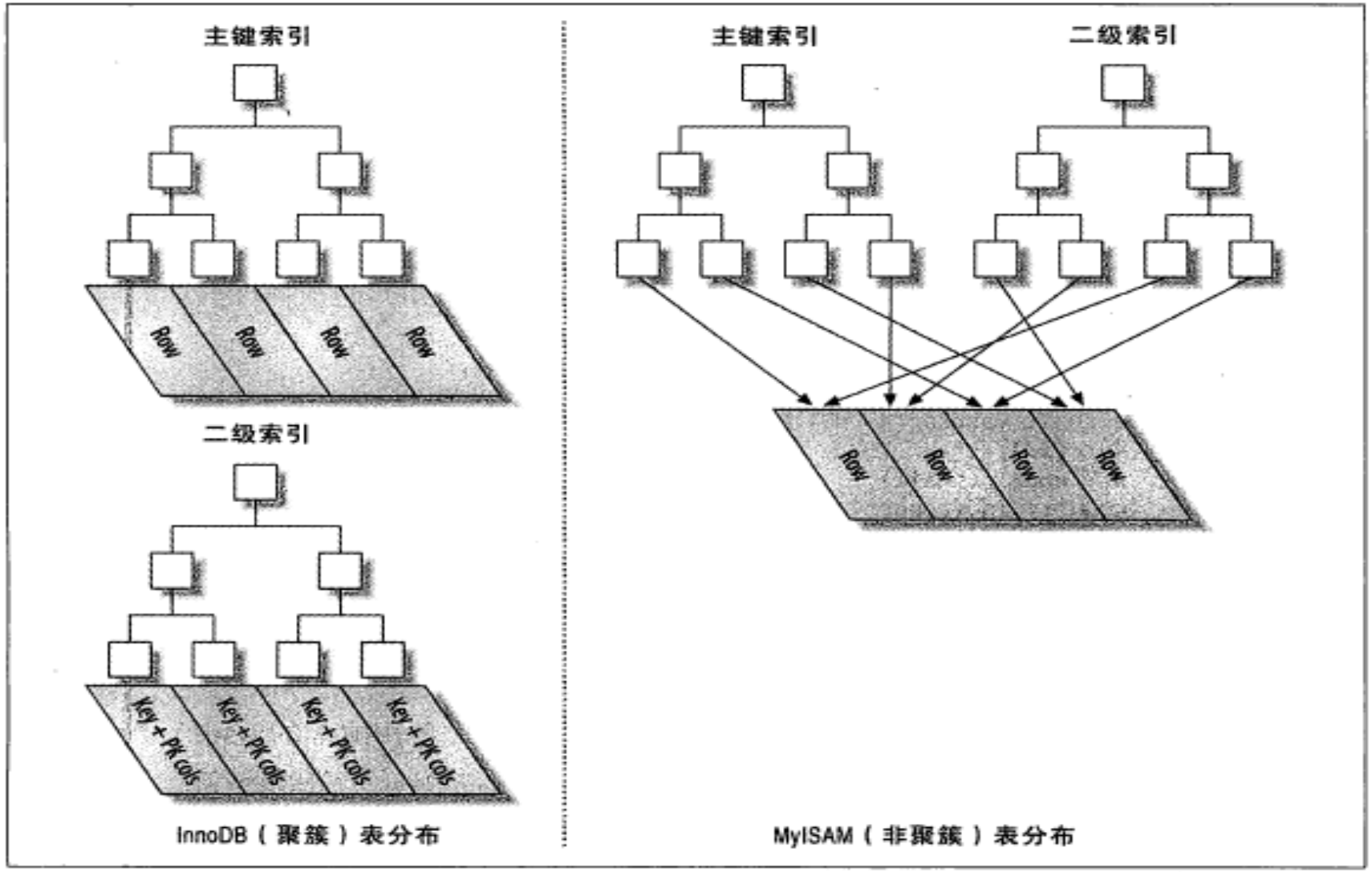
  精确匹配某一列并范围匹配另一列：如：姓wang,名以x开头的

  只访问索引的查询

B-Tree索引

- B-Tree索引的限制：
  - 如果不从最左列开始，则无法使用索引：如：查找名为xiaochun，或姓为g结尾
  - 不能跳过索引中的列：如：查找姓wang,年龄30的，只能使用索引第一列
  - 如果查询中某个列是为范围查询，那么其右侧的列都无法再使用索引：如：姓wang,名x%,年龄30，只能利用姓和名上面的索引
- 特别提示：
  - 索引列的顺序和查询语句的写法应相匹配，才能更好的利用索引
  - 为优化性能，可能需要针对相同的列但顺序不同创建不同的索引来满足不同类型的查询需求

聚簇和非聚簇索引，主键和二级索引



索引

- 高性能索引策略：
  - 独立地使用列：尽量避免其参与运算，独立的列指索引列不能是表达式的一部分，也不能是函数的参数，在where条件中，始终将索引列单独放在比较符号的一侧
  - 左前缀索引：构建指定索引字段的左侧的字符数，要通过索引选择性来评估

    索引选择性：不重复的索引值和数据表的记录总数的比值
  - 多列索引：AND操作时更适合使用多列索引，而非为每个列创建单独的索引
  - 选择合适的索引列顺序：无排序和分组时，将选择性最高放左侧
- 冗余和重复索引：（A），（A，B）即为冗余索引

  不好的索引使用策略，建议扩展索引，而非冗余

索引优化建议

- 只要列中含有NULL值，就最好不要在此例设置索引，复合索引如果有NULL值，此列在使用时也不会使用索引
- 尽量使用短索引，如果可以，应该制定一个前缀长度
- 对于经常在where子句使用的列，最好设置索引
- 对于有多个列where或者order by子句，应该建立复合索引
- 对于like语句，以%或者‘-’开头的不会使用索引，以%结尾会使用索引
- 尽量不要在列上进行运算（函数操作和表达式操作）
- 尽量不要使用not in和<>操作

管理索引

- 创建索引：

  CREATE INDEX index_name ON tbl_name (index_col_name,...);

  help CREATE INDEX

- 删除索引：

  DROP INDEX index_name ON tbl_name;

- 查看索引：

  SHOW INDEXES FROM [db_name.]tbl_name;

- 优化表空间：

  OPTIMIZE TABLE tb_name

日志

- 日志
  - 事务日志：transaction log
  - 错误日志：error log
  - 查询日志：query log
  - 慢查询日志：slow query log
  - 二进制日志：binary log
  - 中继日志：reley log

