MYSQL数据库

讲师：王晓春

本章内容

- 关系型数据库基础
- 安装MySQL
- 管理数据库和表
- 用户和权限管理
- 函数，存储过程和触发器
- MySQL架构
- 存储引擎
- 服务器选项，系统和状态变量
- 优化查询和索引管理
- 锁和事务管理
- 日志管理
- 备份还原
- 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

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

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

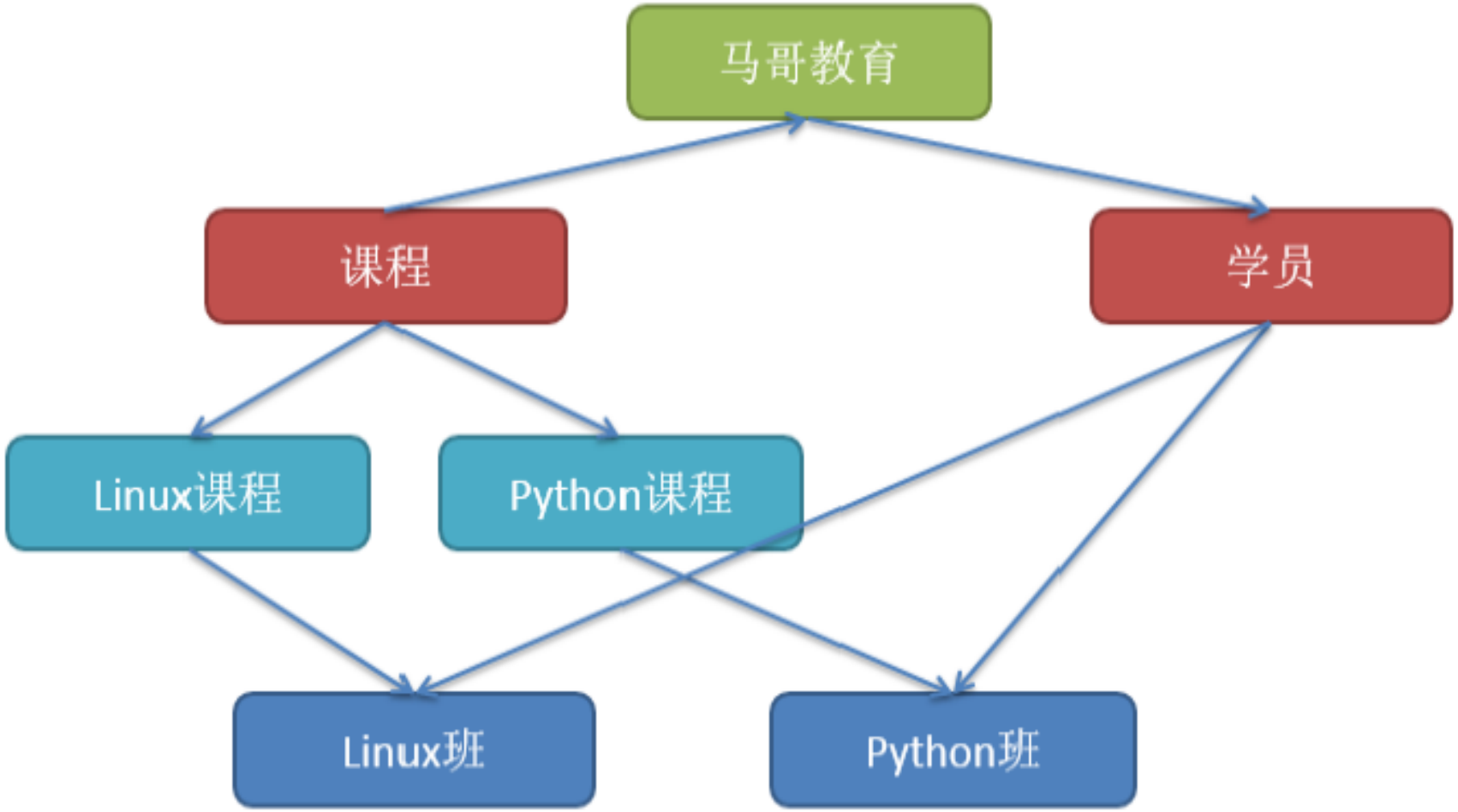
- 数据定义
- 数据处理
- 数据安全
- 数据备份

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

网状数据库

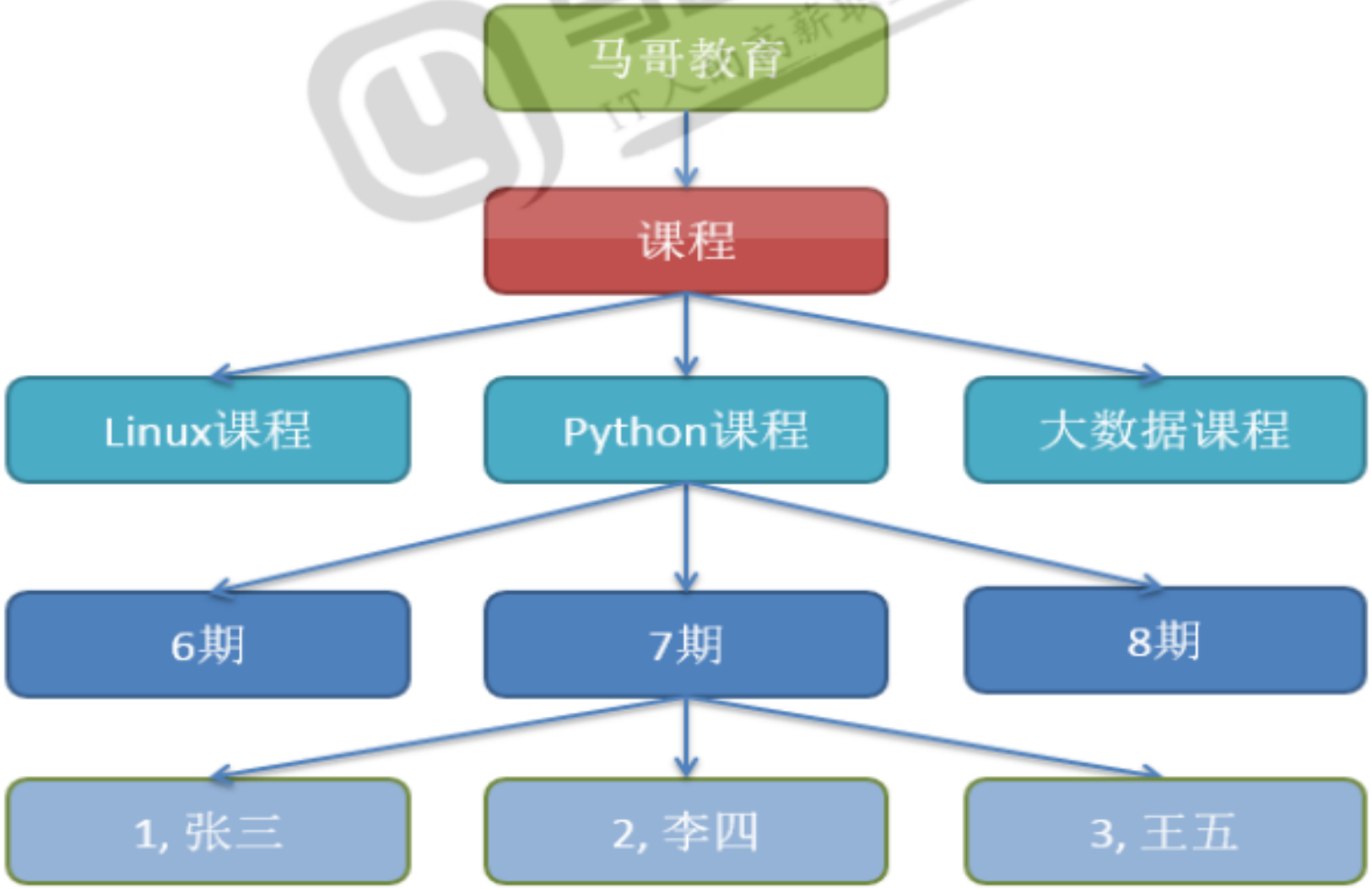
- 最早出现的是网状DBMS，1964年通用电气公司的Charles Bachman成功地开发出世界上第一个网状IDS，也是第一个数据库管理系统，IDS 具有数据模式和日志的特征，只能在GE主机运行



层次数据库

层次数据库

以树型结构表示实体及其之间的联系。关系只支持一对多。代表数据库IBM IMS。



关系型数据库

- 关系 ：关系就是二维表，其中：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
- 数据库排名：
  - https://db-engines.com/en/ranking

数据库排名

343 systems in ranking, June 2018

Rank Jun 2018	Rank May 2018	Rank Jun 2017	DBMS	Database Model	Score Jun 2018	Score May 2018	Score Jun 2017
1.	1.	1.	Oracle	Relational DBMS	1311.25	+20.84	-40.51
2.	2.	2.	MySQL	Relational DBMS	1233.69	+10.35	-111.62
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1087.73	+1.89	-111.23
4.	4.	4.	PostgreSQL	Relational DBMS	410.67	+9.77	+42.13
5.	5.	5.	MongoDB	Document store	343.79	+1.67	+8.79
6.	6.	6.	DB2	Relational DBMS	185.64	+0.03	-1.86
7.	7.	↑ 9.	Redis	Key-value store	136.30	+0.95	+17.42
8.	↑ 9.	↑ 11.	Elasticsearch	Search engine	131.04	+0.60	+19.48
9.	↓ 8.	↓ 7.	Microsoft Access	Relational DBMS	130.99	-2.12	+4.44
10.	10.	↓ 8.	Cassandra	Wide column store	119.21	+1.38	-4.91
11.	11.	↓ 10.	SQLite	Relational DBMS	114.26	-1.19	-2.44
12.	12.	12.	Teradata	Relational DBMS	75.77	+1.36	-1.55
13.	↑ 14.	↑ 18.	MariaDB	Relational DBMS	65.85	+0.85	+12.95
14.	↓ 13.	↑ 16.	Splunk	Search engine	65.78	+0.68	+8.26
15.	15.	↓ 14.	Solr	Search engine	62.06	+0.55	-1.55
16.	16.	↓ 13.	SAP Adaptive Server	Relational DBMS	61.49	-0.02	-6.04
17.	17.	↓ 15.	HBase	Wide column store	59.70	-0.25	-2.17
18.	18.	↑ 20.	Hive	Relational DBMS	57.33	+0.36	+12.95
19.	19.	↓ 17.	FileMaker	Relational DBMS	56.18	+1.51	-0.90
20.	20.	↓ 19.	SAP HANA	Relational DBMS	49.35	+0.97	+1.85

联系类型

- 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
- 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)

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

- 数据库规范化，又称数据库或资料库的正规化、标准化，是数据库设计中的一系列原理和技术，以减少数据库中数据冗余，增进数据的一致性。关系模型的发明者埃德加·科德最早提出这一概念，并于1970年代初定义了第一范式、第二范式和第三范式的概念
- 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，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，确保每一列的原子性。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，通常为表加上每行的唯一标识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

基本概念

- 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

数据模型

- 数据抽象：
  - 物理层：数据存储格式，即RDBMS在磁盘上如何组织文件
  - 逻辑层：DBA角度，描述存储什么数据，以及数据间存在什么样的关系
  - 视图层：用户角度，描述DB中的部分数据
- 关系模型的分类：
  - 关系模型
  - 基于对象的关系模型
  - 半结构化的关系模型：XML数据

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系列

- 官方网址：
  - https://www.mysql.com/
  - http://mariadb.org/
  - https://www.percona.com
- 官方文档
  - https://dev.mysql.com/doc/
  - https://mariadb.com/kb/en/
  - https://www.percona.com/software/mysql-database/percona-server
- 版本演变：
  - MySQL：5.1 --> 5.5 --> 5.6 --> 5.7 -->8.0
  - 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/

  国内镜像：https://mirrors.tuna.tsinghua.edu.cn/mariadb/yum/

  https://mirrors.tuna.tsinghua.edu.cn/mysql/yum/

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语句， 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客户端常用选项：

-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

  每个命令末尾都必须使用命令结束符号，默认为分号

  示例：SELECT VERSION();

mysqladmin

- mysqladmin –help
- 查看mysql服务是否正常，如果正常提示mysqld is alive
  mysqladmin -uroot -pcentos ping
- 关闭mysql服务，但mysqladmin命令无法开启
  mysqladmin –uroot –pcentos shutdown
- 创建数据库testdb
  mysqladmin -uroot –pcentos create testdb
- 删除数据库testdb
  mysqladmin -uroot -pcentos drop testdb
- 修改root密码
  mysqladmin –uroot –pcentos password ‘magedu’
- 日志滚动,生成新文件/var/lib/mysql/ mariadb-bin.00000N
  mysqladmin -uroot -pcentos flush-logs

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：
  - /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 –d /data/mysql mysql
- (2) 准备数据目录，建议使用逻辑卷
  - mkdir /data/mysql
  - chown mysql:mysql /data/mysql
- (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-large.cnf　/etc/mysql/my.cnf

  [mysqld]中添加三个选项：

  datadir = /data/mysql

  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=/data/mysql --user=mysql
```

- (6)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
```

- (7)PATH路径

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

- (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

- 做准备用户和数据目录

  useradd –r –s /sbin/nologin –d /data/mysql/ mysql

  mkdir /data/mysql

  chown mysql.mysql /data/mysql

  tar xvf mariadb-10.2.18.tar.gz

- cmake 编译安装

  cmake的重要特性之一是其独立于源码(out-of-source)的编译功能，即编译工作可以在另一个指定的目录中而非源码目录中进行，这可以保证源码目录不受任何一次编译的影响，因此在同一个源码树上可以进行多次不同的编译，如针对于不同平台编译

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

源码编译安装mariadb


```
cd mariadb-10.2.18/
cmake . \
-DCMAKE_INSTALL_PREFIX=/app/mysql \
-DMYSQL_DATADIR=/data/mysql/ \
-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=/data/mysql/mysql.sock \
-DDEFAULT_CHARSET=utf8 \
-DDEFAULT_COLLATION=utf8_general_ci
make && make install
```

- 提示：如果出错，执行rm -f CMakeCache.txt

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/mysql/ --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
- 存储函数：function
- 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-1986, SQL-1989, SQL-1992, SQL-1999, SQL-2003

  SQL-2008, SQL-2011

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)
- 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 数据定义语言
    CREATE，DROP，ALTER
  - DML: Data Manipulation Language 数据操纵语言
    INSERT，DELETE，UPDATE
  - DCL：Data Control Language 数据控制语言
    GRANT，REVOKE，COMMIT，ROLLBACK
  - DQL：Data Query Language 数据查询语言
    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'
```

- 修改数据库：

```
ALTER DATABASE DB_NAME character set  utf8;
```

- 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;

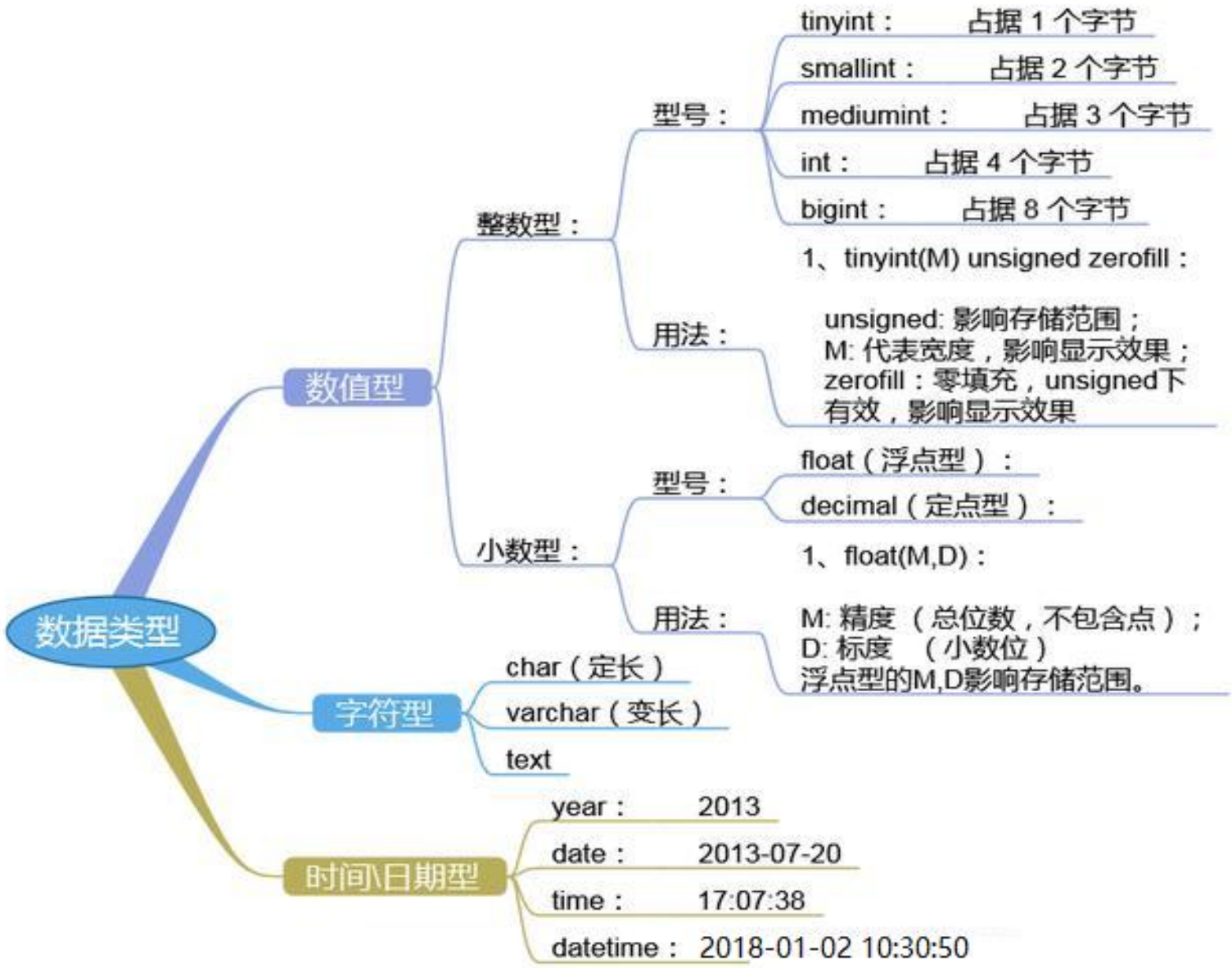
表操作

- 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
  SHOW COLUMNS FROM [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



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)

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;
- ALTER TABLE s1 character set utf8;
- ALTER TABLE s1 change name name varchar(20) character set utf8;
- 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 drop primary key ;
- ALTER TABLE students ADD UNIQUE KEY(name);
- ALTER TABLE students ADD INDEX(age);
- ALTER TABLE students drop primary key ;
- 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：=,<=>（相等或都为空）, <>, !=(非标准SQL), >, >=, <, <=
  BETWEEN min_num AND max_num
  IN (element1, element2, ...)
  IS NULL
  IS NOT NULL

SELECT

- DISTINCT 去除重复列

  SELECT DISTINCT gender FROM students;

- 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

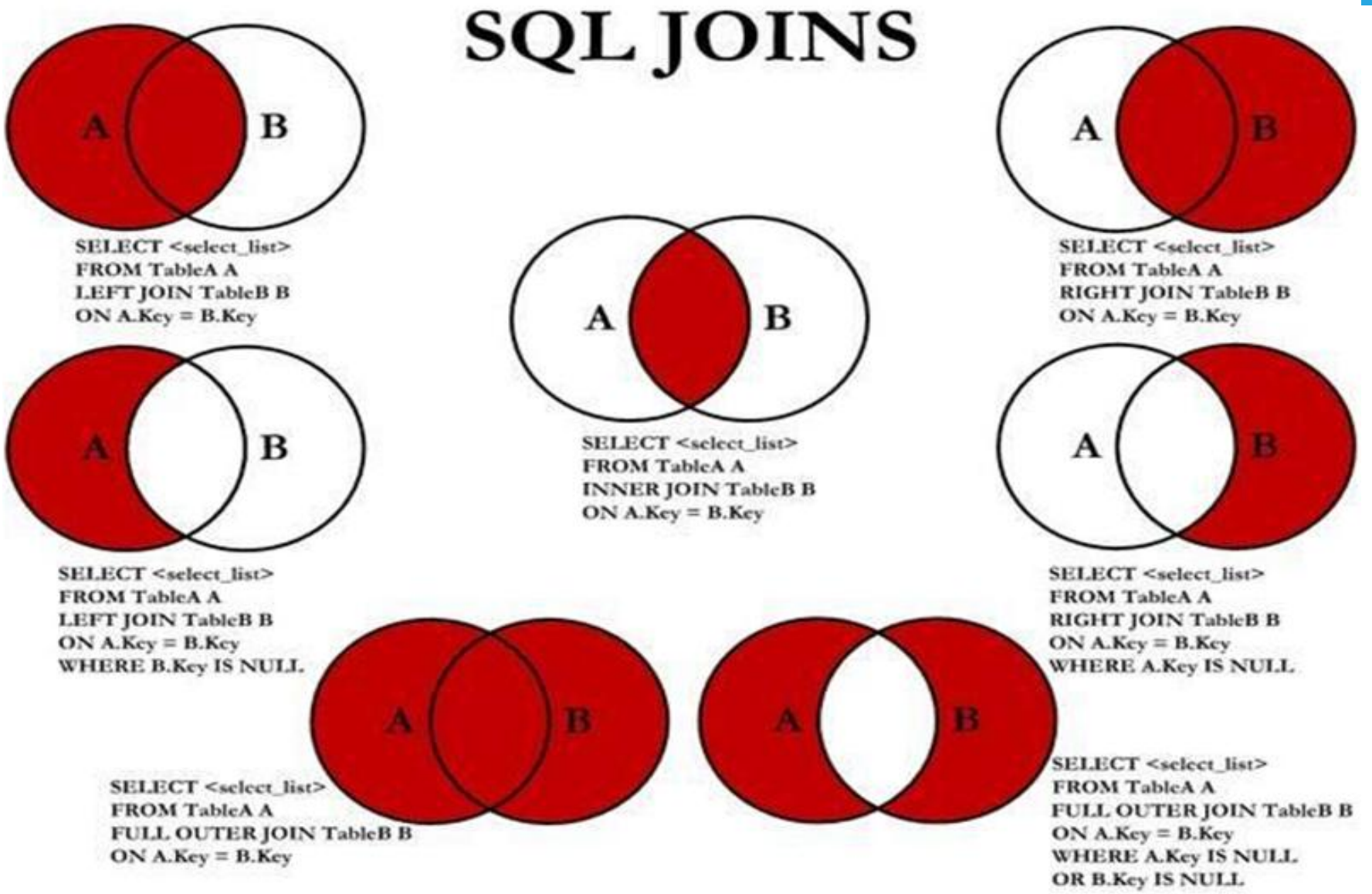
SELECT

- 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

示例

- DESC students;
- INSERT INTO students VALUES(1,'tom' , 'm'),(2,'alice','f');
- INSERT INTO students(id,name) VALUES(3,'jack'),(4,'allen');
- SELECT * FROM students WHERE id < 3;
- SELECT * FROM students WHERE gender='m';
- SELECT * FROM students WHERE gender IS NULL;
- SELECT * FROM students WHERE gender IS NOT NULL;
- SELECT * FROM students ORDER BY name DESC LIMIT 2;
- SELECT * FROM students ORDER BY name DESC LIMIT 1,2;
- SELECT * FROM students WHERE id >=2 and id <=4
- SELECT * FROM students WHERE BETWEEN 2 AND 4
- SELECT * FROM students WHERE name LIKE ‘t%’
- SELECT * FROM students WHERE name RLIKE '.*[lo].*';
- SELECT id stuid,name as stuname FROM students

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;

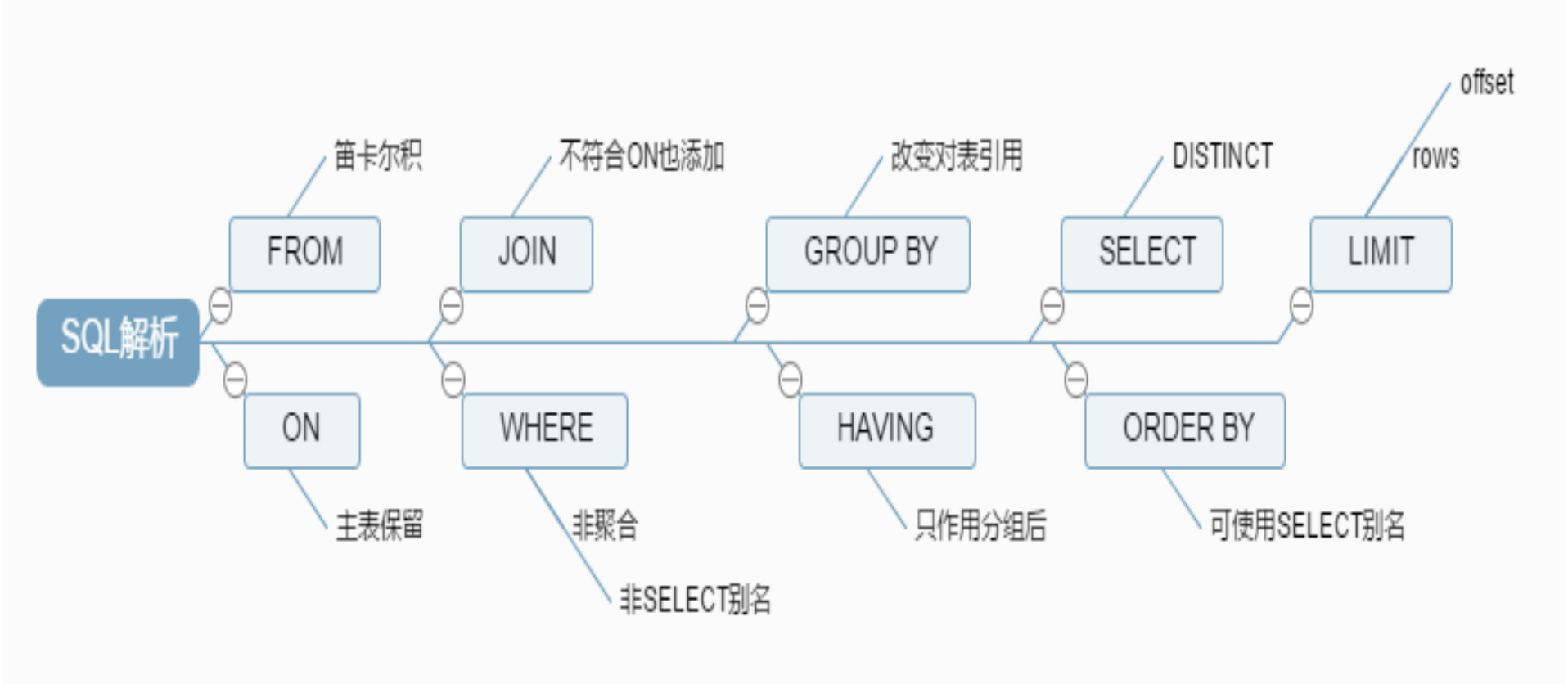
- 联合查询：UNION

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

SELECT语句处理的顺序

查询处理的顺序如下:

1、FROM
2、ON
3、JOIN
4、WHERE
5、GROUP BY
6、HAVING
7、SELECT
8、DISTINCT
9、ORDER BY
10、LIMIT



练习

- 导入hellodb.sql生成数据库
- (1) 在students表中，查询年龄大于25岁，且为男性的同学的名字和年龄
- (2) 以ClassID为分组依据，显示每组的平均年龄
- (3) 显示第2题中平均年龄大于30的分组及平均年龄
- (4) 显示以L开头的名字的同学的信息
- (5) 显示TeacherID非空的同学的相关信息
- (6) 以年龄排序后，显示年龄最大的前10位同学的信息
- (7) 查询年龄大于等于20岁，小于等于25岁的同学的信息

练习

导入hellodb.sql，实现下面要求

- 1、以ClassID分组，显示每班的同学的人数
- 2、以Gender分组，显示其年龄之和
- 3、以ClassID分组，显示其平均年龄大于25的班级
- 4、以Gender分组，显示各组中年龄大于25的学员的年龄之和
- 5、显示前5位同学的姓名、课程及成绩
- 6、显示其成绩高于80的同学的名称及课程
- 7、取每位同学各门课的平均成绩，显示成绩前三名的同学的姓名和平均成绩
- 8、显示每门课程课程名称及学习了这门课的同学的个数
- 9、显示其年龄大于平均年龄的同学的名字
- 10、显示其学习的课程为第1、2，4或第7门课的同学的名字
- 11、显示其成员数最少为3个的班级的同学中年龄大于同班同学平均年龄的同学
- 12、统计各班级中年龄大于全校同学平均年龄的同学

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

- 创建函数

  示例：无参UDF

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

- 查看函数列表：

  SHOW FUNCTION 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：有参数UDF

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

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
  - 存储过程一般可独立执行,而函数往往是作为其他SQL语句的一部分来使用

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

  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 selectById(IN uid SMALLINT UNSIGNED)
BEGIN
   SELECT * FROM students WHERE stuid = uid;
END//
delimiter ;

call selectById(2);
```