日志

- 事务日志：transaction log
  - 事务型存储引擎自行管理和使用
    redo log
    undo log
  - Innodb事务日志相关配置：
    show variables like '%innodb_log%';
    innodb_log_file_size　5242880　每个日志文件大小
    innodb_log_files_in_group　2　　日志组成员个数
    innodb_log_group_home_dir　./　事务文件路径
- 中继日志：relay log
  主从复制架构中，从服务器用于保存从主服务器的二进制日志中读取到的事件

日志

- 错误日志
  - mysqld启动和关闭过程中输出的事件信息
  - mysqld运行中产生的错误信息
  - event scheduler运行一个event时产生的日志信息
  - 在主从复制架构中的从服务器上启动从服务器线程时产生的信息
- 错误日志相关配置
  - SHOW GLOBAL VARIABLES LIKE 'log_error'
  - 错误文件路径：

log_error=/PATH/TO/LOG_ERROR_FILE

  - 是否记录警告信息至错误日志文件
  - log_warnings=1|0 默认值1

日志

- 查询日志：记录查询操作
  - 文件：file，默认值
  - 表：table
- 查询日志相关设置

  general_log=ON|OFF

  general_log_file=HOSTNAME.log

  log_output=TABLE|FILE|NONE

慢查询日志

- 慢查询日志：记录执行查询时长超出指定时长的操作

slow_query_log=ON|OFF　　开启或关闭慢查询

long_query_time=N　　慢查询的阀值，单位秒

slow_query_log_file=HOSTNAME-slow.log　慢查询日志文件

log_slow_filter = admin,filesort,filesort_on_disk,full_join,

full_scan,query_cache,query_cache_miss,tmp_table,tmp_table_on_disk

log_queries_not_using_indexes=ON　不使用索引也没有达到慢查询阀值的语句是否记录日志，默认OFF，即不记录

log_slow_rate_limit = 1　多少次查询才记录，mariadb特有

log_slow_verbosity= Query_plan,explain　　记录内容log_slow_queries = OFF 同slow_query_log　新版已废弃

日志

- 二进制日志

  记录导致数据改变或潜在导致数据改变的SQL语句

  功能：通过“重放”日志文件中的事件来生成数据副本

  注意：建议二进制日志和数据文件分开存放

- 二进制日志相关配置

  查看mariadb自行管理使用中的二进制日志文件列表

  SHOW {BINARY | MASTER} LOGS

  查看使用中的二进制日志文件

  SHOW MASTER STATUS

  查看二进制文件中的指定内容

  SHOW BINLOG EVENTS [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]

  show binlog events in 'mariadb-bin.000001' from 6516 limit 2,3

日志

- 二进制日志记录格式
  - 二进制日志记录三种格式

    基于“语句”记录：statement，记录语句，默认模式

    基于“行”记录：row，记录数据，日志量较大

    混合模式：mixed, 让系统自行判定该基于哪种方式进行
  - 格式配置

    show variables like '%binlog_format%';
- 二进制日志文件的构成

  有两类文件

  日志文件：mysql|mariadb-bin.文件名后缀，二进制格式

  如： mysql-bin.000001

  索引文件：mysql|mariadb-bin.index，文本格式

日志

- 二进制日志相关的服务器变量：
  - sql_log_bin=ON|OFF：是否记录二进制日志，默认ON
  - log_bin=/PATH/BIN_LOG_FILE：指定文件位置；默认OFF，表示不启用二进制日志功能，上述两项都开启才可
  - binlog_format=STATEMENT|ROW|MIXED：二进制日志记录的格式，默认STATEMENT
  - max_binlog_size=1073741824：单个二进制日志文件的最大体积，到达最大值会自动滚动，默认为1G

    说明：文件达到上限时的大小未必为指定的精确值
  - sync_binlog=1|0：设定是否启动二进制日志即时同步磁盘功能，默认0，由操作系统负责同步日志到磁盘
  - expire_logs_days=N：二进制日志可以自动删除的天数。 默认为0，即不自动删除

日志

- mysqlbinlog：二进制日志的客户端命令工具
- 命令格式：

  mysqlbinlog [OPTIONS] log_file...
  --start-position=#　指定开始位置
  --stop-position=#
  --start-datetime=
  --stop-datetime=
  时间格式：YYYY-MM-DD hh:mm:ss
  --base64-output[=name]

示例：mysqlbinlog --start-position=6787 --stop-position=7527 /var/lib/mysql/mariadb-bin.000003

mysqlbinlog --start-datetime="2018-01-30 20:30:10" --stop-datetime="2018-01-30 20:35:22" mariadb-bin.000003;

日志

- 二进制日志事件的格式：

```
 # at 328
#151105 16:31:40 server id 1  end_log_pos 431   Query   thread_id=1     exec_time=0
error_code=0
use `mydb`/*!*/;
SET TIMESTAMP=1446712300/*!*/;
CREATE TABLE tb1 (id int, name char(30))
/*!*/;
```

事件发生的日期和时间：151105 16:31:40
事件发生的服务器标识：server id 1
事件的结束位置：end_log_pos 431
事件的类型：Query
事件发生时所在服务器执行此事件的线程的ID：thread_id=1
语句的时间戳与将其写入二进制文件中的时间差：exec_time=0
错误代码：error_code=0
事件内容：
GTID：Global Transaction ID，mysql5.6以mariadb10以上版本专属属性：GTID

日志

- 清除指定二进制日志：

  PURGE { BINARY | MASTER } LOGS

  { TO 'log_name' | BEFORE datetime_expr }

  示例：

  PURGE BINARY LOGS TO ‘mariadb-bin.000003’;删除3前日志

  PURGE BINARY LOGS BEFORE '2017-01-23';

  PURGE BINARY LOGS BEFORE '2017-03-22 09:25:30';

- 删除所有二进制日志，index文件重新记数

  RESET MASTER [TO #]; 日志文件从#开始记数，默认从1开始，一般是master第一次启动时执行，MariaDB10.1.6开始支持TO #

- 切换日志文件：

  FLUSH LOGS;

备份和恢复

- 为什么要备份

灾难恢复：硬件故障、软件故障、自然灾害、黑客攻击、误操作测试等数据丢失场景

- 备份注意要点
  - 能容忍最多丢失多少数据
  - 恢复数据需要在多长时间内完成
  - 需要恢复哪些数据
- 还原要点
  - 做还原测试，用于测试备份的可用性
  - 还原演练