存储过程示例

- 示例

```
delimiter //
CREATE PROCEDURE dorepeat(n INT)
BEGIN
SET @i = 0;
SET @sum = 0;
REPEAT SET @sum = @sum+@i; SET @i = @i + 1;
UNTIL @i > n END REPEAT;
END//
delimiter ;
CALL dorepeat(100);
SELECT @sum;
```

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 uid SMALLINT UNSIGNED, OUT num
    SMALLINT UNSIGNED)
BEGIN
DELETE FROM students WHERE stuid >= uid;
SELECT row_count() into num;
END//
delimiter ;
call deleteById(2,@Line);
SELECT @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

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_id INT(11) NOT NULL AUTO_INCREMENT,
  stu_name VARCHAR(255) DEFAULT NULL,
  PRIMARY KEY (stu_id)
);
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： % _
      - 示例：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'@'172.16.0.%';

- 查看指定用户获得的授权

  Help SHOW GRANTS

  SHOW GRANTS FOR 'user'@'host';

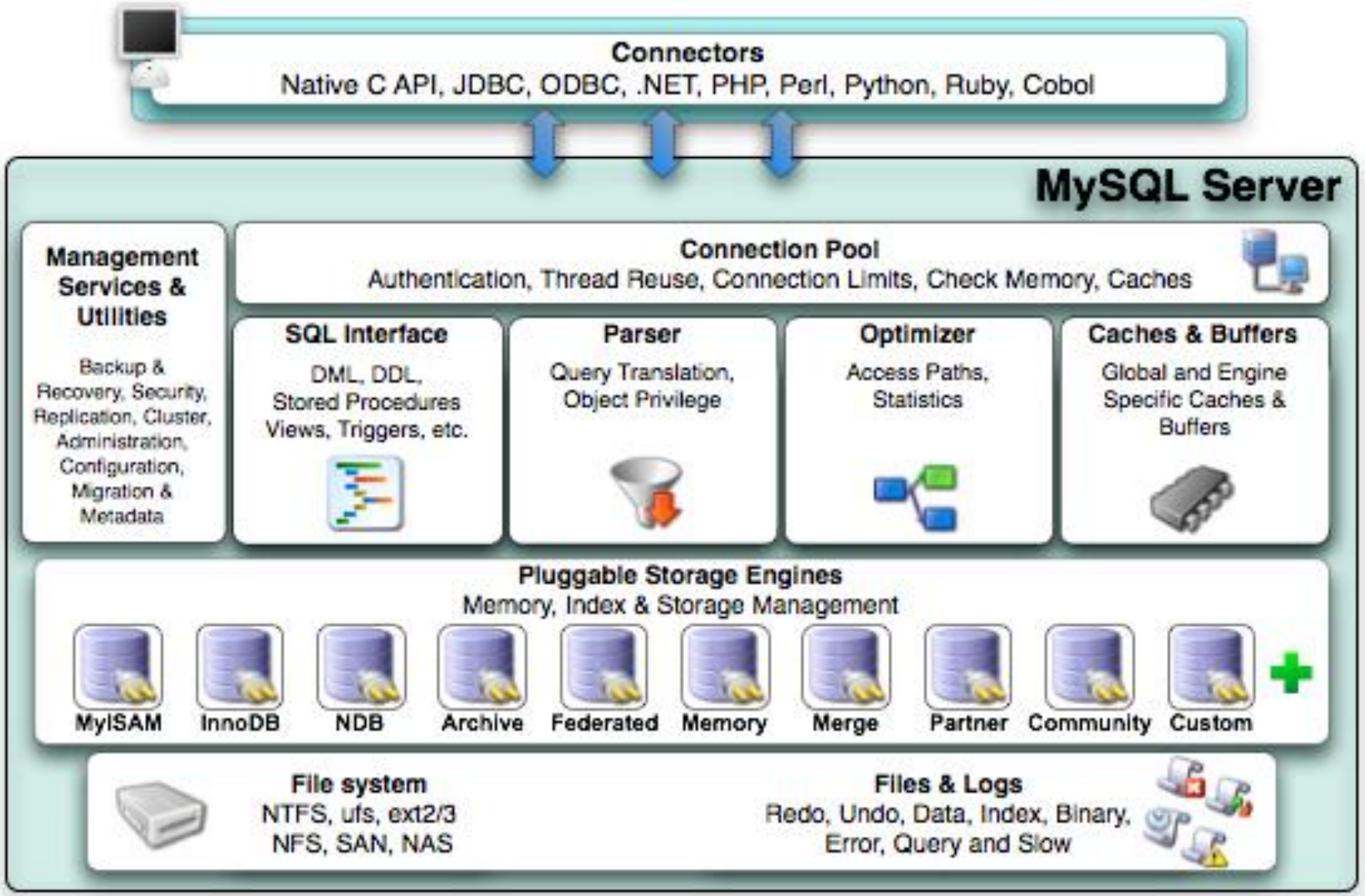
  SHOW GRANTS FOR CURRENT_USER[()];

- 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架构



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

- MyISAM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
  - 行级锁
  - 支持事务，适合处理大量短期事务
  - 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
    参看：https://mariadb.com/kb/en/library/xtradbinnodb-server-system-variables/#innodb_file_per_table
    ON (>= MariaDB 5.5)
    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数据库使用
- 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

其它存储引擎

- MariaDB支持的其它存储引擎：
  - OQGraph
  - SphinxSE
  - TokuDB
  - Cassandra
  - CONNECT
  - SQUENCE

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数据库

  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/server-option-variable-reference.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

服务器配置

- 获取mysqld的可用选项列表：

  mysqld --help --verbose

  mysqld --print-defaults 获取默认设置

- 设置服务器选项方法：
  - 在命令行中设置

    shell> ./mysqld_safe --skip-name-resolve=1
  - 在配置文件my.cnf中设置

    skip_name_resolve=1

服务器端设置

- 服务器系统变量：分全局和会话两种
- 获取系统变量

```
mysql> SHOW GLOBAL VARIABLES;
mysql> SHOW [SESSION] VARIABLES;
mysql> SELECT @@VARIABLES;
```