备份和恢复

- 备份类型：
  - 完全备份，部分备份

    完全备份：整个数据集

    部分备份：只备份数据子集，如部分库或表
  - 完全备份、增量备份、差异备份

    增量备份：仅备份最近一次完全备份或增量备份（如果存在增量）以来变化的数据，备份较快，还原复杂

    差异备份：仅备份最近一次完全备份以来变化的数据，备份较慢，还原简单
- 注意：二进制日志文件不应该与数据文件放在同一磁盘

备份和恢复

- 冷、温、热备份
  - 冷备：读写操作均不可进行
  - 温备：读操作可执行；但写操作不可执行
  - 热备：读写操作均可执行

    MyISAM：温备，不支持热备

    InnoDB: 都支持
- 物理和逻辑备份
  - 物理备份：直接复制数据文件进行备份，与存储引擎有关，占用较多的空间，速度快
  - 逻辑备份：从数据库中“导出”数据另存而进行的备份，与存储引擎无关，占用空间少，速度慢，可能丢失精度

备份和恢复

- 备份时需要考虑的因素
  - 温备的持锁多久
    - 备份产生的负载
    - 备份过程的时长
  - 恢复过程的时长
- 备份什么
  - 数据
  - 二进制日志、InnoDB的事务日志
  - 程序代码（存储过程、存储函数、触发器、事件调度器）
  - 服务器的配置文件

备份和恢复

- 设计备份方案
  - 数据集：完全+增量
  - 备份手段：物理，逻辑
- 备份工具
  - mysqldump：逻辑备份工具，适用所有存储引擎，温备；支持完全或部分备份；对InnoDB存储引擎支持热备
  - cp, tar等复制归档工具：物理备份工具，适用所有存储引擎；只支持冷备；完全和部分备份
  - LVM的快照：先加锁，做快照后解锁，几乎热备；借助文件系统管理工具进行备份
  - mysqlhotcopy：几乎冷备；仅适用于MyISAM存储引擎

备份和恢复

- 备份工具的选择：
  - mysqldump+复制binlog：
    mysqldump：完全备份
    复制binlog中指定时间范围的event：增量备份
  - LVM快照+复制binlog：
    LVM快照：使用cp或tar等做物理备份；完全备份
    复制binlog中指定时间范围的event：增量备份
  - xtrabackup：由Percona提供支持对InnoDB做热备(物理备份)的工具，支持完全备份、增量备份
  - MariaDB Backup： 从MariaDB 10.1.26开始集成，基于Percona XtraBackup 2.3.8实现
  - mysqlbackup：热备份， MySQL Enterprise Edition组件

备份和恢复

- 逻辑备份工具：mysqldump, mydumper, phpMyAdmin
- Schema和数据存储在一起、巨大的SQL语句、单个巨大的备份文件
- mysqldump工具：客户端命令，通过mysql协议连接至mysqld服务器进行备份

  mysqldump [OPTIONS] database [tables]

  mysqldump [OPTIONS] –B DB1 [DB2 DB3...]

  mysqldump [OPTIONS] –A [OPTIONS]
- mysqldump参考：https://dev.mysql.com/doc/refman/5.7/en/mysqldump.html

备份和恢复

- mysqldump常见选项：
  - -A， --all-databases 备份所有数据库，含create database
  - -B , --databases db_name… 指定备份的数据库，包括create database语句
  - -E, --events：备份相关的所有event scheduler
  - -R, --routines：备份所有存储过程和存储函数
  - --triggers：备份表相关的触发器，默认启用,用--skip-triggers，不备份触发器
  - --master-data[=#]： 此选项须启用二进制日志

    1：所备份的数据之前加一条记录为CHANGE MASTER TO语句，非注释，不指定#，默认为1

    2：记录为注释的CHANGE MASTER TO语句

    此选项会自动关闭--lock-tables功能，自动打开--lock-all-tables功能（除非开启--single-transaction）

备份和恢复

- mysqldump常见选项
  - -F, --flush-logs ：备份前滚动日志，锁定表完成后，执行flush logs命令,生成新的二进制日志文件，配合-A时，会导致刷新多次数据库，在同一时刻执行转储和日志刷新，则应同时使用--flush-logs和-x，--master-data或-single-transaction,此时只刷新一次

    建议：和-x，--master-data或 --single-transaction一起使用
  - --compact 去掉注释，适合调试，生产不使用
  - -d, --no-data 只备份表结构
  - -t, --no-create-info 只备份数据,不备份create table
  - -n,--no-create-db 不备份create database，可被-A或-B覆盖
  - --flush-privileges 备份mysql或相关时需要使用
  - -f, --force 忽略SQL错误，继续执行
  - --hex-blob使用十六进制符号转储二进制列（例如，“abc”变为0x616263），受影响的数据类型包括BINARY，VARBINARY，BLOB，BIT
  - -q, --quick 不缓存查询，直接输出，加快备份速度

备份和恢复

- MyISAM备份选项：

  支持温备；不支持热备，所以必须先锁定要备份的库，而后启动备份操作

  锁定方法如下：

  -x,--lock-all-tables：加全局读锁，锁定所有库的所有表，同时加--single-transaction或--lock-tables选项会关闭此选项功能

  注意：数据量大时，可能会导致长时间无法并发访问数据库

  -l,--lock-tables：对于需要备份的每个数据库，在启动备份之前分别锁定其所有表，默认为on,--skip-lock-tables选项可禁用,对备份MyISAM的多个库,可能会造成数据不一致

  注：以上选项对InnoDB表一样生效，实现温备，但不推荐使用

备份和恢复

- InnoDB备份选项：

  支持热备，可用温备但不建议用

  --single-transaction

  此选项Innodb中推荐使用，不适用MyISAM，此选项会开始备份前，先执行START TRANSACTION指令，并且在备份期间，不允许对数据进行修改操作

  此选项和--lock-tables（此选项隐含提交挂起的事务）选项是相互排斥

  备份大型表时，建议将--single-transaction选项和--quick结合一起使用

生产备份策略

- InnoDB建议备份策略

  mysqldump –uroot –A –F –E –R --single-transaction --master-data=1 --flush-privileges --triggers --hex-blob >$BACKUP_DIR/fullbak_$BACKUP_TIME.sql