- 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;
```

服务器端设置

- 服务器状态变量：

  分全局和会话两种

- 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;

服务器变量SQL_MODE

- SQL_MODE：对其设置可以完成一些约束检查的工作,可分别进行全局的设置或当前会话的设置，参看：https://mariadb.com/kb/en/library/sql-mode/
- 常见MODE:
  - NO_AUTO_CREATE_USER

    禁止GRANT创建密码为空的用户
  - NO_ZERO_DATE

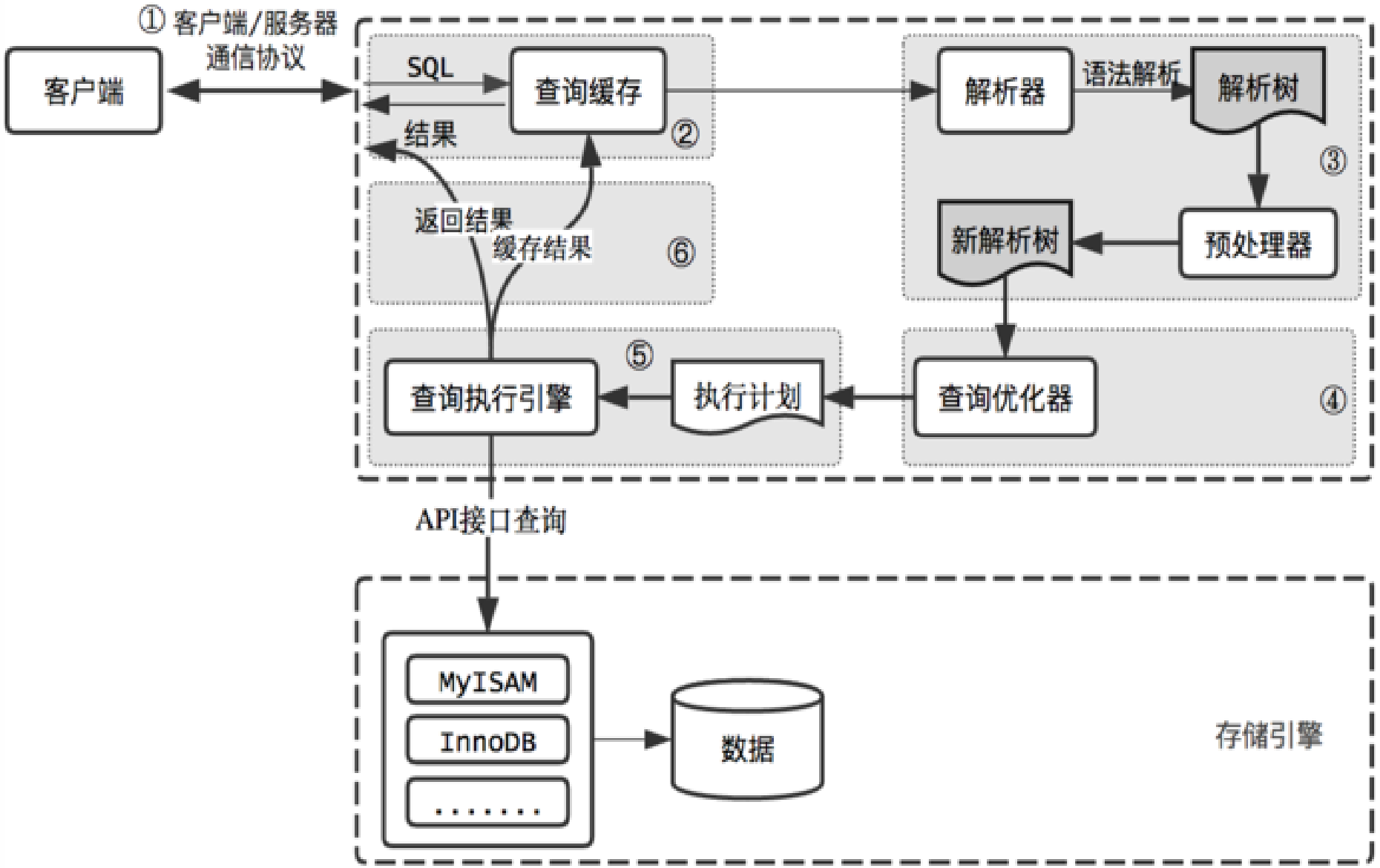
    在严格模式，不允许使用 ‘0000-00-00’ 的时间
  - ONLY_FULL_GROUP_BY

    对于GROUP BY聚合操作，如果在SELECT中的列，没有在GROUP BY中出现，那么将认为这个SQL是不合法的
  - NO_BACKSLASH_ESCAPES

    反斜杠“\”作为普通字符而非转义字符
  - PIPES_AS_CONCAT

    将"||"视为连接操作符而非“或运算符”

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

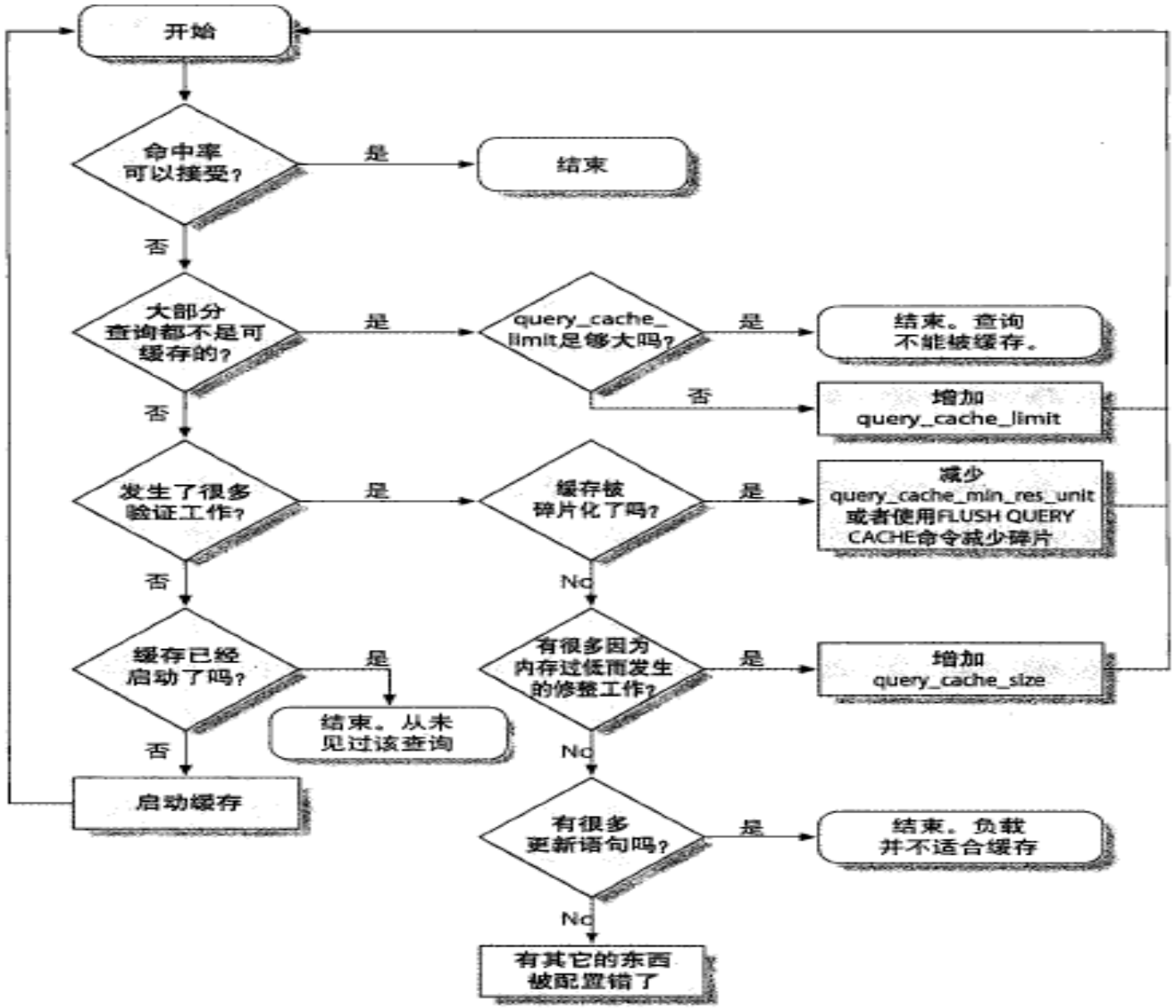
查询缓存

- 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_total_blocks：Query Cache 中总Block ，当Qcache_free_blocks相对此值较大时，可能用内存碎片，执行FLUSH QUERY CACHE清理碎片
  - 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：记录因为内存不足而被移除出查询缓存的查询数
  - 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

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，在MySQL又称为键key，索引通过存储引擎实现
- 优点：

  索引可以降低服务需要扫描的数据量，减少了IO次数

  索引可以帮助服务器避免排序和使用临时表

  索引可以帮助将随机I/O转为顺序I/O
- 缺点：

  占用额外空间，影响插入速度

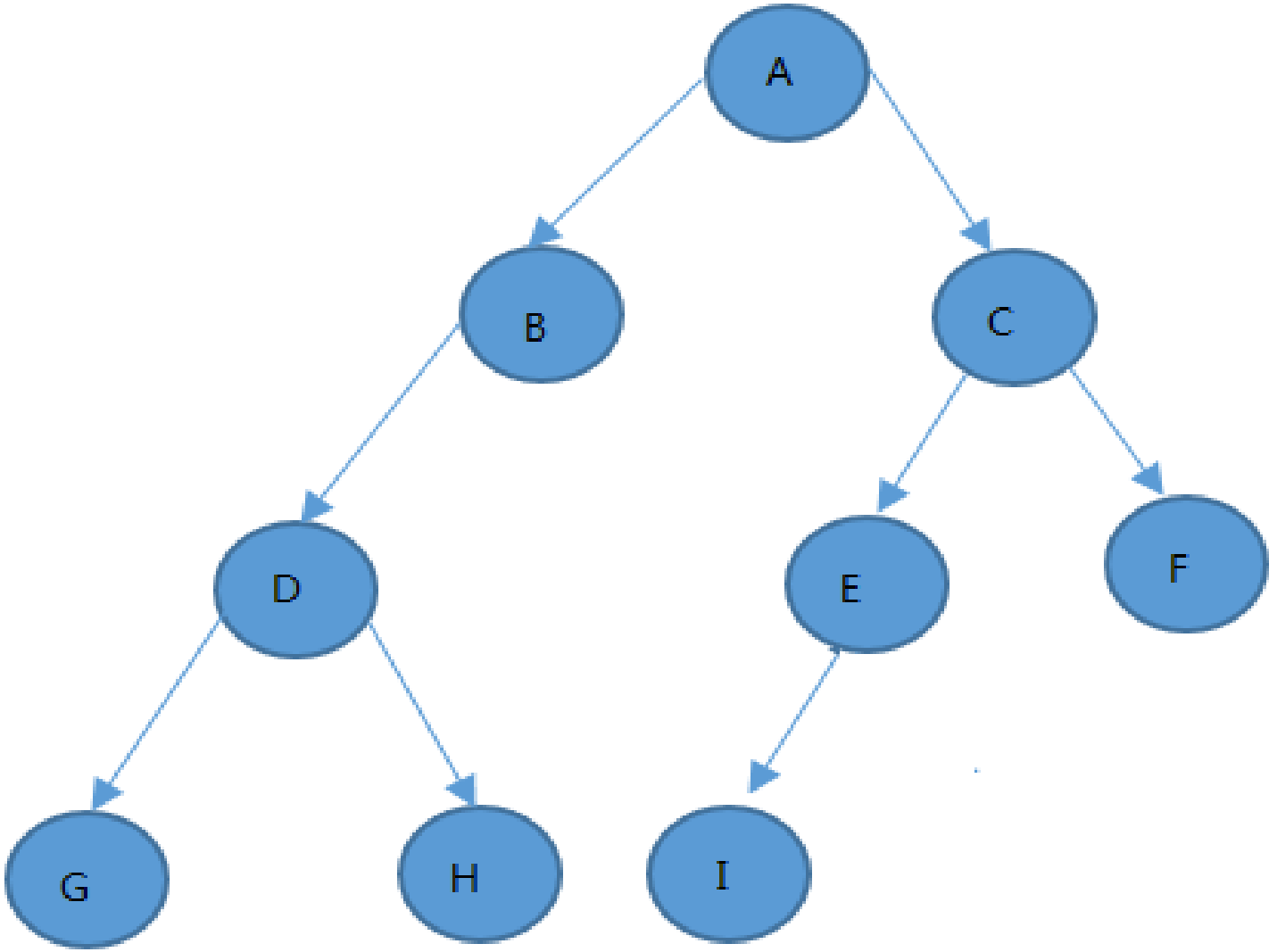
索引

- 索引类型：
  - B+ TREE、HASH、R TREE
  - 聚簇（集）索引、非聚簇索引：数据和索引是否存储在一起
  - 主键索引、二级（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

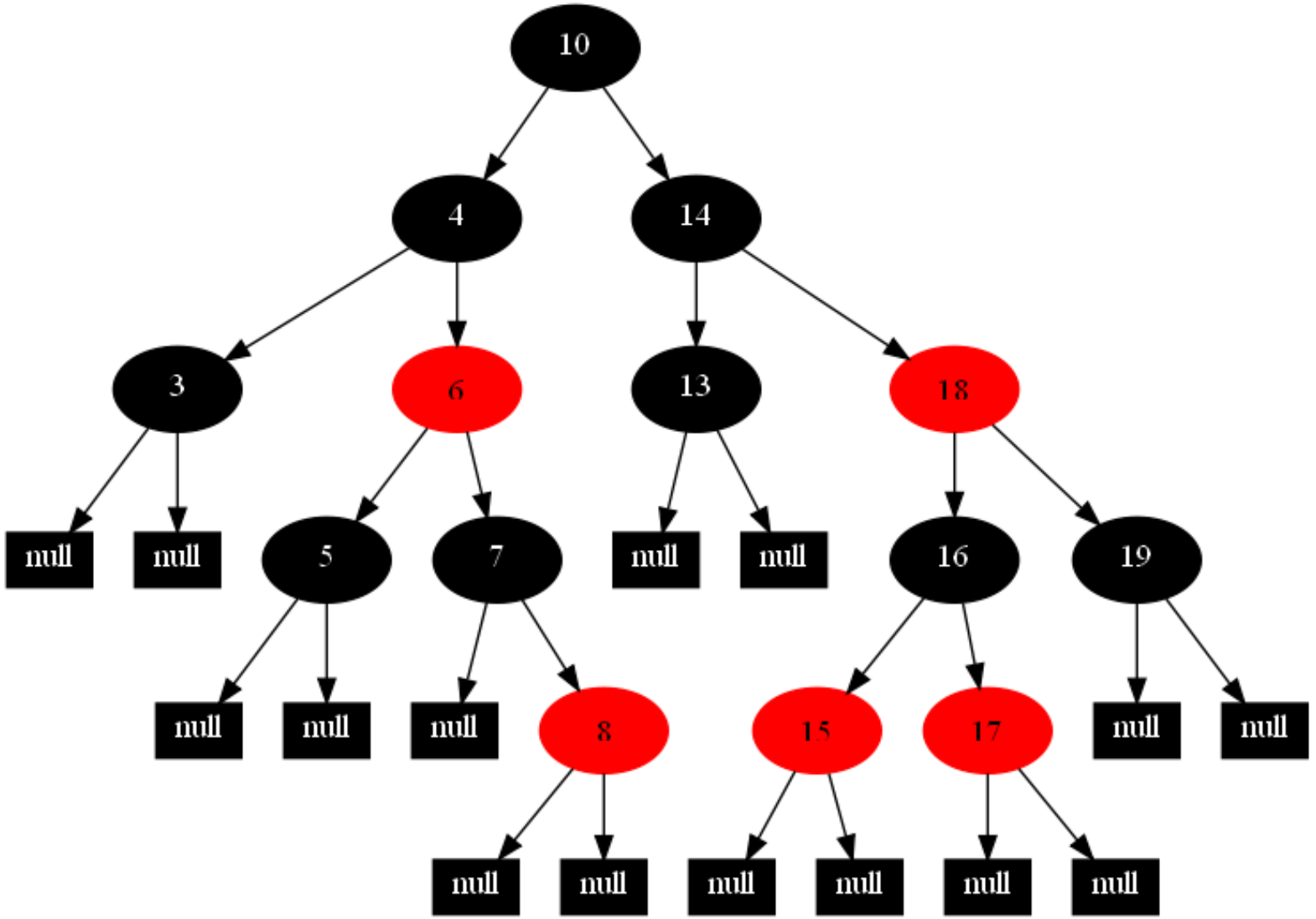
    左前缀索引：取前面的字符做索引

    覆盖索引：从索引中即可取出要查询的数据，性能高

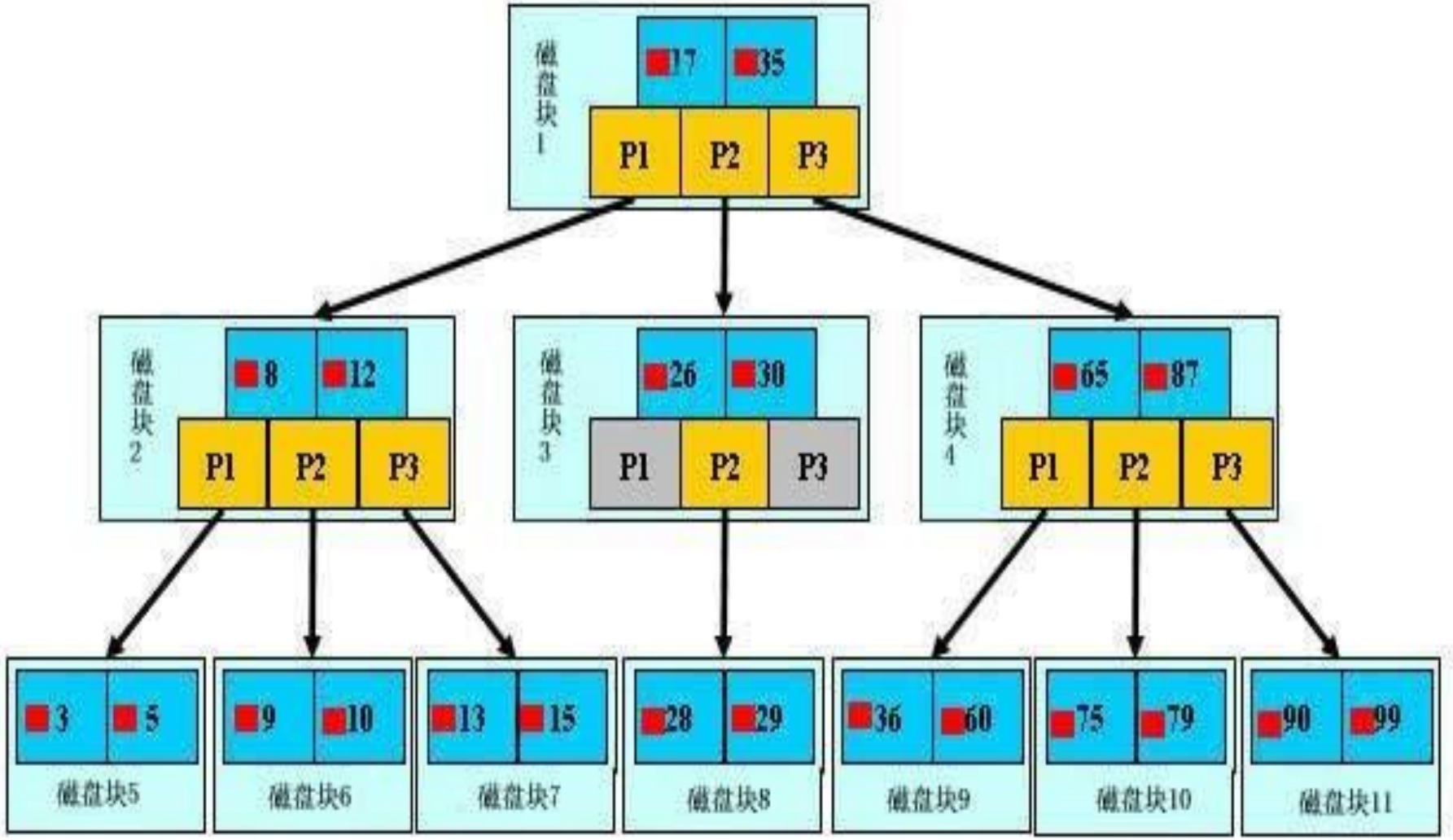
二叉树



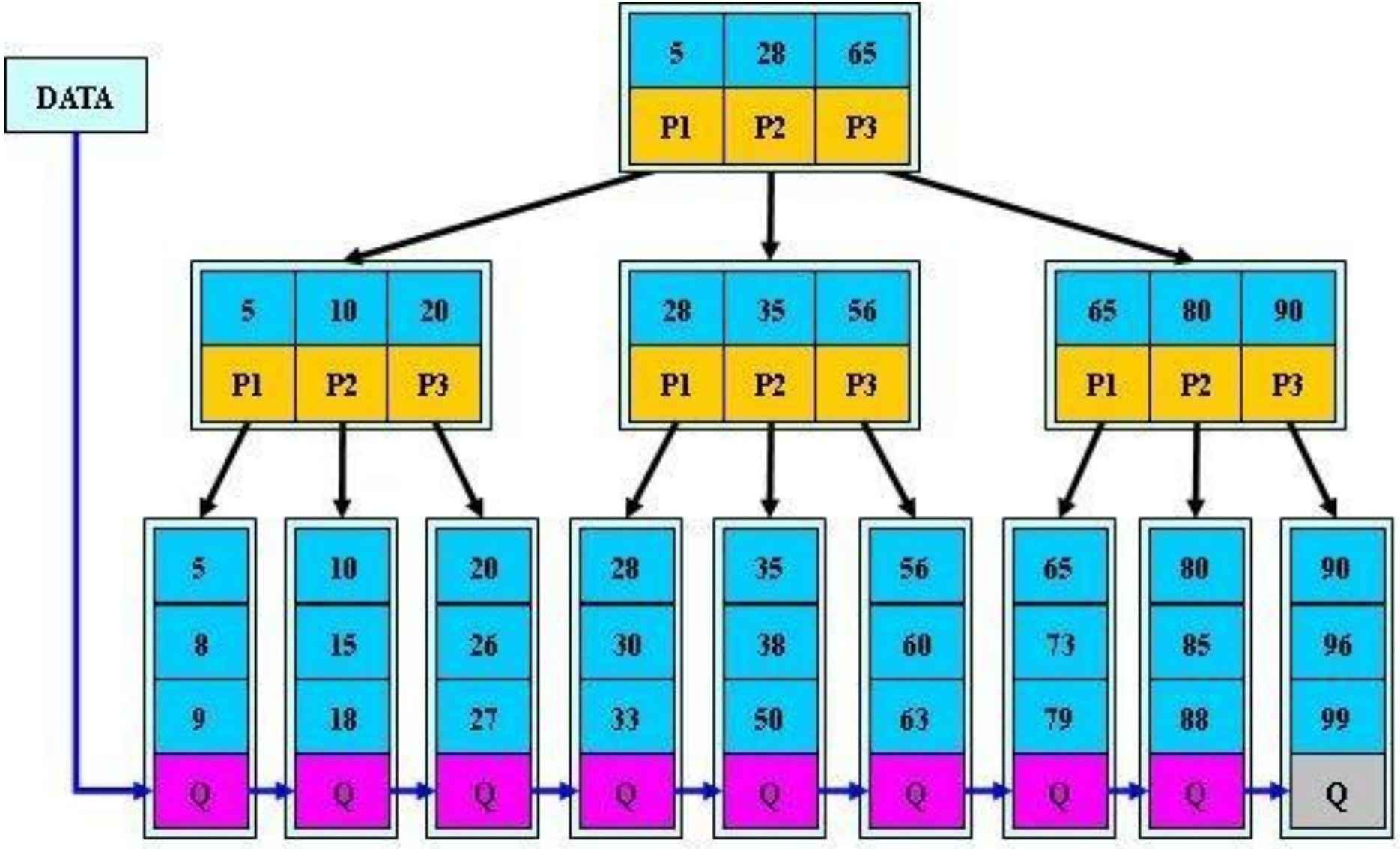