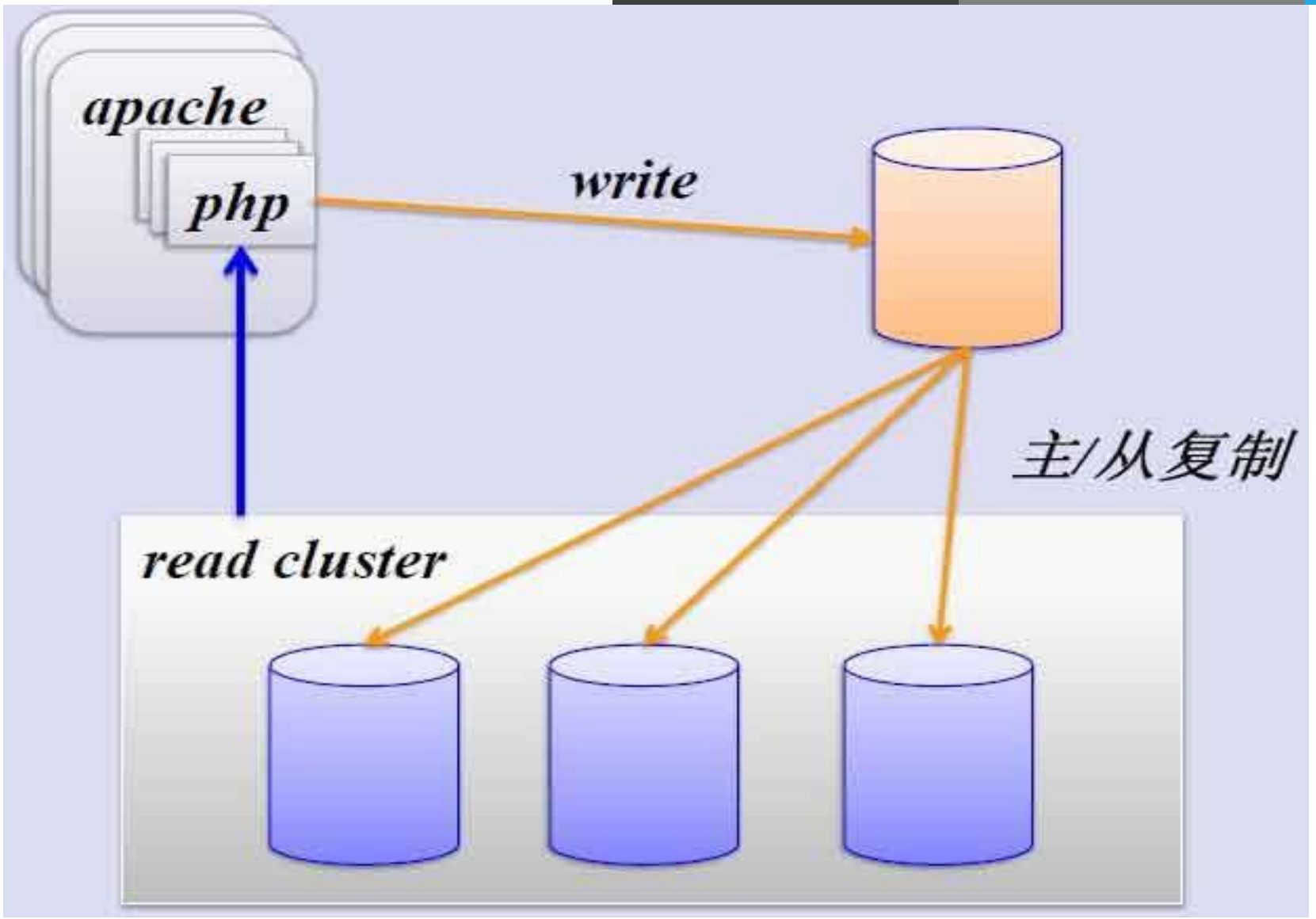
- MyISAM建议备份策略

  mysqldump –uroot –A –F –E –R –x --master-data=1 --flush-privileges --triggers --hex-blob >$BACKUP_DIR/fullbak_$BACKUP_TIME.sql

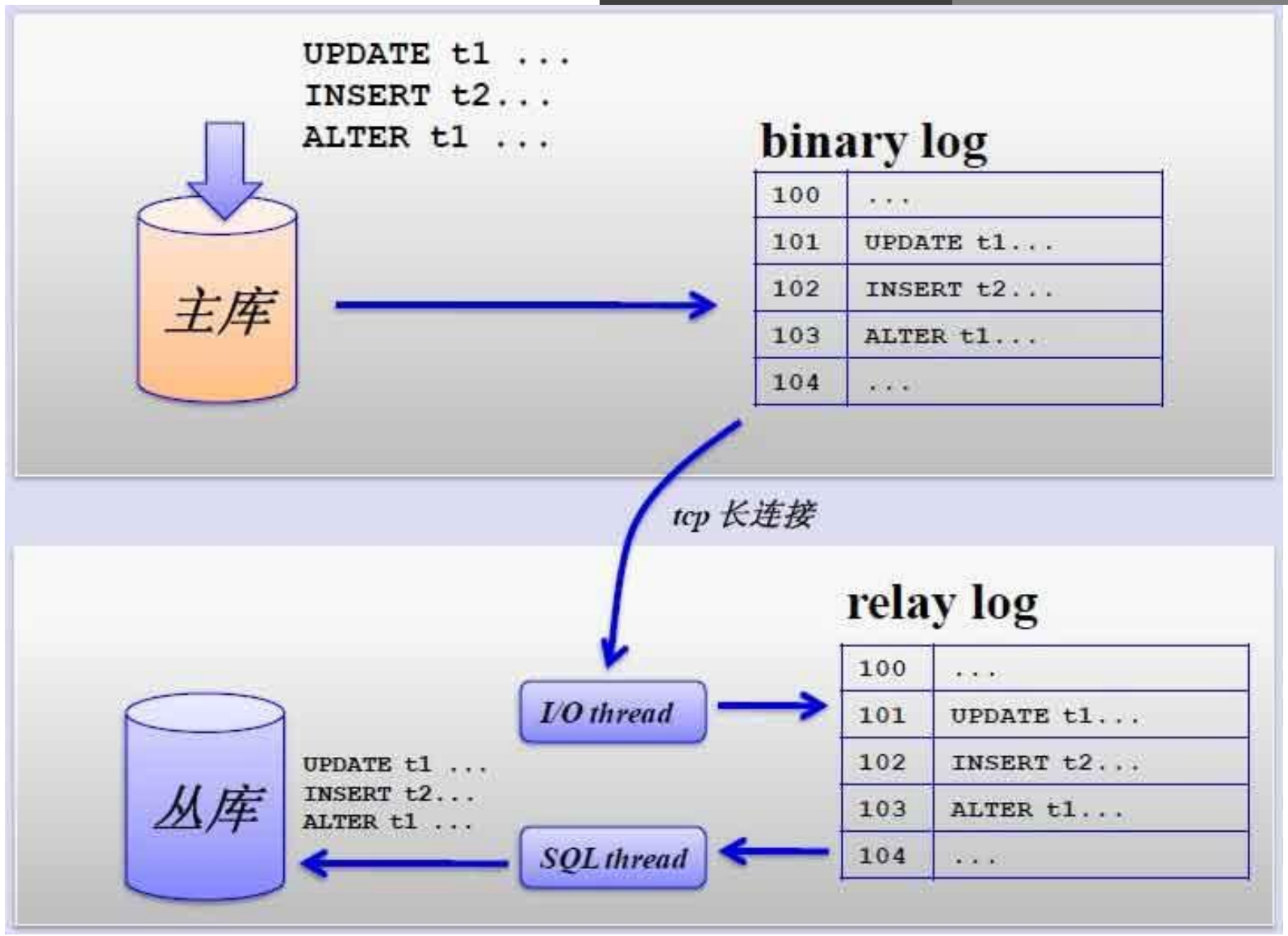
MySQL复制

- 扩展方式： Scale Up ，Scale Out
- MySQL的扩展
  - 复制：每个节点都有相同的数据集
    - 向外扩展
    - 二进制日志
    - 单向
- 复制的功用：
  - 数据分布
  - 负载均衡读
  - 备份
  - 高可用和故障切换
  - MySQL升级测试