红黑树



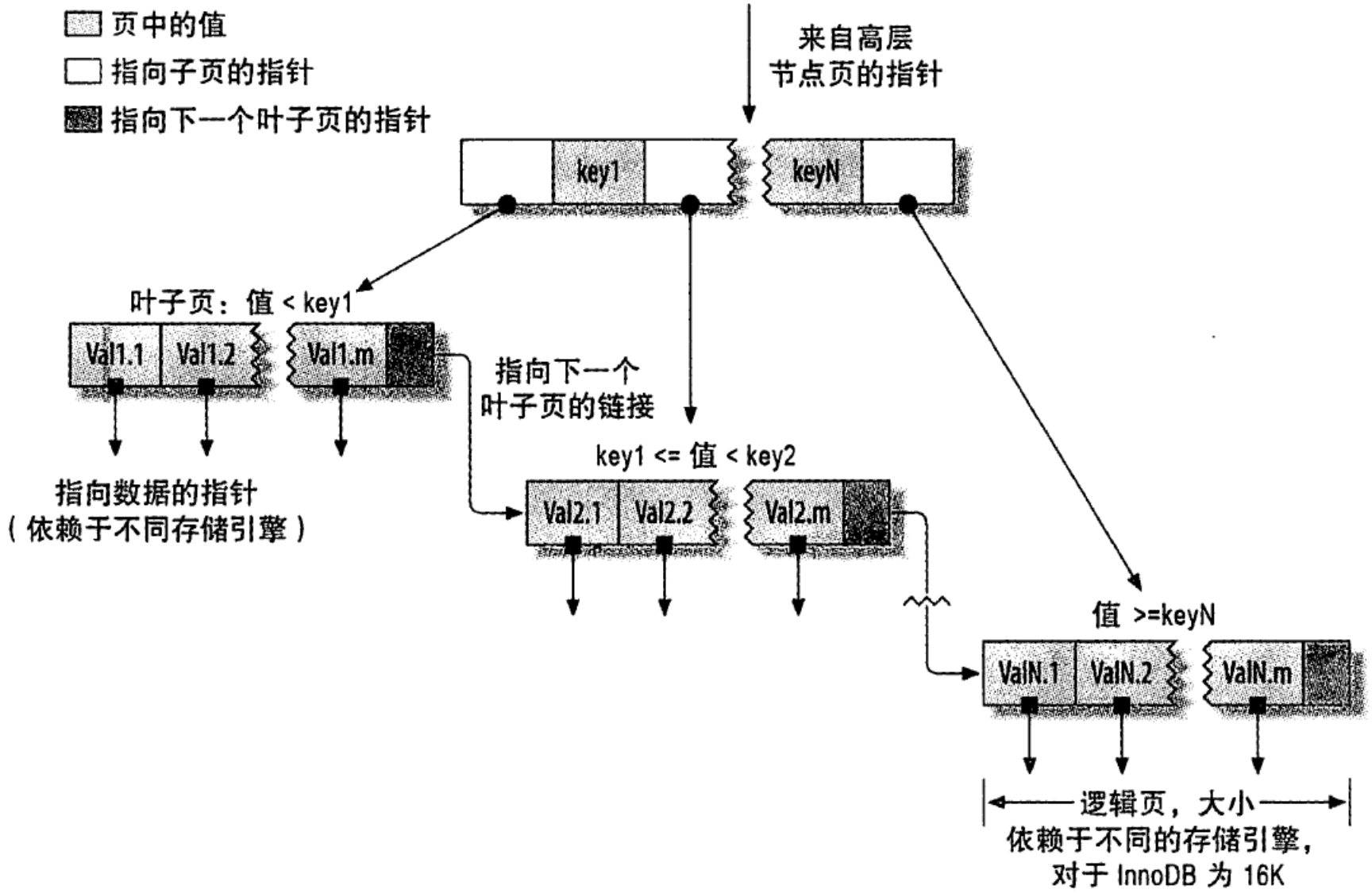
B-TREE索引


磁盘块1
17
35
P1
P2
P3
磁盘块2
8
12
P1
P2
P3
磁盘块3
26
30
P1
P2
P3
磁盘块4
65
87
P1
P2
P3
3
5
磁盘块5
9
10
磁盘块6
13
15
磁盘块7
28
29
磁盘块8
36
60
磁盘块9
75
79
磁盘块10
90
99
磁盘块11

B+TREE索引



B+TREE索引


页中的值
指向子页的指针
指向下一个叶子页的指针
来自高层
节点页的指针
key1
keyN
叶子页：值 < key1
Val1.1
Val1.2
Val1.m
指向下一个
叶子页的链接
key1 <= 值 < key2
Val2.1
Val2.2
Val2.m
指向数据的指针
（依赖于不同存储引擎）
值 >=keyN
ValN.1
ValN.2
ValN.m
逻辑页，大小
依赖于不同的存储引擎，
对于 InnoDB 为 16K

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
- 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

Hash索引

- Hash索引：基于哈希表实现，只有精确匹配索引中的所有列的查询才有效，索引自身只存储索引列对应的哈希值和数据指针，索引结构紧凑，查询性能好
- Memory存储引擎支持显式hash索引，InnoDB和MyISAM存储引擎不支持
- 适用场景：只支持等值比较查询，包括=, <=>, IN()
- 不适合使用hash索引的场景

  不适用于顺序查询：索引存储顺序的不是值的顺序

  不支持模糊匹配

  不支持范围查询

  不支持部分索引列匹配查找：如A，B列索引，只查询A列索引无效

索引

- 空间数据索引R-Tree（ Geospatial indexing ）

  MyISAM支持地理空间索引，可以使用任意维度组合查询，使用特有的函数访问，常用于做地理数据存储，使用不多

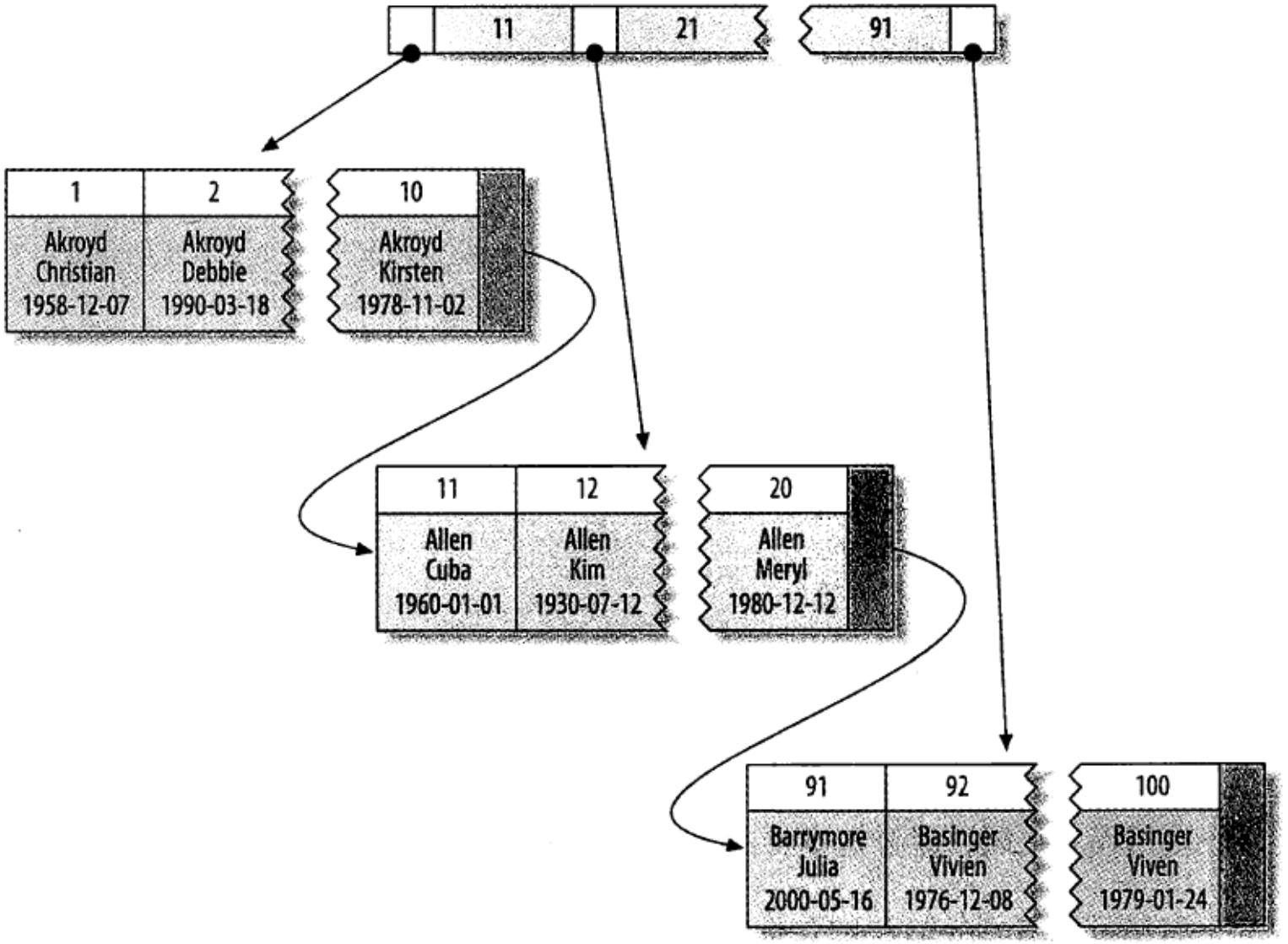
  InnoDB从MySQL5.7之后也开始支持

- 全文索引(FULLTEXT)

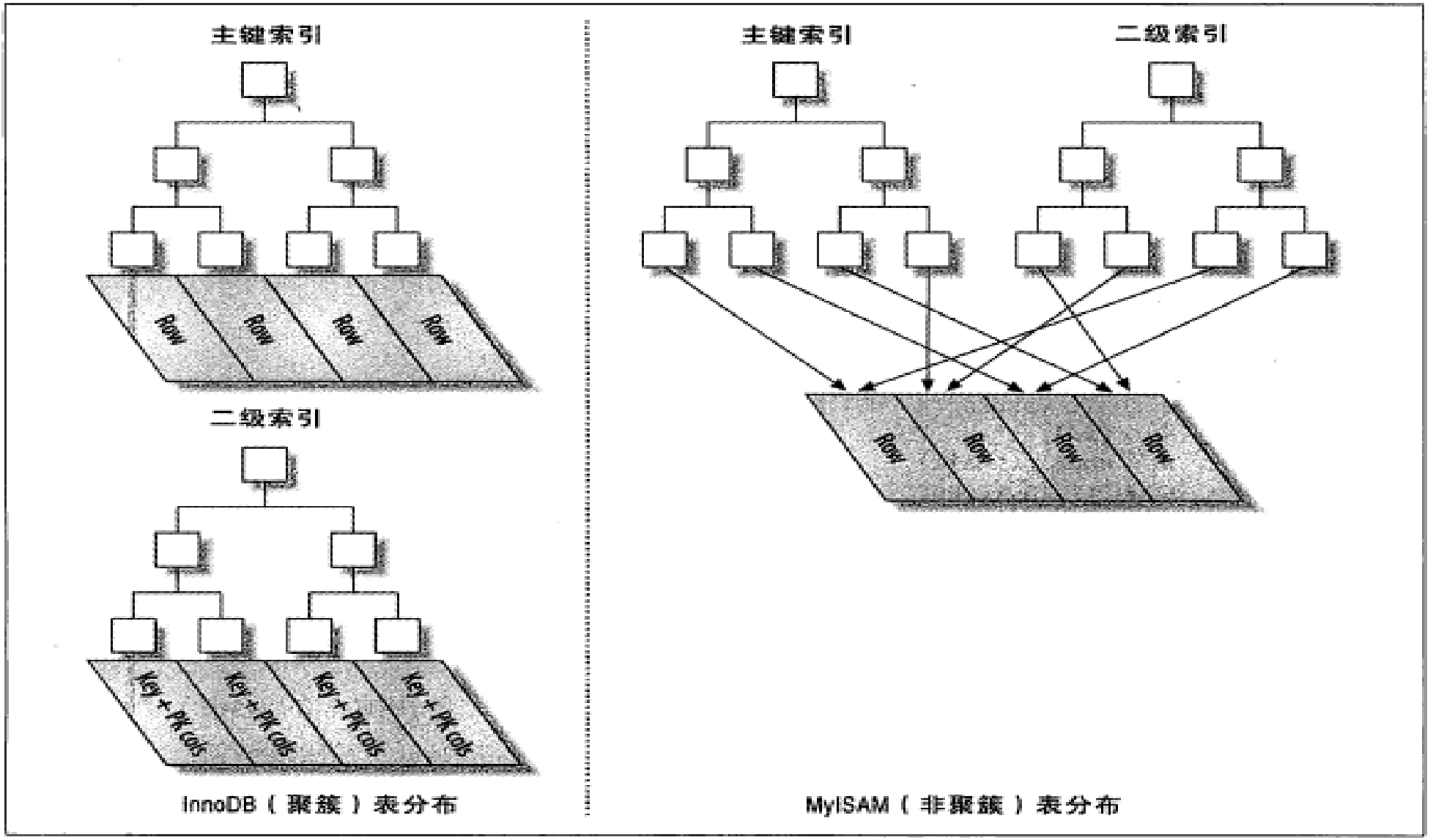
  在文本中查找关键词，而不是直接比较索引中的值，类似搜索引擎

  InnoDB从MySQL 5.6之后也开始支持

聚簇和非聚簇索引



聚簇和非聚簇索引，主键和二级索引



索引

- 冗余和重复索引：

  冗余索引：（A），（A，B）

  重复索引：已经有索引，再次建立索引

- 索引优化策略：
  - 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

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

SQL语句性能优化

- 查询时，能不要*就不用*，尽量写全字段名
- 大部分情况连接效率远大于子查询
- 多表连接时，尽量小表驱动大表，即小表 join 大表
- 在有大量记录的表分页时使用limit
- 对于经常使用的查询，可以开启缓存
- 多使用explain和profile分析查询语句
- 查看慢查询日志，找出执行时间长的sql语句优化

管理索引

- 创建索引：