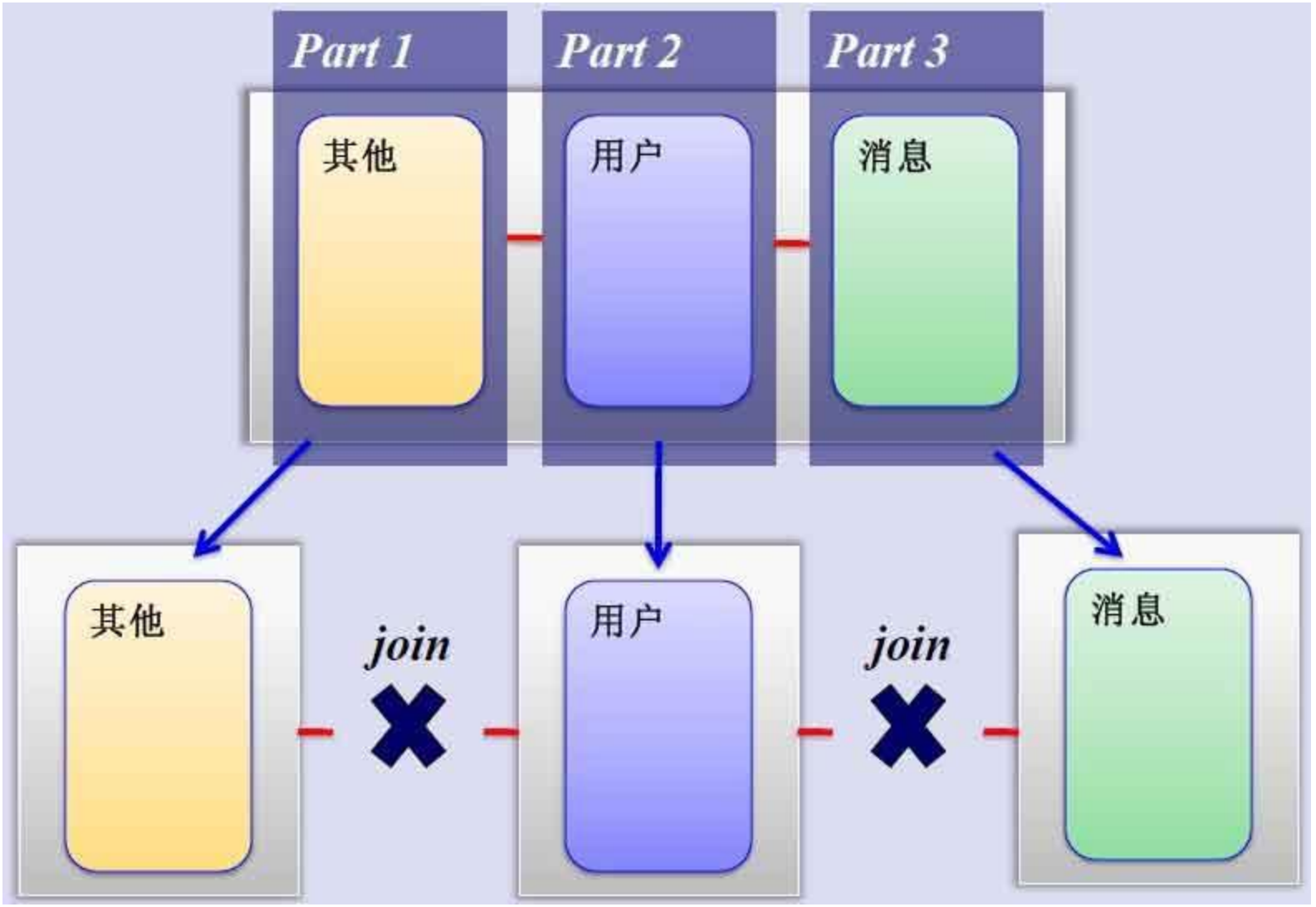
MySQL复制



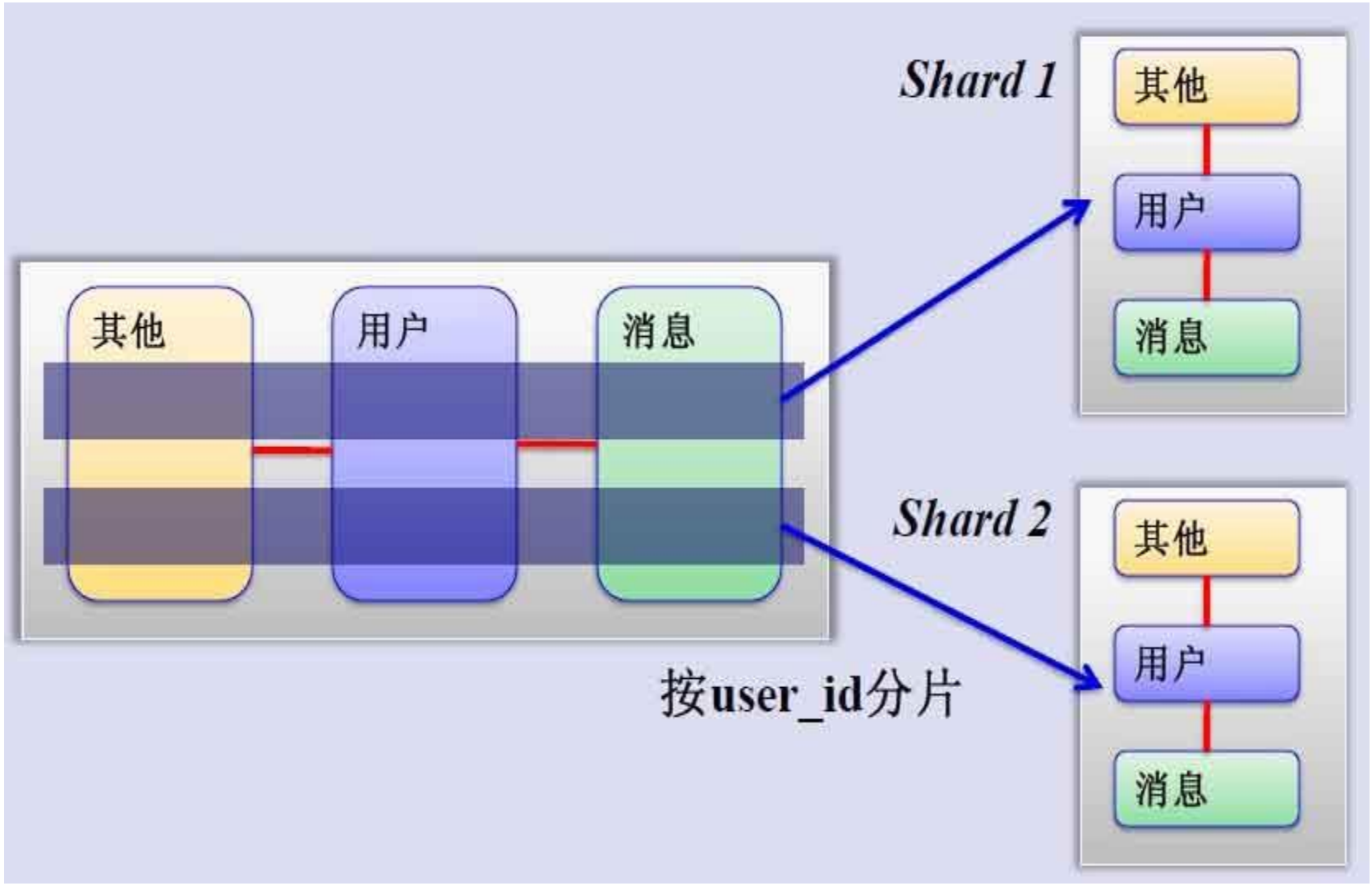
MySQL复制


UPDATE t1 ...
INSERT t2...
ALTER t1 ...
主库
binary log
100 ...
101 UPDATE t1...
102 INSERT t2...
103 ALTER t1...
104 ...
tcp 长连接
relay log
100 ...
101 UPDATE t1...
102 INSERT t2...
103 ALTER t1...
104 ...
I/O thread
SQL thread
从库
UPDATE t1 ...
INSERT t2...
ALTER t1 ...

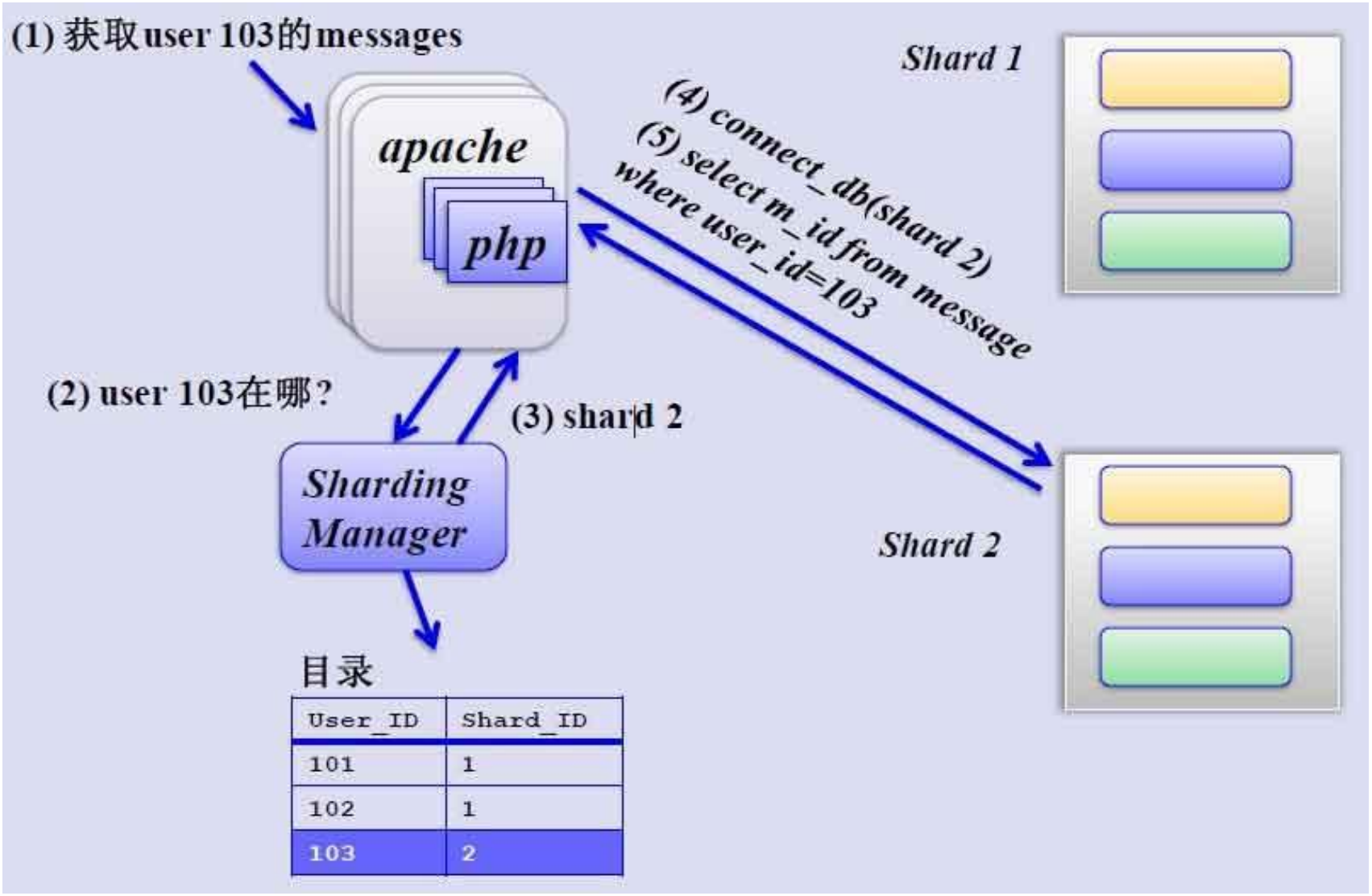
MySQL垂直分区



MySQL水平分片（Sharding）



对应shard中查询相关数据



MySQL复制

- 主从复制线程：
  - 主节点：

    dump Thread：为每个Slave的I/O Thread启动一个dump线程，用于向其发送binary log events
  - 从节点：

    I/O Thread：向Master请求二进制日志事件，并保存于中继日志中

    SQL Thread：从中继日志中读取日志事件，在本地完成重放
- 跟复制功能相关的文件：
  - master.info：用于保存slave连接至master时的相关信息，例如账号、密码、服务器地址等
  - relay-log.info：保存在当前slave节点上已经复制的当前二进制日志和本地replay log日志的对应关系

MySQL复制

- 主从复制特点：
  - 异步复制
  - 主从数据不一致比较常见
- 复制架构：

  Master/Slave, Master/Master, 环状复制

  一主多从

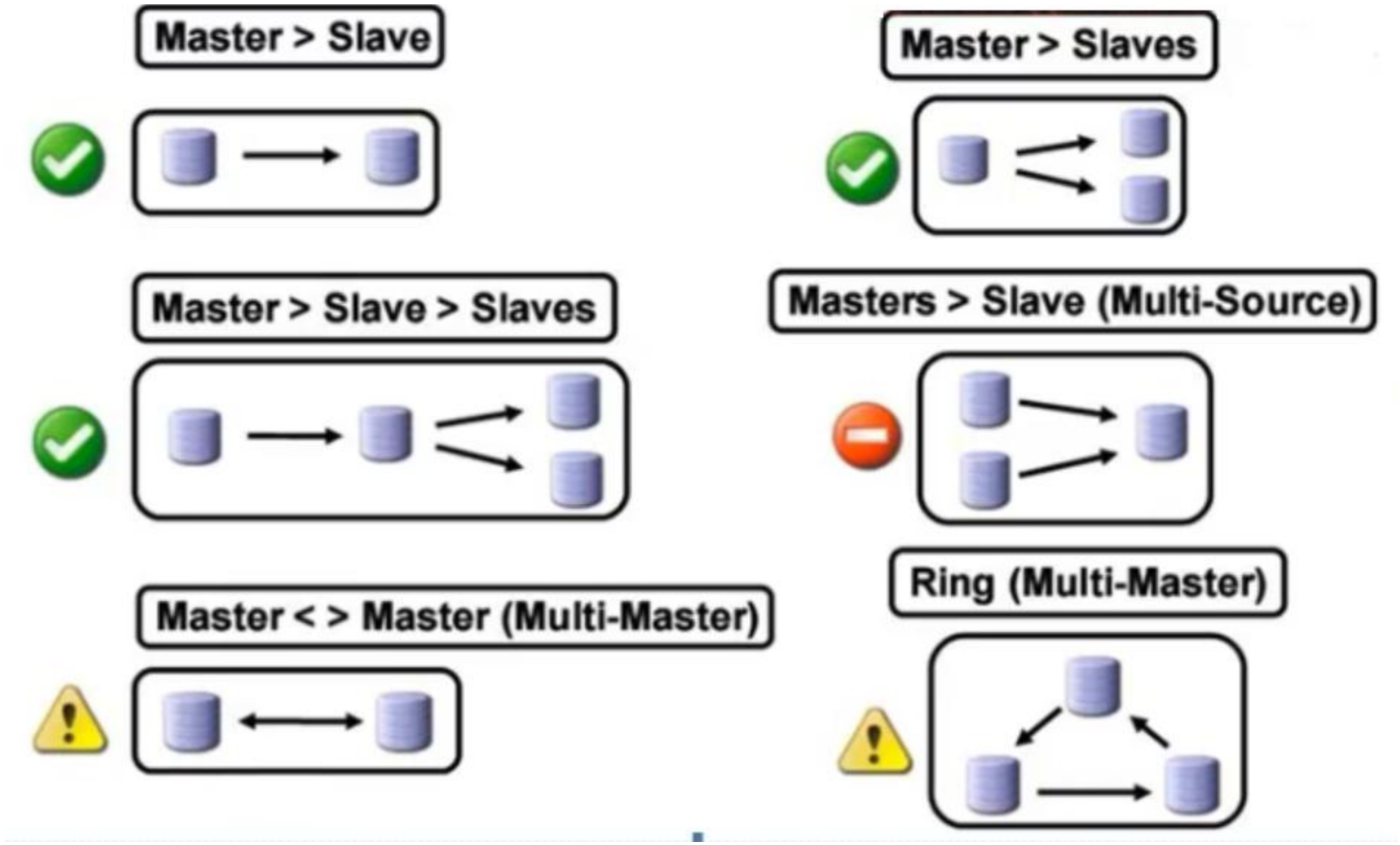
  从服务器还可以再有从服务器

  一从多主:适用于多个不同数据库
- 复制需要考虑二进制日志事件记录格式

  STATEMENT（5.0之前）、ROW（5.1之后，推荐）、MIXED
- 各种复制模型实战：

  主从、主主、半同步复制、复制过滤器

MySQL复制模型



MySQL复制

- 主从配置过程：

  参看：https://mariadb.com/kb/en/library/setting-up-replication/

  https://dev.mysql.com/doc/refman/5.5/en/replication-configuration.html

  主节点：

  (1) 启动二进制日志

  [mysqld]

  log_bin=mysql-bin

  (2) 为当前节点设置一个全局惟的ID号

  [mysqld]

  server_id=#

  (3) 创建有复制权限的用户账号

  GRANT REPLCATION SLAVE ON *.* TO 'repluser'@'HOST' IDENTIFIED BY 'replpass';

MySQL复制

- 从节点配置：
- (1) 启动中继日志

[mysqld]
server_id=# 为当前节点设置一个全局惟的ID号
relay_log=relay-log relay log的文件路径，默认值hostname-relay-bin
relay_log_index=relay-log.index 默认值*hostname*-relay-bin.index