```
CREATE [UNIQUE] INDEX index_name ON tbl_name (index_col_name[(length)],...);
ALTER TABLE tbl_name ADD INDEX  index_name(index_col_name);
help CREATE INDEX;
```

- 删除索引：

```
DROP INDEX index_name ON tbl_name;
ALTER TABLE tbl_name DROP INDEX  index_name(index_col_name);
```

- 查看索引：

```
SHOW INDEXES FROM [db_name.]tbl_name;
```

- 优化表空间：

```
OPTIMIZE TABLE tb_name;
```

- 查看索引的使用

```
SET GLOBAL userstat=1;
SHOW INDEX_STATISTICS;
```

EXPLAIN

- 通过EXPLAIN来分析索引的有效性
- EXPLAIN SELECT clause

  获取查询执行计划信息，用来查看查询优化器如何执行查询
- 输出信息说明：

  参考 https://dev.mysql.com/doc/refman/5.7/en/explain-output.html
- id: 当前查询语句中，每个SELECT语句的编号

  复杂类型的查询有三种：

  简单子查询

  用于FROM中的子查询

  联合查询：UNION

  注意：UNION查询的分析结果会出现一个额外匿名临时表

EXPLAIN

- select_type：
  简单查询为SIMPLE
  复杂查询：

SUBQUERY	简单子查询
PRIMARY	最外面的SELECT
DERIVED	用于FROM中的子查询
UNION	UNION语句的第一个之后的SELECT语句
UNION RESULT	匿名临时表

- table：SELECT语句关联到的表

EXPLAIN

- type：关联类型或访问类型，即MySQL决定的如何去查询表中的行的方式，以下顺序，性能从低到高
  - ALL: 全表扫描
  - index：根据索引的次序进行全表扫描；如果在Extra列出现“Using index”表示了使用覆盖索引，而非全表扫描
  - range：有范围限制的根据索引实现范围扫描；扫描位置始于索引中的某一点，结束于另一点
  - ref: 根据索引返回表中匹配某单个值的所有行
  - eq_ref：仅返回一个行，但与需要额外与某个参考值做比较
  - const, system: 直接返回单个行
- possible_keys：查询可能会用到的索引
- key: 查询中使用到的索引
- key_len: 在索引使用的字节数

EXPLAIN

- ref: 在利用key字段所表示的索引完成查询时所用的列或某常量值
- rows：MySQL估计为找所有的目标行而需要读取的行数
- Extra：额外信息

  Using index：MySQL将会使用覆盖索引，以避免访问表

  Using where：MySQL服务器将在存储引擎检索后，再进行一次过滤

  Using temporary：MySQL对结果排序时会使用临时表

  Using filesort：对结果使用一个外部索引排序

并发控制

- 锁粒度：
  - 表级锁
  - 行级锁
- 锁：
  - 读锁：共享锁，只读不可写（包括当前事务），多个读互不阻塞
  - 写锁：独占锁，排它锁，写锁会阻塞其它事务（不包括当前事务）的读和它锁
- 实现
  - 存储引擎：自行实现其锁策略和锁粒度
  - 服务器级：实现了锁，表级锁，用户可显式请求
- 分类：
  - 隐式锁：由存储引擎自动施加锁
  - 显式锁：用户手动请求

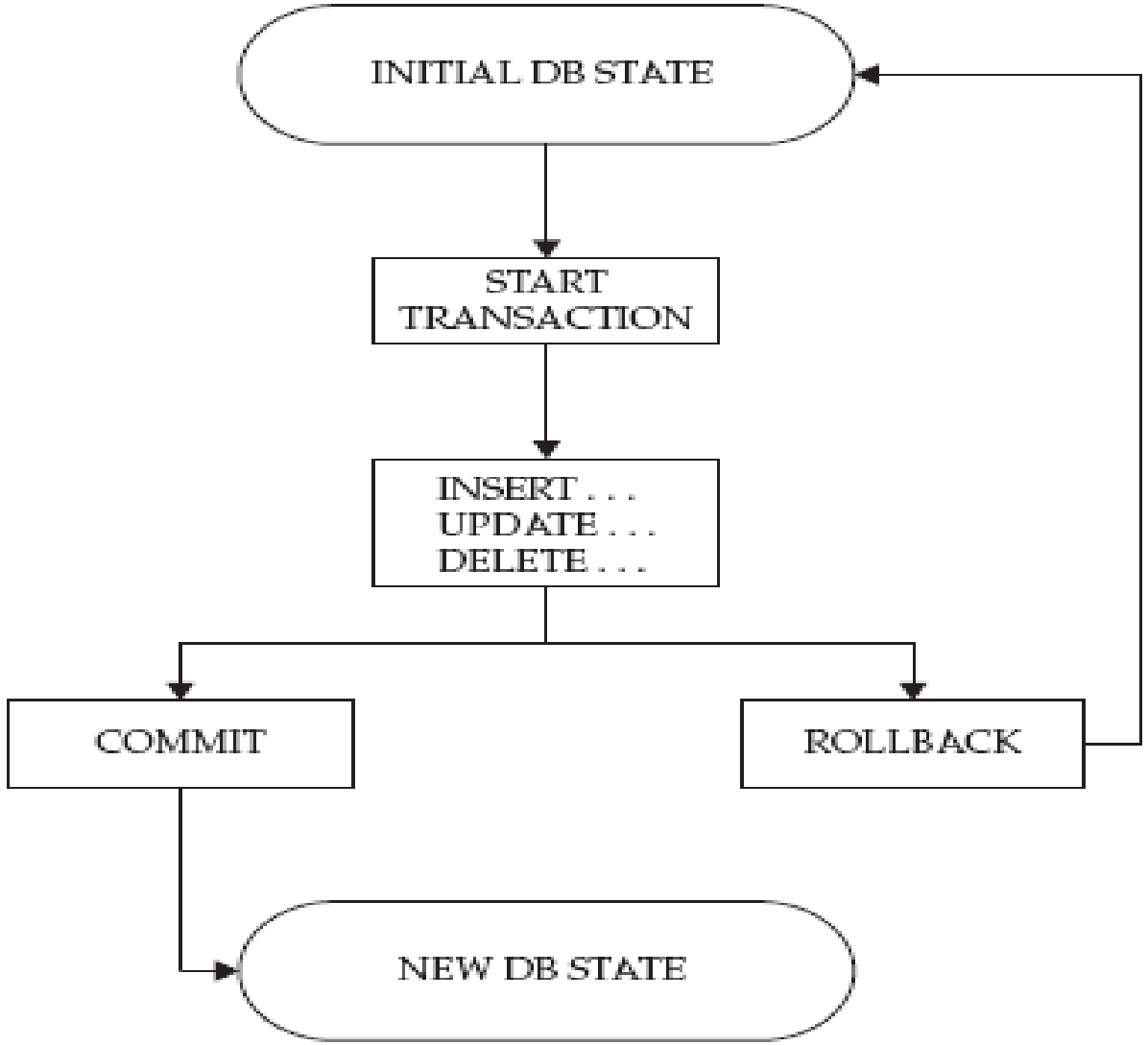
并发控制

- 锁策略：在锁粒度及数据安全性寻求的平衡机制
- 显式使用锁
  - LOCK TABLES 加锁
    tbl_name [[AS] alias] lock_type
    [, tbl_name [[AS] alias] lock_type] ...
    lock_type: READ ， WRITE
    UNLOCK TABLES 解锁
  - FLUSH TABLES [tb_name[,...]] [WITH READ LOCK]
    关闭正在打开的表（清除查询缓存），通常在备份前加全局读锁
  - SELECT clause [FOR UPDATE | LOCK IN SHARE MODE]
    查询时加写或读锁

事务

- 事务Transactions：一组原子性的SQL语句，或一个独立工作单元
- 事务日志：记录事务信息，实现undo,redo等故障恢复功能
- ACID特性：
  - A：atomicity原子性；整个事务中的所有操作要么全部成功执行，要么全部失败后回滚
  - C：consistency一致性；数据库总是从一个一致性状态转换为另一个一致性状态
  - I：Isolation隔离性；一个事务所做出的操作在提交之前，是不能为其它事务所见；隔离有多种隔离级别，实现并发
  - D：durability持久性；一旦事务提交，其所做的修改会永久保存于数据库中

Transaction生命周期



事务

- 启动事务：
  - BEGIN
  - BEGIN WORK
  - START TRANSACTION
- 结束事务：
  - COMMIT：提交
  - ROLLBACK: 回滚
  - 注意：只有事务型存储引擎中的DML语句方能支持此类操作
- 自动提交：set autocommit={1|0}　默认为1，为0时设为非自动提交
  - 建议：显式请求和提交事务，而不要使用“自动提交”功能
- 事务支持保存点：savepoint
  - SAVEPOINT identifier
  - ROLLBACK [WORK] TO [SAVEPOINT] identifier
  - RELEASE SAVEPOINT identifier

事务隔离级别

- 事务隔离级别：从上至下更加严格
  - READ UNCOMMITTED 可读取到未提交数据，产生脏读
  - READ COMMITTED 可读取到提交数据，但未提交数据不可读，产生不可重复读，即可读取到多个提交数据，导致每次读取数据不一致
  - REPEATABLE READ 可重复读，多次读取数据都一致，产生幻读，即读取过程中，即使有其它提交的事务修改数据，仍只能读取到未修改前的旧数据。此为MySQL默认设置
  - SERIALIZABILE 可串行化，未提交的读事务阻塞修改事务，或者未提交的修改事务阻塞读事务。导致并发性能差
- MVCC: 多版本并发控制，和事务级别相关

事务隔离级别

事务隔离级别	脏读可能性	不可重复读可能性	幻读可能性	加锁读
读未提交（read-uncommitted）	是	是	是	否
不可重复读（read-committed）	否	是	是	否
可重复读（repeatable-read）	否	否	是	否
串行化（serializable）	否	否	否	是

事务

- 指定事务隔离级别：
  - 服务器变量tx_isolation指定，默认为REPEATABLE-READ，可在GLOBAL和SESSION级进行设置

    SET tx_isolation=''

    READ-UNCOMMITTED

    READ-COMMITTED

    REPEATABLE-READ

    SERIALIZABLE
  - 服务器选项中指定

    vim /etc/my.cnf

    [mysqld]

    transaction-isolation=SERIALIZABLE

并发控制

- 死锁：

  两个或多个事务在同一资源相互占用，并请求锁定对方占用的资源的状态

- 事务日志：

  事务日志的写入类型为“追加”，因此其操作为“顺序IO”；通常也被称为：预写式日志 write ahead logging

  事务日志文件： ib_logfile0， ib_logfile1

日志

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

事务日志

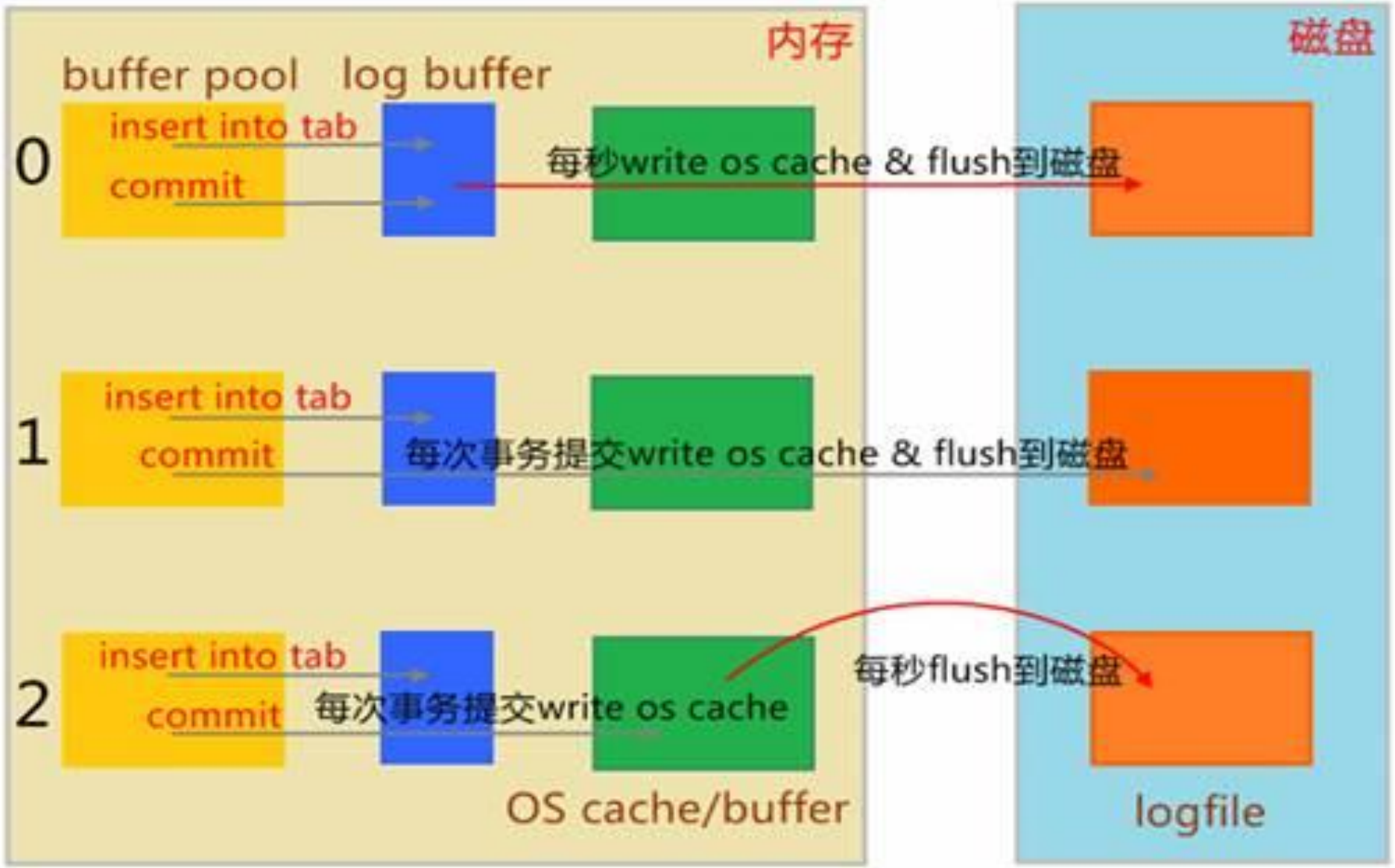
- 事务日志：transaction log
  - 事务型存储引擎自行管理和使用，建议和数据文件分开存放
    redo log
    undo log
  - Innodb事务日志相关配置：
    show variables like '%innodb_log%';

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
innodb_flush_log_at_trx_commit		默认为1

事务日志

- innodb_flush_log_at_trx_commit
- 说明：设置为1，同时sync_binlog = 1表示最高级别的容错 innodb_use_global_flush_log_at_trx_commit的值确定是否可以使用SET语句重置此变量
- 1默认情况下，日志缓冲区将写入日志文件，并在每次事务后执行刷新到磁盘。这是完全遵守ACID特性
- 0提交时没有任何操作; 而是每秒执行一次日志缓冲区写入和刷新。 这样可以提供更好的性能，但服务器崩溃可能丢失最后一秒的事务
- 2每次提交后都会写入日志缓冲区，但每秒都会进行一次刷新。 性能比0略好一些，但操作系统或停电可能导致最后一秒的交易丢失
- 3模拟MariaDB 5.5组提交（每组提交3个同步），此项MariaDB 10.0支持

事务日志优化


内存
磁盘
buffer pool
log buffer
0
insert into tab
commit
每秒write os cache & flush到磁盘
1
insert into tab
commit
每次事务提交write os cache & flush到磁盘
2
insert into tab
commit
每次事务提交write os cache
每秒flush到磁盘
OS cache/buffer
logfile

错误日志

- 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

通用日志

- 通用日志：记录对数据库的通用操作，包括错误的SQL语句

  文件：file，默认值

  表：table

- 通用日志相关设置

  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

上述查询类型且查询时长超过long_query_time，则记录日志

log_queries_not_using_indexes=ON　不使用索引或使用全索引扫描，不论是否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

二进制日志

- 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

  记录已提交的日志

  不依赖于存储引擎类型

  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

- 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的事件

二进制日志

- 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： mariadb-bin.000001

  索引文件：mysql|mariadb-bin.index，文本格式

二进制日志

- 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

二进制日志

- 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，及大小

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 ‘mysql-bin.000001' from 6516 limit 2,3

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]
  -v -vvv

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

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

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之前的日志

  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 #]; 删除所有二进制日志文件，并重新生成日志文件，文件名从#开始记数，默认从1开始，一般是master主机第一次启动时执行，MariaDB10.1.6开始支持TO #

- 切换日志文件：

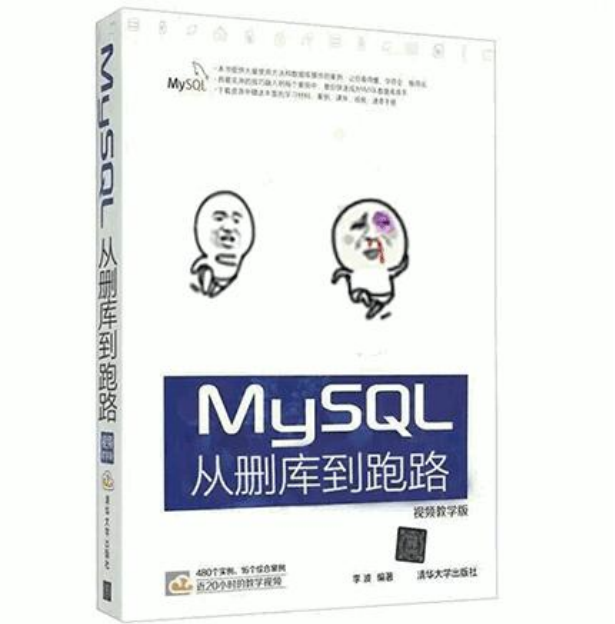
  FLUSH LOGS;

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、函数、触发器、事件调度器）
  - 服务器的配置文件

备份和恢复

- 备份工具
  - 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；借助文件系统工具进行备份
  - 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，结合binlog的增量备份
  - 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
  - mysqlhotcopy：PERL 语言实现，几乎冷备，仅适用于MyISAM存储引擎，使用LOCK TABLES、FLUSH TABLES和cp或scp来快速备份数据库

基于LVM的备份

- (1) 请求锁定所有表

  mysql> FLUSH TABLES WITH READ LOCK;
- (2) 记录二进制日志文件及事件位置

  mysql> FLUSH LOGS;

  mysql> SHOW MASTER STATUS;

  mysql -e 'SHOW MASTER STATUS' > /PATH/TO/SOMEFILE
- (3) 创建快照

  lvcreate -L # -s -p r -n NAME /DEV/VG_NAME/LV_NAME
- (4) 释放锁

  mysql> UNLOCK TABLES;
- (5) 挂载快照卷，执行数据备份
- (6) 备份完成后，删除快照卷
- (7) 制定好策略，通过原卷备份二进制日志

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mysql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自定义函数
  - --triggers：备份表相关触发器，默认启用,用--skip-triggers，不备份触发器
  - --default-character-set=utf8 指定字符集