- (2) 使用有复制权限的用户账号连接至主服务器，并启动复制线程

```
mysql> CHANGE MASTER TO MASTER_HOST='host', MASTER_USER='repluser',
  MASTER_PASSWORD='replpass', MASTER_LOG_FILE='mysql-bin.xxxxx',
  MASTER_LOG_POS=#;
mysql> START SLAVE [IO_THREAD|SQL_THREAD];
```

MySQL复制

- 如果主节点已经运行了一段时间，且有大量数据时，如何配置并启动slave节点
  - 通过备份恢复数据至从服务器
  - 复制起始位置为备份时，二进制日志文件及其POS
- 如果要启用级联复制,需要在从服务器启用以下配置

```
[mysqld]
log_bin
log_slave_updates
```

MySQL复制

- 复制架构中应该注意的问题：
- 1、限制从服务器为只读
  - 在从服务器上设置read_only=ON
    注意：此限制对拥有SUPER权限的用户均无效
  - 阻止所有用户，包括主服务器复制的更新

```
mysql> FLUSH TABLES WITH READ LOCK;
```

MySQL复制

- 2、如何保证主从复制的事务安全

  参看https://mariadb.com/kb/en/library/server-system-variables/

  - 在master节点启用参数：

    sync_binlog=1 每次写后立即同步二进制日志到磁盘，性能差

    如果用到的为InnoDB存储引擎：

    innodb_flush_logs_at_trx_commit=1

    每次事务提交立即同步日志写磁盘

    innodb_support_xa=ON 默认值，分布式事务MariaDB10.3.0废除

    sync_master_info=# 多少次事件后master.info同步到磁盘

  - 在slave节点启用服务器选项：

    --skip_slave_start=ON 不自动启动slave

  - 在slave节点启用参数：

    sync_relay_log=# #次写后同步relay log到磁盘

    sync_relay_log_info=#多个次事务后同步relay-log.info到磁盘

主主复制

- 主主复制：互为主从
  - 容易产生的问题：数据不一致；因此慎用
  - 考虑要点：自动增长id

    配置一个节点使用奇数id

    auto_increment_offset=1　开始点

    auto_increment_increment=2　增长幅度

    另一个节点使用偶数id

    auto_increment_offset=2

    auto_increment_increment=2

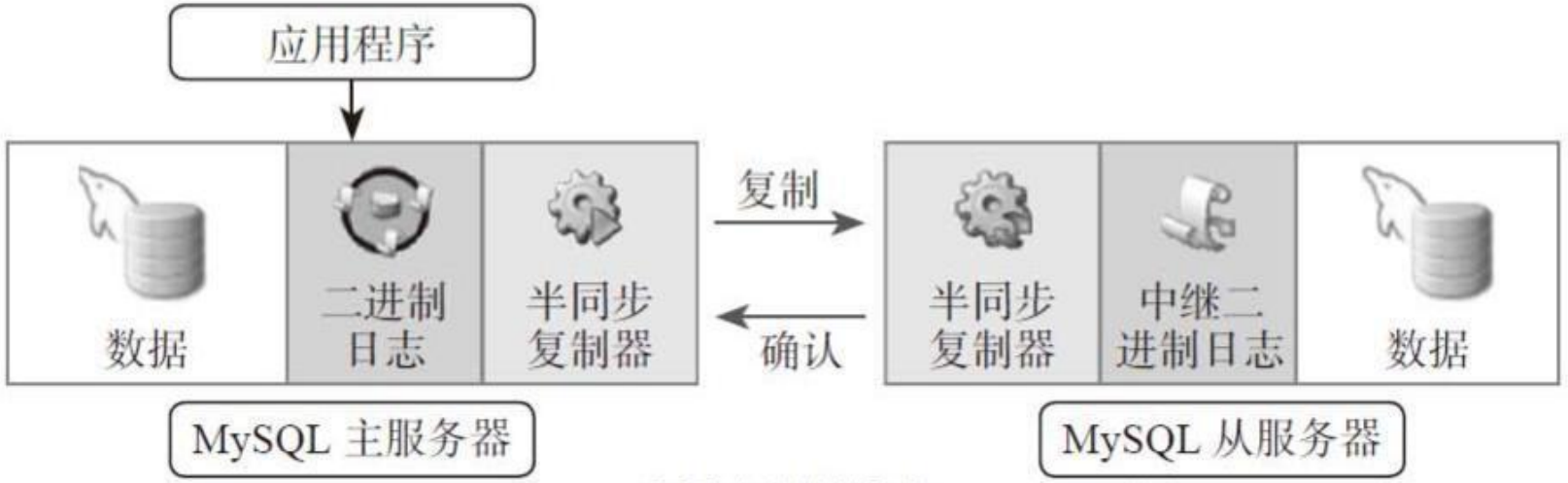
主主复制

- 主主复制的配置步骤：

(1) 各节点使用一个惟一server_id
(2) 都启动binary log和relay log
(3) 创建拥有复制权限的用户账号
(4) 定义自动增长id字段的数值范围各为奇偶
(5) 均把对方指定为主节点，并启动复制线程

半同步复制

- 默认情况下，MySQL的复制功能是异步的，异步复制可以提供最佳的性能，主库把binlog日志发送给从库即结束，并不验证从库是否接收完毕。这意味着当主服务器或从服务器端发生故障时，有可能从服务器没有接收到主服务器发送过来的binlog日志，这就会造成主服务器和从服务器的数据不一致，甚至在恢复时造成数据的丢失



半同步复制模式

半同步复制

- 半同步复制实现：
  - 主服务器配置:

```
mysql> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
mysql> SET GLOBAL VARIABLES rpl_semi_sync_master_enabled=1;
mysql> SHOW GLOBAL VARIABLES LIKE '%semi%';
mysql> SHOW GLOBAL STATUS LIKE '%semi%  ';
```

  - 从服务器配置:

```
mysql> INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
mysql> SET GLOBAL VARIABLES rpl_semi_sync_slave_enabled=1;
```

读写分离

- 读写分离应用：
  - mysql-proxy：Oracle
    https://downloads.mysql.com/archives/proxy/
  - Atlas：Qihoo
    https://github.com/Qihoo360/Atlas/blob/master/README_ZH.md
  - dbproxy：美团
    https://github.com/Meituan-Dianping/DBProxy
  - Amoeba：
    https://sourceforge.net/projects/amoeba/

关于马哥教育

马哥教育
IT人的高薪职业学院

- 博客：http://mageedu.blog.51cto.com
- 主页：http://www.magedu.com
- QQ：1661815153, 113228115
- QQ群：203585050, 279599283

祝大家学业有成

谢　谢