  - --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-x | --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 或 -B 选项时，会导致刷新多次数据库。建议在同一时刻执行转储和日志刷新，可通过和--single-transaction或-x，--master-data 一起使用实现，此时只刷新一次日志
  - --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，当有包括BINARY，VARBINARY，BLOB，BIT的数据类型的列时使用，避免乱码
  - -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指令开启事务

  此选项通过在单个事务中转储所有表来创建一致的快照。 仅适用于存储在支持多版本控制的存储引擎中的表（目前只有InnoDB可以）；转储不保证与其他存储引擎保持一致。 在进行单事务转储时，要确保有效的转储文件（正确的表内容和二进制日志位置），没有其他连接应该使用以下语句：ALTER TABLE，DROP TABLE，RENAME TABLE，TRUNCATE TABLE

  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

生产环境实战备份策略

- InnoDB建议备份策略

  mysqldump –uroot –A –F –E –R --single-transaction --master-data=1 --flush-privileges --triggers --default-character-set=utf8 --hex-blob >$BACKUP/fullbak_$BACKUP_TIME.sql

- MyISAM建议备份策略

  mysqldump –uroot –A –F –E –R –x --master-data=1 --flush-privileges --triggers --default-character-set=utf8 --hex-blob >$BACKUP/fullbak_$BACKUP_TIME.sql

xtrabackup

- Percona

  官网：www.percona.com

  percona-server

  InnoDB --> XtraDB

- Xtrabackup

  percona提供的mysql数据库备份工具，惟一开源的能够对innodb和xtradb数据库进行热备的工具

  手册：https://www.percona.com/doc/percona-xtrabackup/LATEST/index.html

- 特点：
  - 备份还原过程快速、可靠
  - 备份过程不会打断正在执行的事务
  - 能够基于压缩等功能节约磁盘空间和流量
  - 自动实现备份检验
  - 开源，免费

xtrabackup

- Xtrabackup2.2版之前包括4个可执行文件：

  innobackupex:　　Perl 脚本

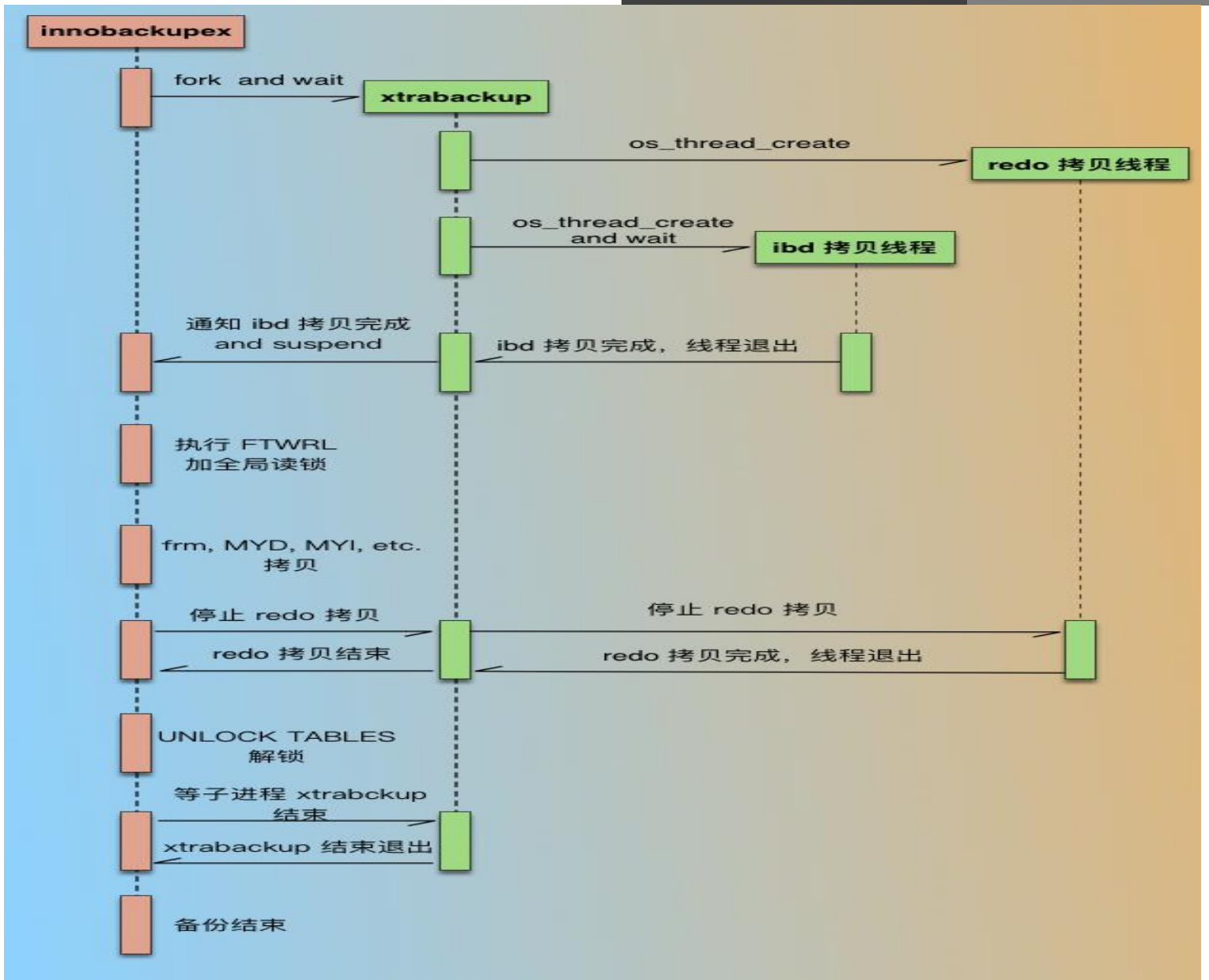
  xtrabackup: C/C++　　编译的二进制

  xbcrypt:　　加解密

  xbstream:　　支持并发写的流文件格式

- xtrabackup 是用来备份 InnoDB 表的，不能备份非 InnoDB 表，和 MySQL Server 没有交互
- innobackupex 脚本用来备份非 InnoDB 表，同时会调用 xtrabackup 命令来备份 InnoDB 表，还会和 MySQL Server 发送命令进行交互，如加全局读锁（FTWRL）、获取位点（SHOW SLAVE STATUS）等。即innobackupex是在xtrabackup 之上做了一层封装实现的

xtrabackup备份过程



xtrabackup的新版变化

- xtrabackup版本升级到2.4后，相比之前的2.1有了比较大的变化：innobackupex 功能全部集成到 xtrabackup 里面，只有一个 binary程序，另外为了兼容考虑，innobackupex作为 xtrabackup 的软链接，即xtrabackup现在支持非Innodb表备份，并且 Innobackupex 在下一版本中移除，建议通过xtrabackup替换innobackupex
- xtrabackup安装：

  yum install percona-xtrabackup 在EPEL源中

  最新版本下载安装：

  https://www.percona.com/downloads/XtraBackup/LATEST/

xtrabackup用法

- 备份：innobackupex [option] BACKUP-ROOT-DIR
- 选项说明：https://www.percona.com/doc/percona-xtrabackup/LATEST/genindex.html
  - --user：该选项表示备份账号
  - --password：该选项表示备份的密码
  - --host：该选项表示备份数据库的地址
  - --databases：该选项接受的参数为数据库名，如果要指定多个数据库，彼此间需要以空格隔开；如："xtra_test dba_test"，同时，在指定某数据库时，也可以只指定其中的某张表。如："mydatabase.mytable"。该选项对innodb引擎表无效，还是会备份所有innodb表
  - --defaults-file：该选项指定从哪个文件读取MySQL配置，必须放在命令行第一个选项位置
  - --incremental：该选项表示创建一个增量备份，需要指定--incremental-basedir
  - --incremental-basedir：该选项指定为前一次全备份或增量备份的目录，与--incremental同时使用
  - --incremental-dir：该选项表示还原时增量备份的目录
  - --include=name：指定表名，格式：databasename.tablename

xtrabackup用法

- Prepare：innobackupex --apply-log [option] BACKUP-DIR
- 选项说明：
- --apply-log：一般情况下,在备份完成后，数据尚且不能用于恢复操作，因为备份的数据中可能会包含尚未提交的事务或已经提交但尚未同步至数据文件中的事务。因此，此时数据文件仍处理不一致状态。此选项作用是通过回滚未提交的事务及同步已经提交的事务至数据文件使数据文件处于一致性状态
- --use-memory：和--apply-log选项一起使用，当prepare 备份时，做crash recovery分配的内存大小，单位字节，也可1MB,1M,1G,1GB等，推荐1G
- --export：表示开启可导出单独的表之后再导入其他Mysql中
- --redo-only：此选项在prepare base full backup，往其中合并增量备份时候使用，但不包括对最后一个增量备份的合并

xtrabackup用法

- 还原：innobackupex --copy-back [选项] BACKUP-DIR
- innobackupex --move-back [选项] [--defaults-group=GROUP-NAME] BACKUP-DIR
- 选项说明：
- --copy-back：做数据恢复时将备份数据文件拷贝到MySQL服务器的datadir
- --move-back：这个选项与--copy-back相似，唯一的区别是它不拷贝文件，而是移动文件到目的地。这个选项移除backup文件，用时候必须小心。使用场景：没有足够的磁盘空间同事保留数据文件和Backup副本

xtrabackup用法

- 还原注意事项：
  - 1.datadir 目录必须为空。除非指定innobackupex --force-non-empty-directorires选项指定，否则--copy-back选项不会覆盖
  - 2.在restore之前,必须shutdown MySQL实例，不能将一个运行中的实例restore到datadir目录中
  - 3.由于文件属性会被保留，大部分情况下需要在启动实例之前将文件的属主改为mysql，这些文件将属于创建备份的用户

    chown -R mysql:mysql /data/mysql

    以上需要在用户调用innobackupex之前完成

    --force-non-empty-directories：指定该参数时候，使得innobackupex --copy-back或--move-back选项转移文件到非空目录，已存在的文件不会被覆盖。如果--copy-back和--move-back文件需要从备份目录拷贝一个在datadir已经存在的文件，会报错失败

备份生成的相关文件

- 使用innobackupex备份时，其会调用xtrabackup备份所有的InnoDB表，复制所有关于表结构定义的相关文件(.frm)、以及MyISAM、MERGE、CSV和ARCHIVE表的相关文件，同时还会备份触发器和数据库配置信息相关的文件。这些文件会被保存至一个以时间命名的目录中,在备份时，innobackupex还会在备份目录中创建如下文件：
- (1)xtrabackup_info：innobackupex工具执行时的相关信息，包括版本，备份选项，备份时长，备份LSN(log sequence number日志序列号)，BINLOG的位置
- (2)xtrabackup_checkpoints：备份类型（如完全或增量）、备份状态（如是否已经为prepared状态）和LSN范围信息,每个InnoDB页(通常为16k大小)都会包含一个日志序列号LSN。LSN是整个数据库系统的系统版本号，每个页面相关的LSN能够表明此页面最近是如何发生改变的
- (3)xtrabackup_binlog_info：MySQL服务器当前正在使用的二进制日志文件及至备份这一刻为止二进制日志事件的位置，可利用实现基于binlog的恢复
- (4)backup-my.cnf：备份命令用到的配置选项信息
- (5)xtrabackup_logfile：备份生成的日志文件

示例：新版xtrabackup完全备份及还原

- 1 在原主机做完全备份到/backup

  xtrabackup --backup --target-dir=/backup/

  scp -r /backup/*　　目标主机:/backup

- 2 在目标主机上

  1）预准备：确保数据一致，提交完成的事务，回滚未完成的事务

  xtrabackup --prepare --target-dir=/backup/

  2）复制到数据库目录

  注意：数据库目录必须为空，MySQL服务不能启动

  xtrabackup --copy-back --target-dir=/backup/

  3）还原属性

  chown -R mysql:mysql /var/lib/mysql

  4）启动服务

  systemctl start mariadb

示例：新版xtrabackup完全，增量备份及还原

- 1 备份过程

1）完全备份：xtrabackup --backup --target-dir=/backup/base

2）第一次修改数据

3）第一次增量备份

xtrabackup --backup --target-dir=/backup/inc1 --incremental-basedir=/backup/base

4）第二次修改数据

5）第二次增量

xtrabackup --backup --target-dir=/backup/inc2 --incremental-basedir=/backup/inc1

6）scp -r /backup/*　目标主机:/backup/

- 备份过程生成三个备份目录

/backup/{base，inc1，inc2}

示例：新版xtrabackup完全，增量备份及还原

◆ 2还原过程

1）预准备完成备份，此选项--apply-log-only 阻止回滚未完成的事务

xtrabackup --prepare --apply-log-only --target-dir=/backup/base

2）合并第1次增量备份到完全备份，

xtrabackup --prepare --apply-log-only --target-dir=/backup/base --incremental-dir=/backup/inc1

3）合并第2次增量备份到完全备份：最后一次还原不需要加选项--apply-log-only

xtrabackup --prepare --target-dir=/backup/base --incremental-dir=/backup/inc2

4）复制到数据库目录，注意数据库目录必须为空，MySQL服务不能启动

xtrabackup --copy-back --target-dir=/backup/base

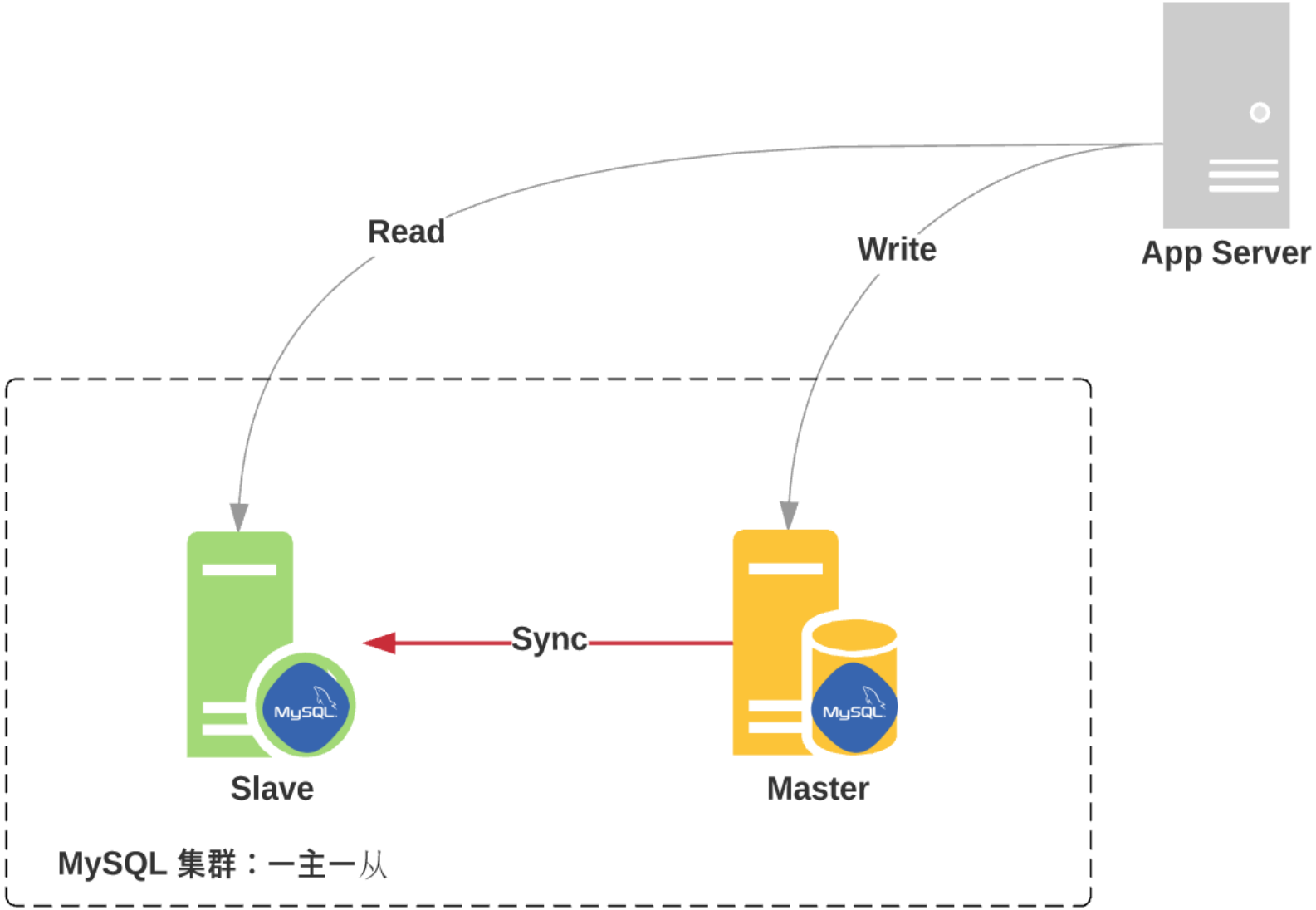
5）还原属性：chown -R mysql:mysql /var/lib/mysql

6）启动服务：systemctl start mariadb

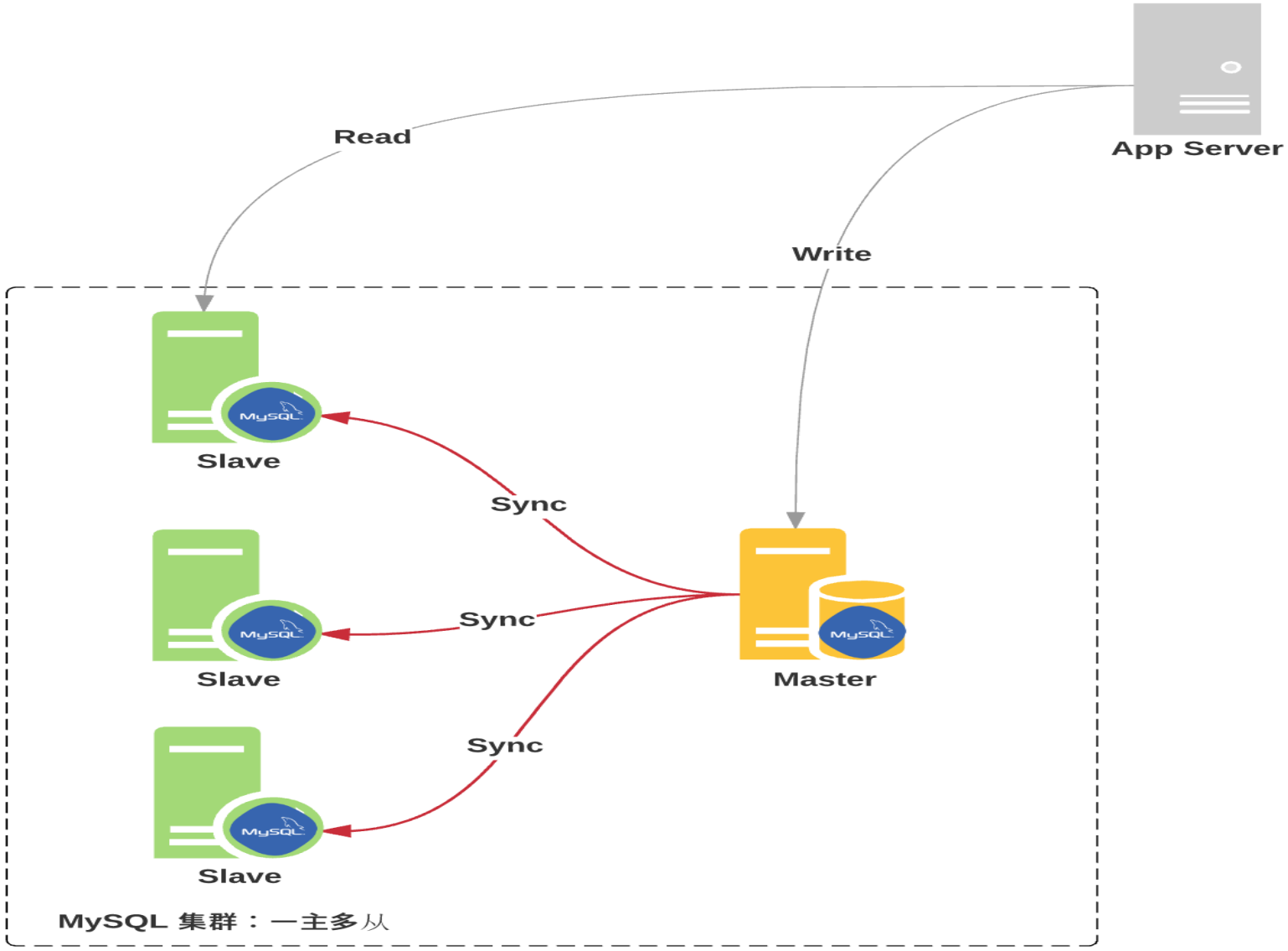
MySQL复制

- 扩展方式： Scale Up ，Scale Out
- MySQL的扩展
  - 读写分离
  - 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

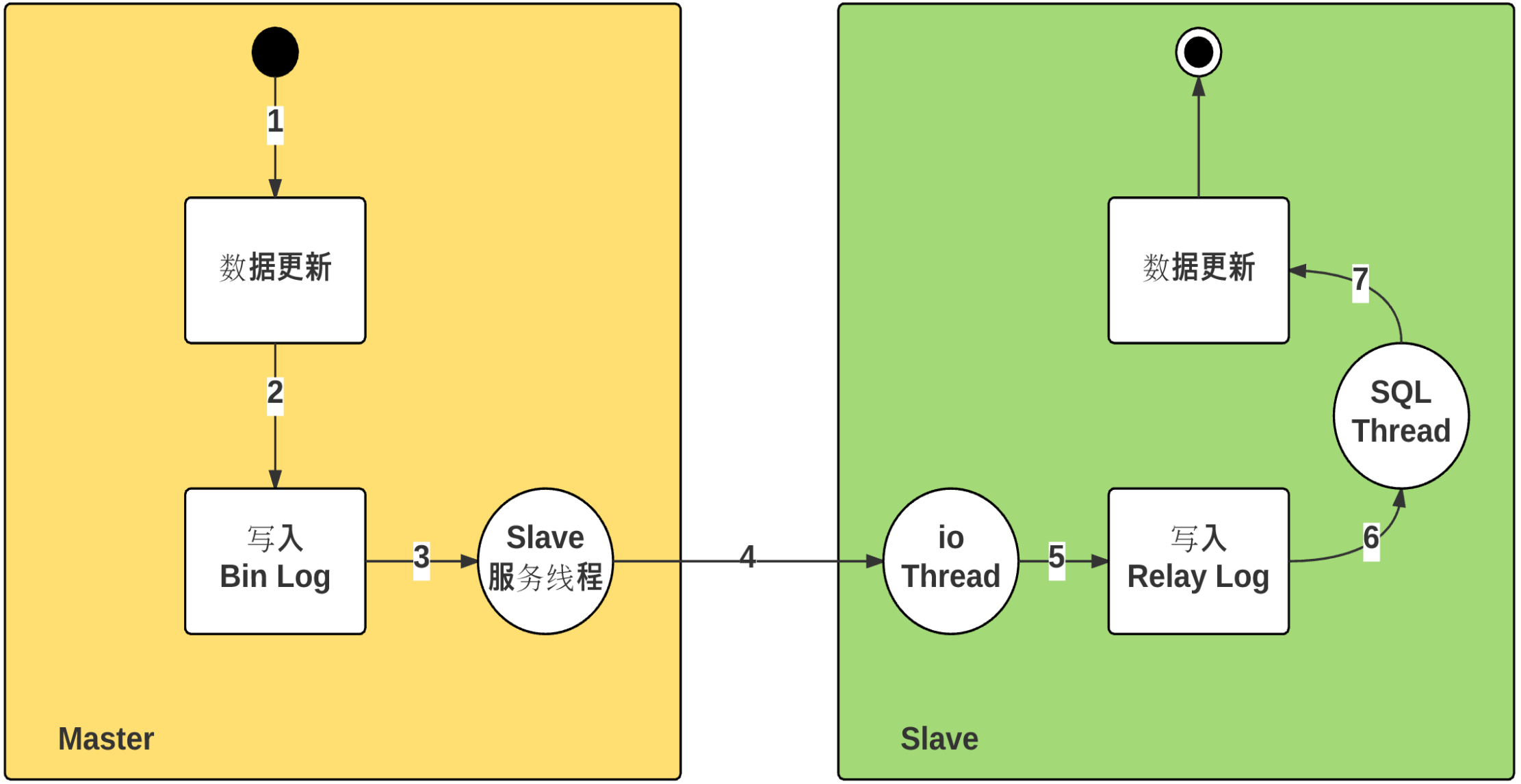
一主一从



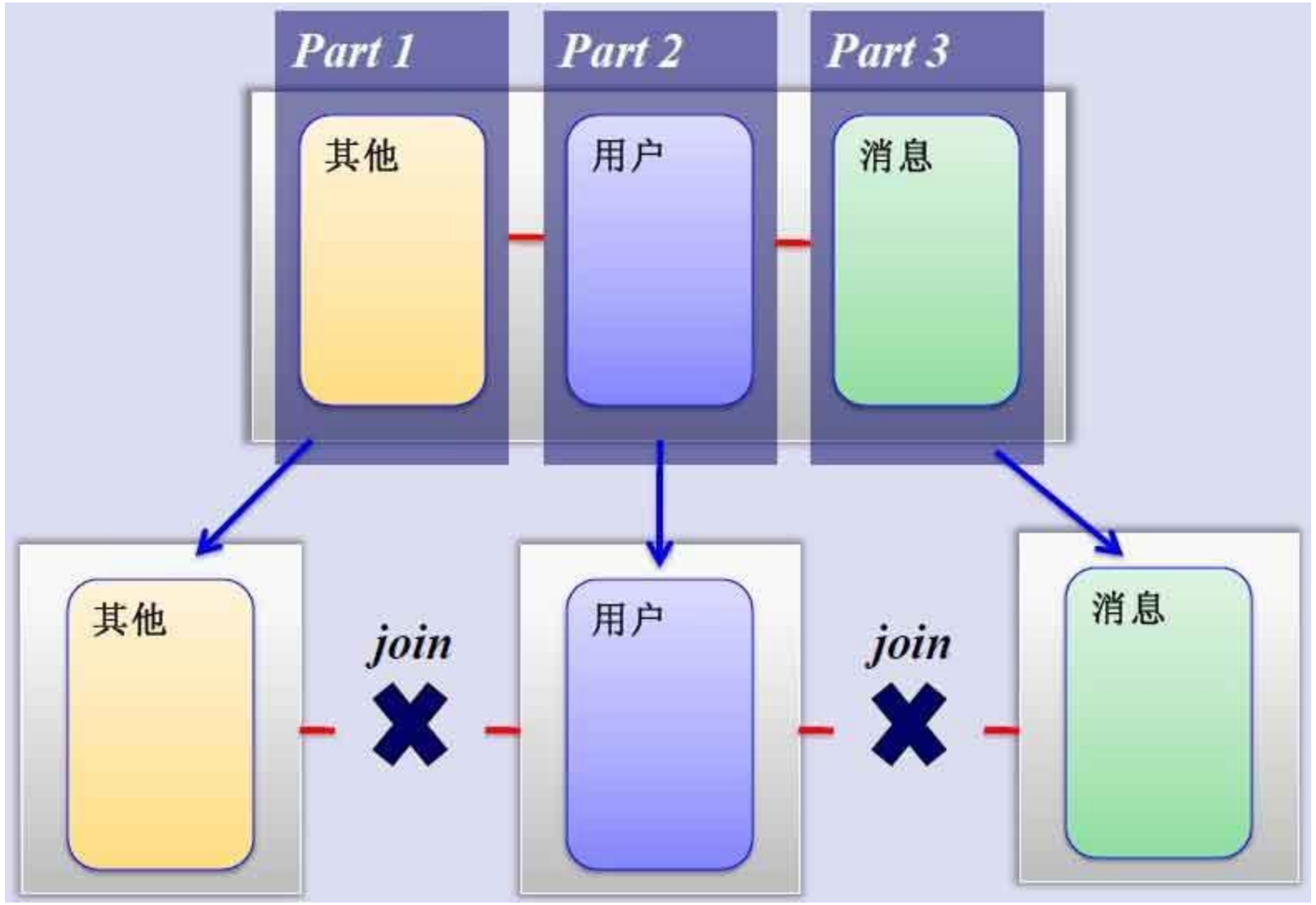
一主多从


App Server
Read
Write
Slave
Sync
Slave
Sync
Master
Sync
Slave
MySQL 集群：一主多从

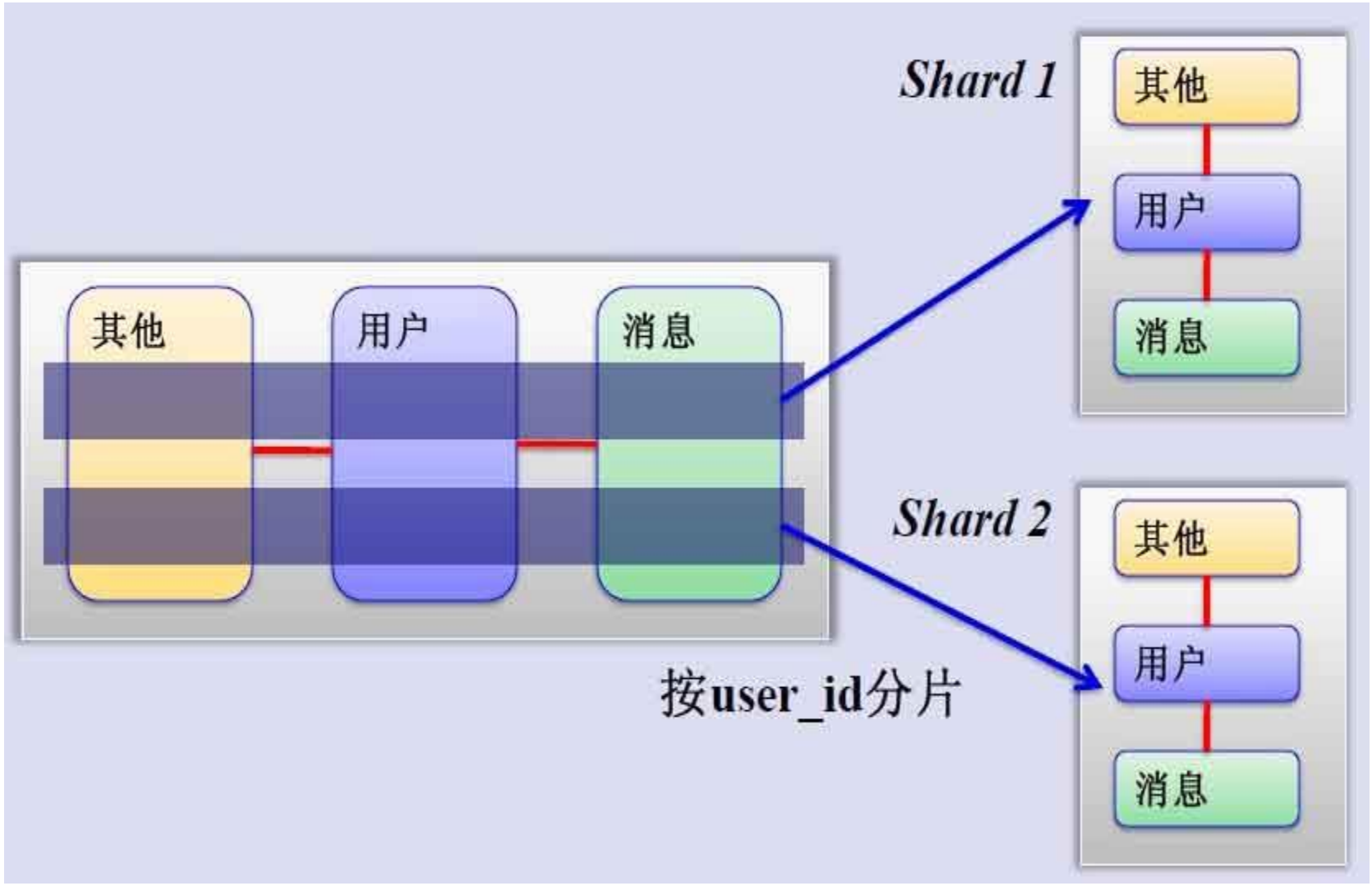
主从复制原理


1
数据更新
2
写入
Bin Log
3
Slave
服务线程
4
io
Thread
5
写入
Relay Log
6
SQL
Thread
7
数据更新
Master
Slave

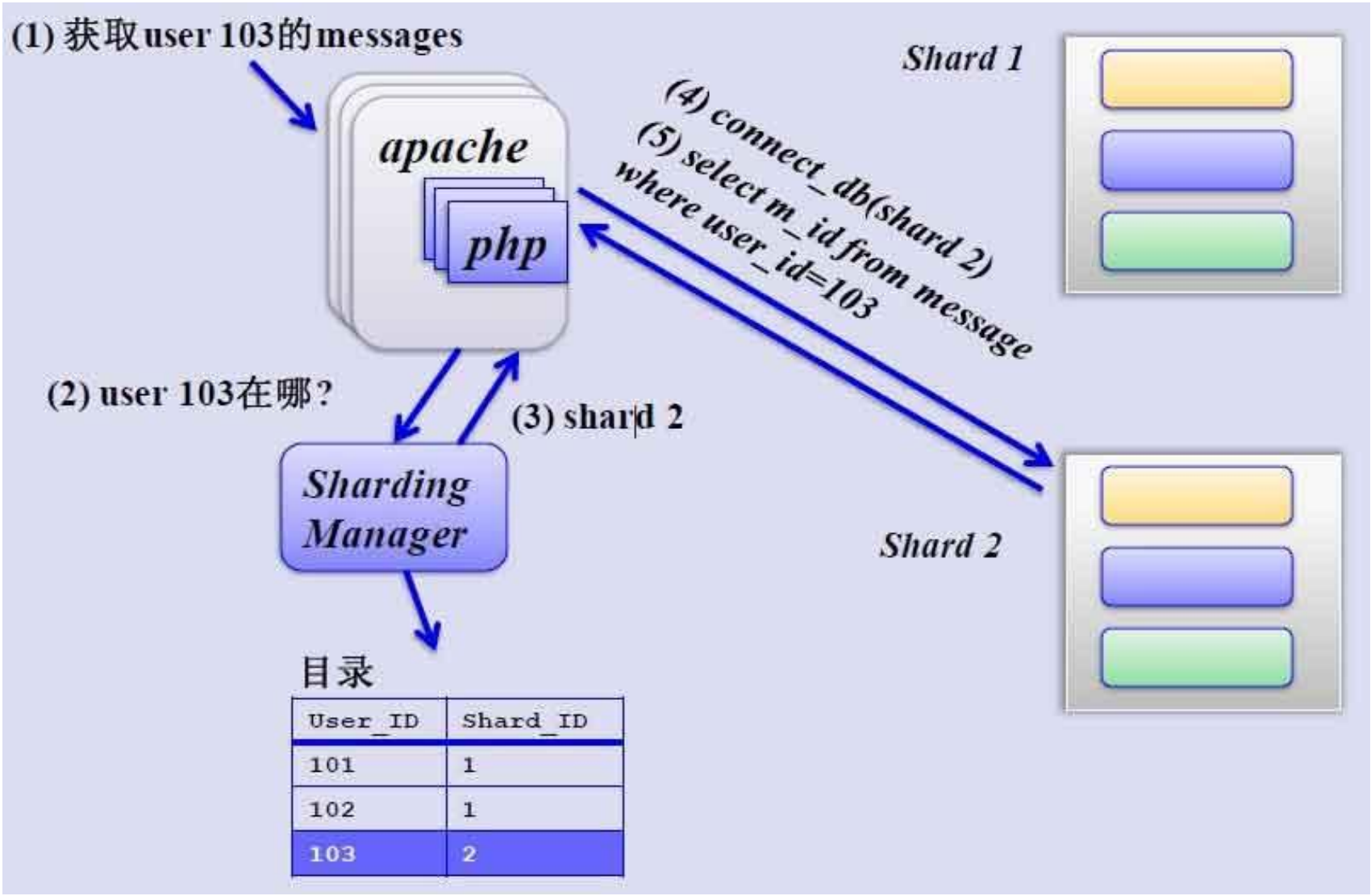
MySQL垂直分区



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：

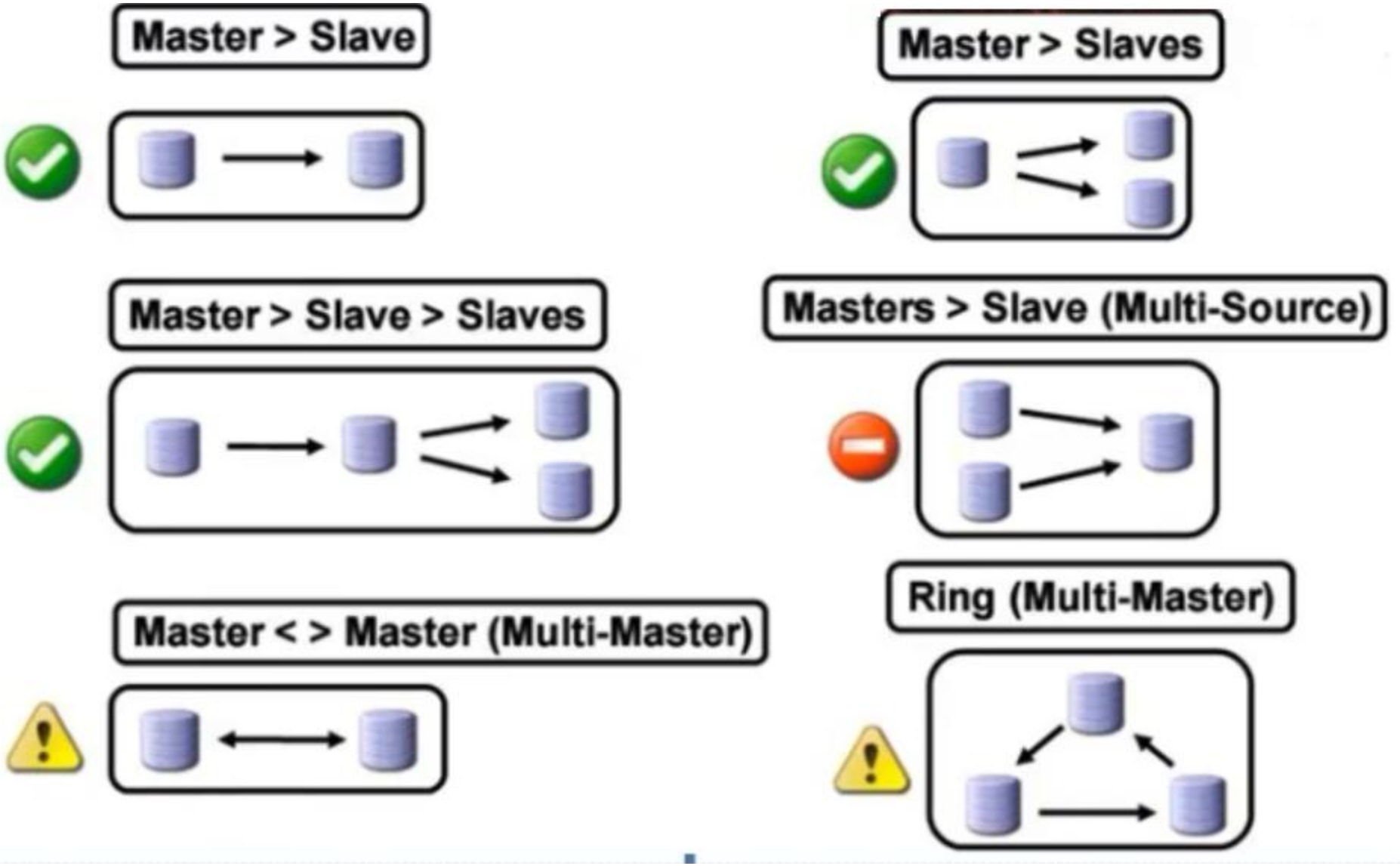
    I/O Thread：向Master请求二进制日志事件，并保存于中继日志中

    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relay 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

MySQL复制模型



MySQL复制

- 主从配置过程：参看官网
  https://mariadb.com/kb/en/library/setting-up-replication/
  https://dev.mysql.com/doc/refman/5.5/en/replication-configuration.html
- 主节点配置：

(1) 启用二进制日志

```
[mysqld]
log_bin
```

(2) 为当前节点设置一个全局惟一的ID号

```
[mysqld]
server_id=#
log-basename=master  可选项，设置datadir中日志名称，确保不依赖主机名
```

(3) 创建有复制权限的用户账号

```
GRANT REPLICATION 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
read_only=ON                 设置数据库只读
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=' mariadb-bin.xxxxxx', 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;
- 2、RESET SLAVE：从服务器清除master.info ，relay-log.info, relay log ，开始新的relay log

  RESET SLAVE ALL：清除所有从服务器上设置的主服务器同步信息，如PORT, HOST, USER和 PASSWORD 等

  注意：以上都需要先STOP SLAVE
- 3、sql_slave_skip_counter = N 从服务器忽略几个主服务器的复制事件，global变量

MySQL复制

- 4、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_log_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=#　　#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=#　　#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

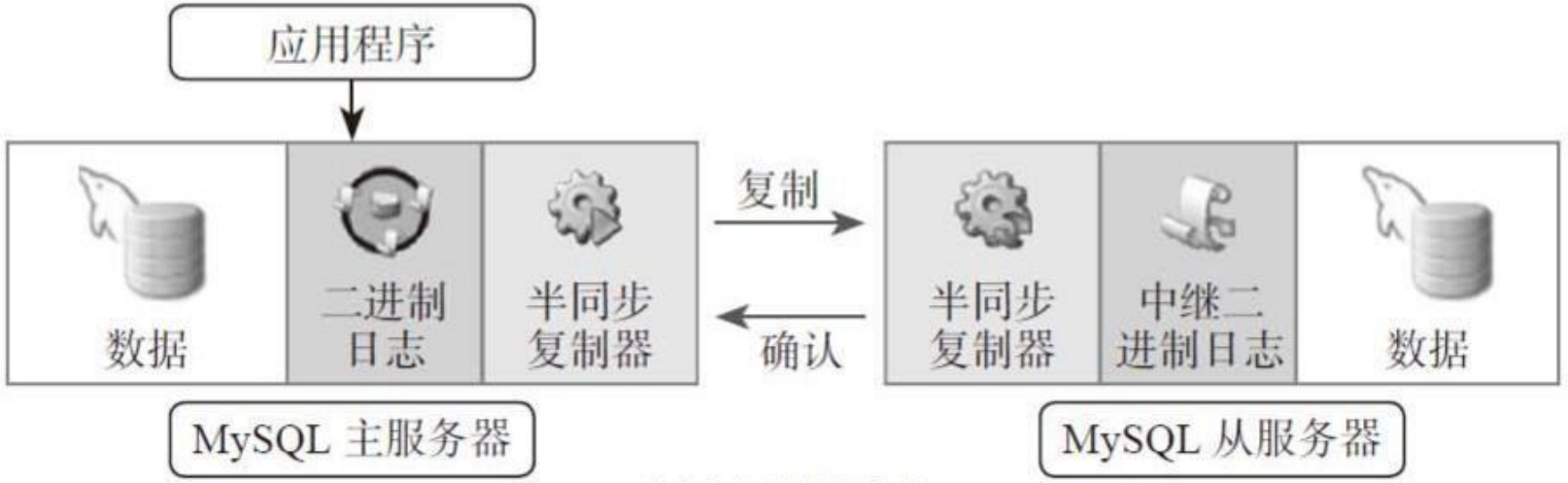
主主复制

- 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 rpl_semi_sync_master_enabled=1;
mysql>SET GLOBAL rpl_semi_sync_master_timeout = 1000;超时长1s
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 rpl_semi_sync_slave_enabled=1;
```

MySQL复制

- 复制过滤器：

  让从节点仅复制指定的数据库，或指定数据库的指定表

- 两种实现方式：
  - (1) 服务器选项：主服务器仅向二进制日志中记录与特定数据库相关的事件

    注意：此项和binlog_format相关

    参看：https://mariadb.com/kb/en/library/mysqld-options/#-binlog-ignore-db

    binlog-do-db = 数据库白名单列表，多个数据库需多行实现

    binlog-ignore-db = 数据库黑名单列表

    问题：基于二进制还原将无法实现；不建议使用
  - (2) 从服务器SQL_THREAD在relay log中的事件时，仅读取与特定数据库(特定表)相关的事件并应用于本地

    问题：会造成网络及磁盘IO浪费

MySQL复制

- 从服务器上的复制过滤器相关变量
  - replicate_do_db= 指定复制库的白名单
  - replicate_ignore_db= 指定复制库黑名单
  - replicate_do_table= 指定复制表的白名单
  - replicate_ignore_table= 指定复制表的黑名单
  - replicate_wild_do_table= foo%.bar% 支持通配符
  - replicate_wild_ignore_table=

MySQL复制加密

- 基于SSL复制：

  在默认的主从复制过程或远程连接到MySQL/MariaDB所有的链接通信中的数据都是明文的，外网里访问数据或则复制，存在安全隐患。通过SSL/TLS加密的方式进行复制的方法，来进一步提高数据的安全性

- 配置实现：

  参看：https://mariadb.com/kb/en/library/replication-with-secure-connections/

  - 主服务器开启SSL，配置证书和私钥路径
  - 并且创建一个要求必须使用SSL连接的复制账号

    mysql>GRANT REPLICATION SLAVE ON *.* TO 'repluser'@'192.168.8.%' IDENTIFIED BY ‘magedu' **REQUIRE SSL;**
  - 从服务器使用CHANGER MASTER TO 命令时指明ssl相关选项

MySQL复制加密

- Master服务器配置

```
[mysqld]
log-bin
server_id=1
ssl
ssl-ca=/etc/my.cnf.d/ssl/cacert.pem
ssl-cert=/etc/my.cnf.d/ssl/master.crt
ssl-key=/etc/my.cnf.d/ssl/master.key
```

MySQL复制加密

- Slave服务器配置

```
mysql>
CHANGE MASTER TO
MASTER_HOST='MASTERIP',
MASTER_USER='rep',
MASTER_PASSWORD='centos',
MASTER_LOG_FILE='mariadb-bin.000001',
MASTER_LOG_POS=245,
MASTER_SSL=1,
MASTER_SSL_CA = '/etc/my.cnf.d/ssl/cacert.pem',
MASTER_SSL_CERT = '/etc/my.cnf.d/ssl/slave.crt',
MASTER_SSL_KEY = '/etc/my.cnf.d/ssl/slave.key';
```

复制的监控和维护

- (1) 清理日志

  PURGE { BINARY | MASTER } LOGS { TO 'log_name' | BEFORE datetime_expr }

  RESET MASTER

  RESET SLAVE

- (2) 复制监控

  SHOW MASTER STATUS

  SHOW BINARY LOGS

  SHOW BINLOG EVENTS

  SHOW SLAVE STATUS

  SHOW PROCESSLIST

复制的监控和维护

- (3) 从服务器是否落后于主服务
  Seconds_Behind_Master：0
- (4) 如何确定主从节点数据是否一致
  percona-tools
- (5) 数据不一致如何修复
  删除从数据库，重新复制

GTID复制

- GTID复制：（global transaction id 全局事务标识符） MySQL5.6版本开始支持，GTID复制不像传统的复制方式（异步复制、半同步复制）需要找到binlog和POS点，只需知道master的IP、端口、账号、密码即可。开启GTID后，执行change master to master_auto_postion=1即可，它会自动寻找同步
- GTID 架构

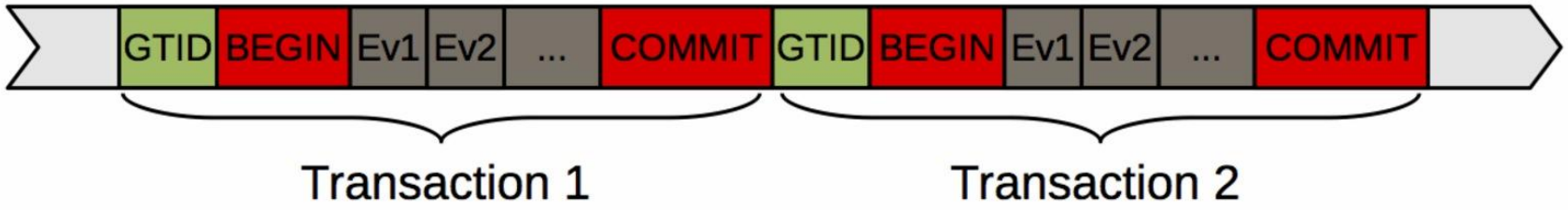
  GTID = server_uuid:transaction_id，在一组复制中，全局唯一

  server_uuid 来源于 auto.cnf
- GTID服务器相关选项

  gtid_mode　　gtid模式

  enforce_gtid_consistency　　保证GTID安全的参数

GTID在binlog中的结构和GTID event 结构



GTID配置示例

- 主服务器

```
vim /etc/my.cnf
server-id=1
log-bin=mysql-bin
gtid_mode=ON
enforce_gtid_consistency

mysql> grant replication slave on *.* to 'repluser'@'192.168.8.%'
identified by 'P@ssw0rd!';
```

GTID配置示例

- 从服务器

```
vim /etc/my.cnf
server-id=2
gtid_mode=ON
enforce_gtid_consistency

mysql>CHANGE MASTER TO  MASTER_HOST='192.168.8.100',
  MASTER_USER='repluser',
  MASTER_PASSWORD='P@ssw0rd!',
  MASTER_PORT=3306,
  MASTER_AUTO_POSITION=1;

mysql>start slave;
```

MySQL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

  Cetus：网易乐得，https://github.com/Lede-Inc/cetus

  Amoeba：https://sourceforge.net/projects/amoeba/

  Cobar：阿里巴巴，Amoeba的升级版

  Mycat：基于Cobar， http://www.mycat.io/

  ProxySQL：https://proxysql.com/

ProxySQL

- ProxySQL： MySQL中间件
  两个版本：官方版和percona版，percona版是基于官方版基础上修改
  C++语言开发，轻量级但性能优异(支持处理千亿级数据)
  具有中间件所需的绝大多数功能，包括：
  - 多种方式的读/写分离
  - 定制基于用户、基于schema、基于语句的规则对SQL语句进行路由
  - 缓存查询结果
  - 后端节点监控
- 官方站点：https://proxysql.com/
- 官方手册：https://github.com/sysown/proxysql/wiki

ProxySQL安装

- 准备：

  实现读写分离前，先实现主从复制

  注意：slave节点需要设置read_only=1

- 基于YUM仓库安装

```
cat <<EOF | tee /etc/yum.repos.d/proxysql.repo
[proxysql_repo]
name= ProxySQL YUM repository
baseurl=http://repo.proxysql.com/ProxySQL/proxysql-1.4.x/centos/\$releasever
gpgcheck=1
gpgkey=http://repo.proxysql.com/ProxySQL/repo_pub_key
EOF
```

- 基于RPM下载安装：https://github.com/sysown/proxysql/releases

ProxySQL安装

- ProxySQL组成
  - 服务脚本：/etc/init.d/proxysql
  - 配置文件：/etc/proxysql.cnf
  - 主程序：/usr/bin/proxysql
  - 基于SQLITE的数据库文件：/var/lib/proxysql/
- 启动ProxySQL：service proxysql start
  - 启动后会监听两个默认端口
  - 6032：ProxySQL的管理端口
  - 6033：ProxySQL对外提供服务的端口
- 使用mysql客户端连接到ProxySQL的管理接口6032，默认管理员用户和密码都是admin：

```
mysql -uadmin -padmin -P6032 -h127.0.0.1
```

ProxySQL实现读写分离

- 数据库说明：

  main 是默认的”数据库”名，表里存放后端db实例、用户验证、路由规则等信息。
  表名以 runtime_开头的表示proxysql当前运行的配置内容，不能通过dml语句修改，
  只能修改对应的不以 runtime_ 开头的（在内存）里的表，然后 LOAD 使其生效，
  SAVE 使其存到硬盘以供下次重启加载

  disk 是持久化到硬盘的配置，sqlite数据文件

  stats 是proxysql运行抓取的统计信息，包括到后端各命令的执行次数、流量、
  processlist、查询种类汇总/执行时间，等等

  monitor 库存储 monitor 模块收集的信息，主要是对后端db的健康/延迟检查

ProxySQL实现读写分离

- 说明：

在main和monitor数据库中的表， runtime_开头的是运行时的配置，不能修改，只能修改非runtime_表

修改后必须执行LOAD … TO RUNTIME才能加载到RUNTIME生效

执行save … to disk 才将配置持久化保存到磁盘，即保存在proxysql.db文件中

global_variables 有许多变量可以设置，其中就包括监听的端口、管理账号等

参考: https://github.com/sysown/proxysql/wiki/Global-variables

ProxySQL实现读写分离

- 向ProxySQL中添加MySQL节点，以下操作不需要use main也可成功

```
MySQL>  show tables;
MySQL > select * from sqlite_master where  name='mysql_servers'\G
MySQL > select * from  mysql_servers;
MySQL > insert into mysql_servers(hostgroup_id,hostname,port)
values(10,'192.168.8.17',3306);
MySQL > insert into mysql_servers(hostgroup_id,hostname,port)
values(10,'192.168.8.27',3306);
MySQL > load mysql servers to runtime;
MySQL > save mysql servers to disk;
```

ProxySQL实现读写分离

- 添加监控后端节点的用户。ProxySQL通过每个节点的read_only值来自动调整它们是属于读组还是写组
- 在master上执行

```
MySQL> grant replication client on *.* to monitor@'192.168.8.%'
identified by 'magedu';
```

- ProxySQL上配置监控

```
MySQL [(none)]> set mysql-monitor_username='monitor';
MySQL [(none)]> set mysql-monitor_password='magedu';
```

- 加载到RUNTIME，并保存到disk

```
MySQL [(none)]> load mysql variables to runtime;
MySQL [(none)]> save mysql variables to disk;
```

ProxySQL实现读写分离

- 监控模块的指标保存在monitor库的log表中
- 查看监控连接是否正常的 (对connect指标的监控)：
  如果connect_error的结果为NULL则表示正常
  MySQL> select * from mysql_server_connect_log;
- 查看监控心跳信息 (对ping指标的监控)：
  MySQL> select * from mysql_server_ping_log;

ProxySQL实现读写分离

- 设置分组信息

需要修改的是main库中的mysql_replication_hostgroups表，该表有3个字段：writer_hostgroup，reader_hostgroup，comment, 指定写组的id为10，读组的id为20

MySQL> insert into mysql_replication_hostgroups values(10,20,"test");

将mysql_replication_hostgroups表的修改加载到RUNTIME生效

MySQL> load mysql servers to runtime;

MySQL> save mysql servers to disk;

Monitor模块监控后端的read_only值，按照read_only的值将节点自动移动到读/写组

MySQL> select hostgroup_id,hostname,port,status,weight from mysql_servers;

```
+--------------+--------------+------+--------+--------+
| hostgroup_id | hostname     | port | status | weight |
+--------------+--------------+------+--------+--------+
| 10           | 192.168.8.17 | 3306 | ONLINE | 1      |
| 20           | 192.168.8.27 | 3306 | ONLINE | 1      |
```

ProxySQL实现读写分离

- 配置发送SQL语句的用户
- 在master节点上创建访问用户

  MySQL> grant all on *.* to sqluser@'192.168.8.%' identified by 'magedu';
- 在ProxySQL配置，将用户sqluser添加到mysql_users表中， default_hostgroup默认组设置为写组10，当读写分离的路由规则不符合时，会访问默认组的数据库

  MySQL> insert into mysql_users(username,password,default_hostgroup) values('sqluser','magedu',10);

  MySQL> load mysql users to runtime;

  MySQL> save mysql users to disk;
- 使用sqluser用户测试是否能路由到默认的10写组实现读、写数据

  mysql -usqluser -pmagedu -P6033 -h127.0.0.1 -e 'select @@server_id'

  mysql -usqluser -pmagedu -P6033 -h127.0.0.1 -e 'create database testdb'

  mysql -usqluser -pmagedu testdb -P6033 -h127.0.0.1 -e 'create table t(id int)'

ProxySQL实现读写分离

- 在proxysql上配置路由规则，实现读写分离
- 与规则有关的表：mysql_query_rules和mysql_query_rules_fast_routing，后者是前者的扩展表，1.4.7之后支持
- 插入路由规则：将select语句分离到20的读组，select语句中有一个特殊语句SELECT...FOR UPDATE它会申请写锁，应路由到10的写组

```
MySQL> insert into mysql_query_rules
(rule_id,active,match_digest,destination_hostgroup,apply)VALUES
(1,1,'^SELECT.*FOR UPDATE$',10,1),(2,1,'^SELECT',20,1);
MySQL> load mysql query rules to runtime;
MySQL> save mysql query rules to disk;
```

- 注意：因ProxySQL根据rule_id顺序进行规则匹配，select ... for update规则的rule_id必须要小于普通的select规则的rule_id

ProxySQL实现读写分离

- 测试读操作是否路由给20的读组

```
mysql -usqluser -pmagedu -P6033 -h127.0.0.1 -e 'select @@server_id'
```

- 测试写操作，以事务方式进行测试

```
mysql -usqluser -pmagedu -P6033 -h127.0.0.1 \
-e 'start transaction;select @@server_id;commit;select @@server_id'
mysql -usqluser -pmagedu -P6033 -h127.0.0.1 -e 'insert testdb.t values (1)'
mysql -usqluser -pmagedu -P6033 -h127.0.0.1 -e 'select id from  testdb.t'
```

- 路由的信息：查询stats库中的stats_mysql_query_digest表

```
MySQL > SELECT hostgroup hg,sum_time, count_star, digest_text
FROM stats_mysql_query_digest  ORDER BY sum_time DESC;
```

MySQL高可用

- MMM: Multi-Master Replication Manager for MySQL，Mysql主主复制管理器是一套灵活的脚本程序，基于perl实现，用来对mysql replication进行监控和故障迁移，并能管理mysql Master-Master复制的配置(同一时间只有一个节点是可写的)

  官网： http://www.mysql-mmm.org

  https://code.google.com/archive/p/mysql-master-master/downloads

- MHA：Master High Availability，对主节点进行监控，可实现自动故障转移至其它从节点；通过提升某一从节点为新的主节点，基于主从复制实现，还需要客户端配合实现，目前MHA主要支持一主多从的架构，要搭建MHA,要求一个复制集群中必须最少有三台数据库服务器，一主二从，即一台充当master，一台充当备用master，另外一台充当从库，出于机器成本的考虑，淘宝进行了改造，目前淘宝TMHA已经支持一主一从

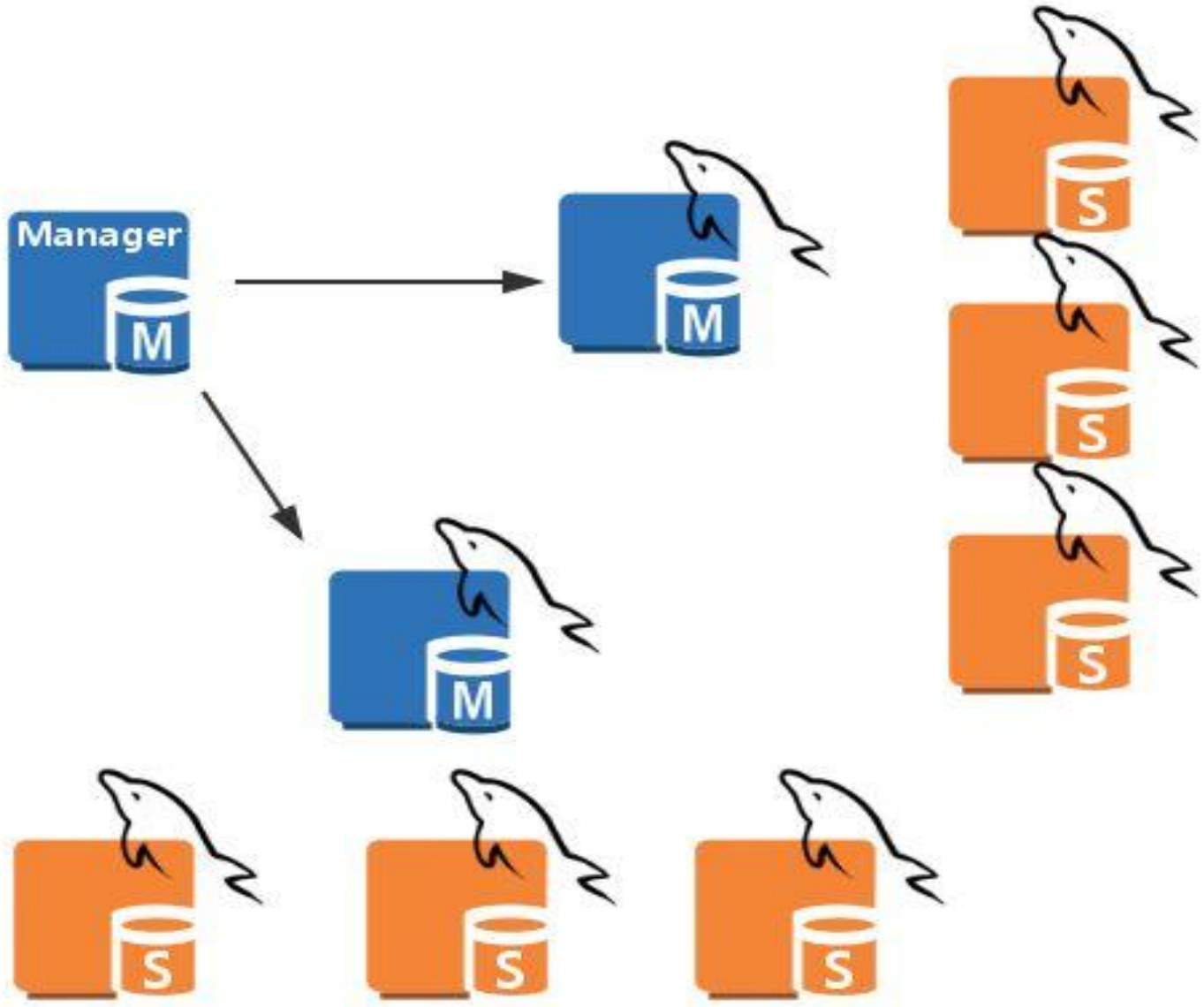
  官网:https://code.google.com/archive/p/mysql-master-ha/

- Galera Cluster：wsrep(MySQL extended with the Write Set Replication)

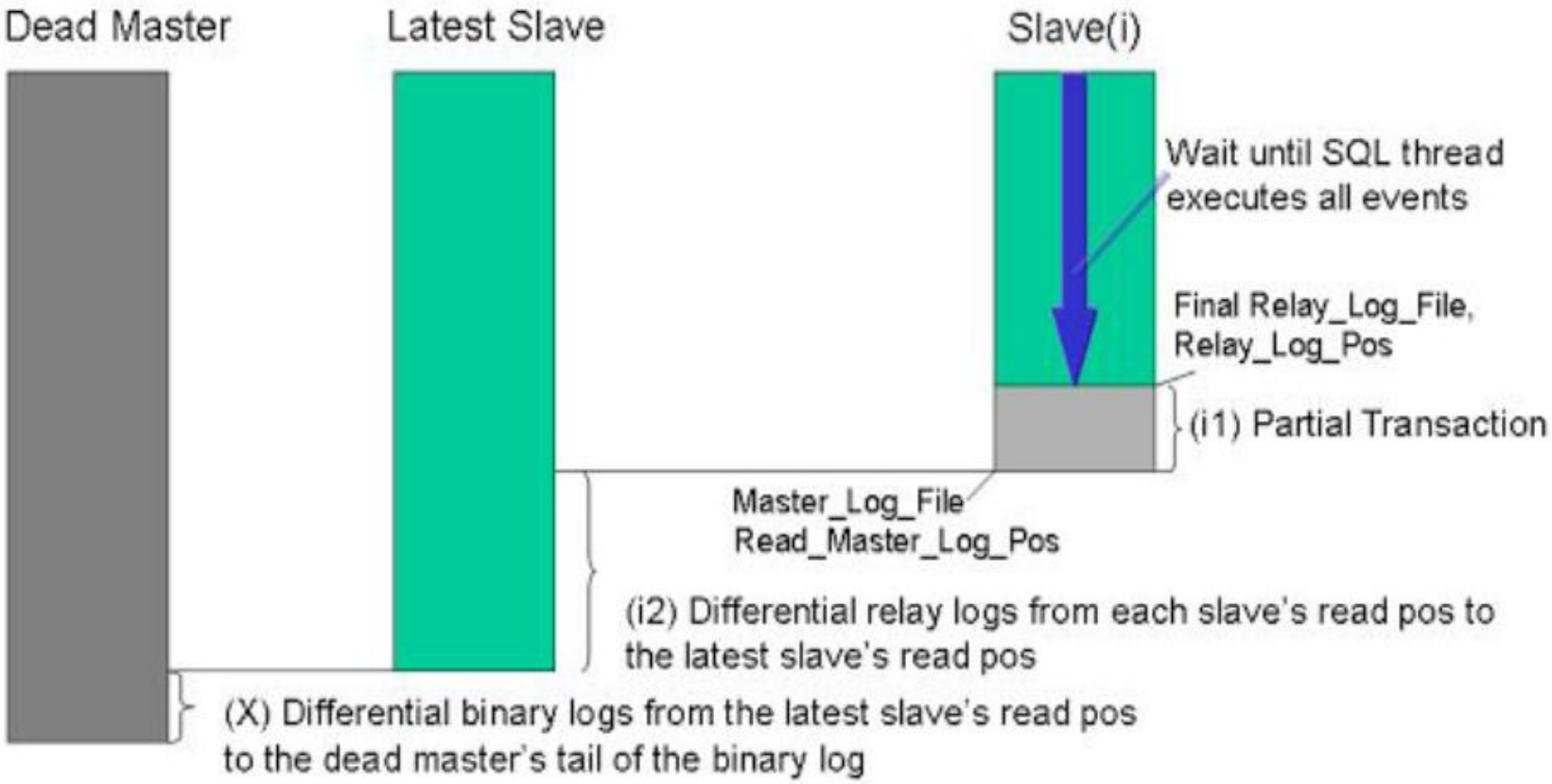
  通过wsrep协议在全局实现复制；任何一节点都可读写，不需要主从复制，实现多主读写

- GR（Group Replication）：MySQL官方提供的组复制技术(MySQL 5.7.17引入的技术)，基于原生复制技术Paxos算法，实现了多主更新，复制组由多个server成员构成，组中的每个server可独立地执行事务，但所有读写事务只在冲突检测成功后才会提交

MHA集群架构



MHA



- On slave(i),
  - Wait until the SQL thread executes events
  - Apply i1 -> i2 -> X
    - On the latest slave, i2 is empty

MHA工作原理

MHA工作原理

- 1 从宕机崩溃的master保存二进制日志事件（binlog events）
- 2 识别含有最新更新的slave
- 3 应用差异的中继日志（relay log）到其他的slave
- 4 应用从master保存的二进制日志事件（binlog events）
- 5 提升一个slave为新的master
- 6 使其他的slave连接新的master进行复制

MHA

- MHA软件由两部分组成，Manager工具包和Node工具包
- Manager工具包主要包括以下几个工具：
  - masterha_check_ssh　　检查MHA的SSH配置状况
  - masterha_check_repl　　检查MySQL复制状况
  - masterha_manger　　启动MHA
  - masterha_check_status　　检测当前MHA运行状态
  - masterha_master_monitor　　检测master是否宕机
  - masterha_master_switch　　故障转移（自动或手动）
  - masterha_conf_host　　添加或删除配置的server信息

MHA

- Node工具包：这些工具通常由MHA Manager的脚本触发，无需人为操作）主要包括以下几个工具：
  - save_binary_logs　　保存和复制master的二进制日志
  - apply_diff_relay_logs　　识别差异的中继日志事件并将其差异的事件应用于其他的slave
  - filter_mysqlbinlog　　去除不必要的ROLLBACK事件（MHA已不再使用此工具）
  - purge_relay_logs　　清除中继日志（不会阻塞SQL线程）
- 注意：为了尽可能的减少主库硬件损坏宕机造成的数据丢失，因此在配置MHA的同时建议配置成MySQL 5.5的半同步复制

MHA

- 自定义扩展：
  - secondary_check_script： 通过多条网络路由检测master的可用性
  - master_ip_ailover_script： 更新Application使用的masterip
  - shutdown_script： 强制关闭master节点
  - report_script： 发送报告
  - init_conf_load_script： 加载初始配置参数
  - master_ip_online_change_script：更新master节点ip地址
- 配置文件：

  global配置，为各application提供默认配置

  application配置：为每个主从复制集群

实现MHA

- 在管理节点上安装两个包
  mha4mysql-manager
  mha4mysql-node
- 在被管理节点安装
  mha4mysql-node

实现MHA

- 在管理节点建立配置文件

vim /etc/mastermha/app1.cnf

```
[server default]
user=mhauser
password=magedu
manager_workdir=/data/mastermha/app1/
manager_log=/data/mastermha/app1/manager.log
remote_workdir=/data/mastermha/app1/
ssh_user=root
repl_user=repluser
repl_password=magedu
ping_interval=1

[server1]
hostname=192.168.8.17
candidate_master=1
[server2]
hostname=192.168.8.27
candidate_master=1
[server3]
hostname=192.168.8.37
```

实现MHA

- 实现Master

```
vim /etc/my.cnf
[mysqld]
log-bin
server_id=1
skip_name_resolve=1

mysql>show master logs
mysql>grant replication slave on *.* to repluser@'192.168.8.%' identified
   by  ‘magedu';
mysql>grant all on *.* to mhauser@'192.168.8.%’  identified by  ‘magedu';
```

实现MHA

- 实现slave

```
vim /etc/my.cnf
[mysqld]
server_id=2      不同节点此值各不相同
log-bin
read_only
relay_log_purge=0
skip_name_resolve=1

mysql>CHANGE MASTER TO MASTER_HOST= ‘MASTER_IP',
   MASTER_USER='repluser', MASTER_PASSWORD= ‘magedu',
   MASTER_LOG_FILE='mariadb-bin.000001', MASTER_LOG_POS=245;
```

实现MHA

- 在所有节点实现相互之间ssh key验证
- Mha验证和启动

  masterha_check_ssh --conf=/etc/mastermha/app1.cnf

  masterha_check_repl --conf=/etc/mastermha/app1.cnf

  masterha_manager --conf=/etc/mastermha/app1.cnf
- 排错日志：

  /data/mastermha/app1/manager.log

Galera Cluster

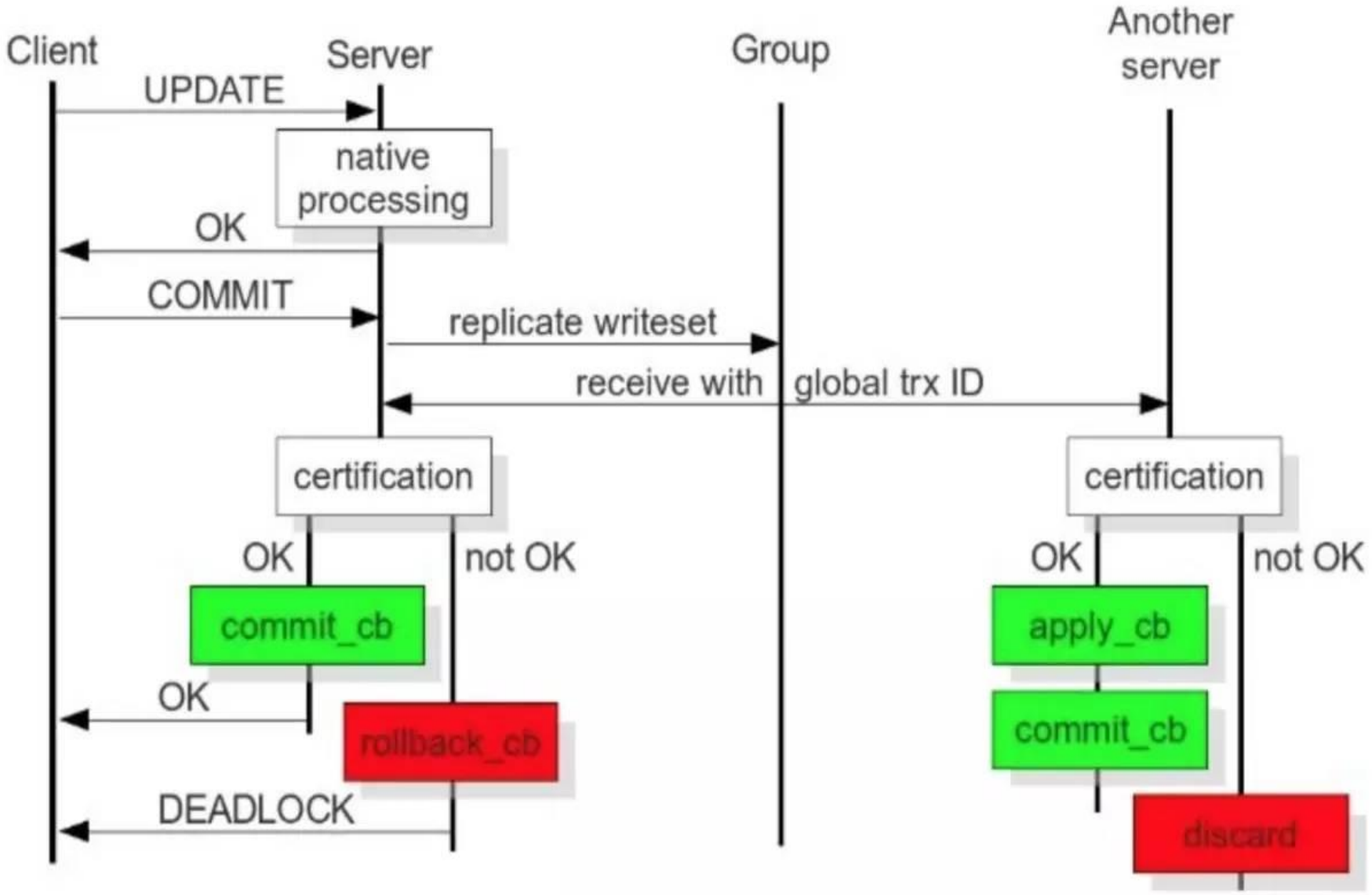
- Galera Cluster：集成了Galera插件的MySQL集群，是一种新型的，数据不共享的，高度冗余的高可用方案，目前Galera Cluster有两个版本，分别是Percona Xtradb Cluster及MariaDB Cluster，Galera本身是具有多主特性的，即采用multi-master的集群架构，是一个既稳健，又在数据一致性、完整性及高性能方面有出色表现的高可用解决方案

右图图示：三个节点组成了一个集群，与普通的主从架构不同，它们都可以作为主节点，三个节点是对等的，称为multi-master架构，当有客户端要写入或者读取数据时，连接哪个实例都是一样的，读到的数据是相同的，写入某一个节点之后，集群自己会将新数据同步到其它节点上面，这种架构不共享任何数据，是一种高冗余架构

Galera Cluster特点

- 多主架构：真正的多点读写的集群，在任何时候读写数据，都是最新的
- 同步复制：集群不同节点之间数据同步，没有延迟，在数据库挂掉之后，数据不会丢失
- 并发复制：从节点APPLY数据时，支持并行执行，更好的性能
- 故障切换：在出现数据库故障时，因支持多点写入，切换容易
- 热插拔：在服务期间，如果数据库挂了，只要监控程序发现的够快，不可服务时间就会非常少。在节点故障期间，节点本身对集群的影响非常小
- 自动节点克隆：在新增节点，或者停机维护时，增量数据或者基础数据不需要人工手动备份提供，Galera Cluster会自动拉取在线节点数据，最终集群会变为一致
- 对应用透明：集群的维护，对应用程序是透明的

Galera Cluster工作过程



Galera Cluster

- Galera Cluster官方文档：
  http://galeracluster.com/documentation-webpages/galera-documentation.pdf
  http://galeracluster.com/documentation-webpages/index.html
  https://mariadb.com/kb/en/mariadb/getting-started-with-mariadb-galera-cluster/
- Galera Cluster包括两个组件
  - Galera replication library (galera-3)
  - WSREP：MySQL extended with the Write Set Replication
- WSREP复制实现：
  - PXC：Percona XtraDB Cluster，是Percona对Galera的实现
  - MariaDB Galera Cluster

    参考仓库：https://mirrors.tuna.tsinghua.edu.cn/mariadb/mariadb-5.5.X/yum/centos7-amd64/
- 注意：都至少需要三个节点，不能安装mariadb-server

MySQL复制

- yum install MariaDB-Galera-server
- vim /etc/my.cnf.d/server.cnf

```
[galera]
wsrep_provider = /usr/lib64/galera/libgalera_smm.so
wsrep_cluster_address="gcomm://192.168.8.7,192.168.8.17,192.168.8.27"
binlog_format=row
default_storage_engine=InnoDB
innodb_autoinc_lock_mode=2
bind-address=0.0.0.0
下面配置可选项
wsrep_cluster_name = 'mycluster'  默认my_wsrep_cluster
wsrep_node_name = 'node1'
wsrep_node_address = '192.168.8.7'
```

MySQL复制

- 首次启动时，需要初始化集群，在其中一个节点上执行命令
  /etc/init.d/mysql start --wsrep-new-cluster
- 而后正常启动其它节点
  service mysql start
- 查看集群中相关系统变量和状态变量
  SHOW VARIABLES LIKE 'wsrep_%';
  SHOW STATUS LIKE 'wsrep_%';
  SHOW STATUS LIKE 'wsrep_cluster_size';

复制的问题和解决方案

- 复制的问题和解决方案：
  (1) 数据损坏或丢失
  Master： MHA + semi repl
  Slave： 重新复制
  (2) 混合使用存储引擎
  MyISAM：不支持事务
  InnoDB： 支持事务
  (3) 不惟一的server id
  重新复制
  (4) 复制延迟
  需要额外的监控工具的辅助
  一从多主：mariadb10版后支持
  多线程复制：对多个数据库复制

TiDB概述

- TiDB 是 PingCAP 公司受 Google Spanner / F1 论文启发而设计的开源分布式 HTAP (Hybrid Transactional and Analytical Processing) 数据库，结合了传统的 RDBMS 和NoSQL 的最佳特性。TiDB 兼容 MySQL，支持无限的水平扩展，具备强一致性和高可用性。 tidb和mysql几乎完全兼容
- TiDB 是一个分布式 NewSQL 数据库。它支持水平弹性扩展、ACID 事务、标准 SQL、MySQL 语法和 MySQL 协议，具有数据强一致的高可用特性，是一个不仅适合 OLTP 场景还适合 OLAP 场景的混合数据库。
- TiDB 的目标是为 OLTP(Online Transactional Processing) 和 OLAP (Online Analytical Processing) 场景提供一站式的解决方案。

TiDB 具备如下核心特点

- 1 高度兼容 MySQL　大多数情况下，无需修改代码即可从 MySQL 轻松迁移至 TiDB，分库分表后的 MySQL 集群亦可通过 TiDB 工具进行实时迁移
- 2水平弹性扩展　通过简单地增加新节点即可实现 TiDB 的水平扩展，按需扩展吞吐或存储，轻松应对高并发、海量数据场景。
- 3分布式事务　TiDB 100% 支持标准的 ACID 事务
- 4 真正金融级高可用　相比于传统主从 (M-S) 复制方案，基于 Raft 的多数派选举协议可以提供金融级的 100% 数据强一致性保证，且在不丢失大多数副本的前提下，可以实现故障的自动恢复 (auto-failover)，无需人工介入。
- 5 一站式 HTAP 解决方案　TiDB 作为典型的 OLTP 行存数据库，同时兼具强大的 OLAP 性能，配合 TiSpark，可提供一站式 HTAP解决方案，一份存储同时处理OLTP & OLAP(OLAP、OLTP的介绍和比较)无需传统繁琐的 ETL 过程。
- 6云原生 SQL 数据库　TiDB 是为云而设计的数据库，同 Kubernetes （十分钟带你理解Kubernetes核心概念 ）深度耦合，支持公有云、私有云和混合云，使部署、配置和维护变得十分简单。　TiDB 的设计目标是 100% 的 OLTP 场景和 80% 的 OLAP 场景，更复杂的 OLAP 分析可以通过 TiSpark 项目来完成。 TiDB 对业务没有任何侵入性，能优雅的替换传统的数据库中间件、数据库分库分表等 Sharding 方案。同时它也让开发运维人员不用关注数据库 Scale 的细节问题，专注于业务开发，极大的提升研发的生产力.

性能衡量指标

- 数据库服务衡量指标：
  - Qps：query per second
  - Tps：transaction per second
- 压力测试工具：
  - mysqlslap
  - Sysbench：功能强大
    - https://github.com/akopytov/sysbench
  - tpcc-mysql
  - MySQL Benchmark Suite
  - MySQL super-smack
  - MyBench

MYSQL压力测试

- Mysqlslap：来自于mariadb包，测试的过程默认生成一个mysqlslap的schema,生成测试表t1，查询和插入测试数据，mysqlslap库自动生成，如果已经存在则先删除。用--only-print来打印实际的测试过程，整个测试完成后不会在数据库中留下痕迹
- 使用格式：mysqlslap [options]
- 常用参数 [options] 说明：
  - --auto-generate-sql, -a 自动生成测试表和数据，表示用mysqlslap工具自己生成的SQL脚本来测试并发压力
  - --auto-generate-sql-load-type=type 测试语句的类型。代表要测试的环境是读操作还是写操作还是两者混合的。取值包括：read，key，write，update和mixed(默认)

MYSQL压力测试

- --auto-generate-sql-add-auto-increment 代表对生成的表自动添加auto_increment列，从5.1.18版本开始支持
- --number-char-cols=N, -x N 自动生成的测试表中包含多少个字符类型的列，默认1
- --number-int-cols=N, -y N 自动生成的测试表中包含多少个数字类型的列，默认1
- --number-of-queries=N 总的测试查询次数(并发客户数×每客户查询次数)
- --query=name,-q 使用自定义脚本执行测试，例如可以调用自定义的存储过程或者sql语句来执行测试
- --create-schema 代表自定义的测试库名称，测试的schema
- --commint=N 多少条DML后提交一次

MYSQL压力测试

- --compress, -C 如服务器和客户端都支持压缩，则压缩信息
- --concurrency=N, -c N 表示并发量，即模拟多少个客户端同时执行select。可指定多个值，以逗号或者--delimiter参数指定值做为分隔符

  如：--concurrency=100,200,500
- --engine=engine_name, -e engine_name 代表要测试的引擎，可以有多个，用分隔符隔开。例如：--engines=myisam,innodb
- --iterations=N, -i N 测试执行的迭代次数，代表要在不同并发环境下，各自运行测试多少次
- --only-print 只打印测试语句而不实际执行。
- --detach=N 执行N条语句后断开重连
- --debug-info, -T 打印内存和CPU的相关信息

mysqlslap示例

- 单线程测试

  mysqlslap -a -uroot -pmagedu

- 多线程测试。使用--concurrency来模拟并发连接

  mysqlslap -a -c 100 -uroot -pmagedu

- 迭代测试。用于需要多次执行测试得到平均值

  mysqlslap -a -i 10 -uroot -pmagedu

  mysqlslap ---auto-generate-sql-add-autoincrement -a

  mysqlslap -a --auto-generate-sql-load-type=read

  mysqlslap -a --auto-generate-secondary-indexes=3

  mysqlslap -a --auto-generate-sql-write-number=1000

  mysqlslap --create-schema world -q "select count(*) from City”

  mysqlslap -a -e innodb -uroot -pmagedu

  mysqlslap -a --number-of-queries=10 -uroot -pmagedu

mysqlslap示例

- 测试同时不同的存储引擎的性能进行对比

  mysqlslap -a --concurrency=50,100 --number-of-queries 1000 --iterations=5 --engine=myisam,innodb --debug-info -uroot -pmagedu

- 执行一次测试，分别50和100个并发，执行1000次总查询

  mysqlslap -a --concurrency=50,100 --number-of-queries 1000 --debug-info -uroot -pmagedu

- 50和100个并发分别得到一次测试结果(Benchmark)，并发数越多，执行完所有查询的时间越长。为了准确起见，可以多迭代测试几次

  mysqlslap -a --concurrency=50,100 --number-of-queries 1000 --iterations=5 --debug-info -uroot -pmagedu

生产环境 my.cnf 配置示例

◆ 硬件：内存32G

innodb_file_per_table = 1

打开独立表空间

max_connections = 8000

#MySQL 服务所允许的同时会话数的上限，经常出现Too Many Connections的错误提示，则需要增大此值

open_files_limit = 10240

#所有线程所打开表的数量

back_log = 300

#back_log 是操作系统在监听队列中所能保持的连接数

max_connect_errors = 1000

#每个客户端连接最大的错误允许数量，当超过该次数，MYSQL服务器将禁止此主机的连接请求，直到MYSQL服务器重启或通过flush hosts命令清空此主机的相关信息

生产环境 my.cnf 配置示例


```
max_allowed_packet = 32M
#每个连接传输数据大小.最大1G，须是1024的倍数，一般设为最大的BLOB的值
wait_timeout = 10
#指定一个请求的最大连接时间
sort_buffer_size = 16M
# 排序缓冲被用来处理类似ORDER BY以及GROUP BY队列所引起的排序
join_buffer_size = 16M
#不带索引的全表扫描.使用的buffer的最小值
query_cache_size = 128M
#查询缓冲大小
query_cache_limit = 4M
#指定单个查询能够使用的缓冲区大小，缺省为1M
```

生产环境 my.cnf 配置示例

transaction_isolation = REPEATABLE-READ

设定默认的事务隔离级别

thread_stack = 512K

线程使用的堆大小. 此值限制内存中能处理的存储过程的递归深度和SQL语句复杂性，此容量的内存在每次连接时被预留.

log-bin

二进制日志功能

binlog_format=row

#二进制日志格式

innodb_buffer_pool_size = 24G

#InnoDB使用一个缓冲池来保存索引和原始数据, 可设置这个变量到物理内存大小的80%

innodb_file_io_threads = 4

#用来同步IO操作的IO线程的数量

生产环境 my.cnf 配置示例


```
innodb_thread_concurrency = 16
#在InnoDb核心内的允许线程数量，建议的设置是CPU数量加上磁盘数量的两倍
innodb_log_buffer_size = 16M
# 用来缓冲日志数据的缓冲区的大小
innodb_log_file_size = 512M
在日志组中每个日志文件的大小
innodb_log_files_in_group = 3
# 在日志组中的文件总数
innodb_lock_wait_timeout = 120
# SQL语句在被回滚前,InnoDB事务等待InnoDB行锁的时间
long_query_time = 2
#慢查询时长
log-queries-not-using-indexes
#将没有使用索引的查询也记录下来
```

MYSQL配置最佳实践

- 高并发大数据的互联网业务，架构设计思路是“解放数据库CPU，将计算转移到服务层”，并发量大的情况下，这些功能很可能将数据库拖死，业务逻辑放到服务层具备更好的扩展性，能够轻易实现“增机器就加性能”
- 参考：

  阿里巴巴Java开发手册

  58到家数据库30条军规解读

  http://zhuanlan.51cto.com/art/201702/531364.htm

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