尚观科技云计算课程

DB100_ 开源数据库云端综合部署实战

https://github.com/BoobooWei/DB100_mysql

DB100_开源数据库课程简介

- MySQL

- Redis

redis

- MongoDB

mongoDB®

1 MySQL

MYSQL

- MySQL 管理课程简介 day1-1 学时
- MySQL 和 MariaDB 数据库介绍 day1-3 学时
- 结构化查询语言 SQL 介绍和基本操作 day1-1 学时 ~day2-3 学时
- MySQL 逻辑架构和 Innodb 存储引擎 day2-2 学时
- MySQL 备份与恢复 day3~day4-10 学时
- MySQL 复制 replication day5~day6 -10 学时
- MySQL 高可用 HA day7-5 学时

MariaDB

PERCONA

1.1 MySQL 管理课程简介

MySQL 管理课程简介

1. 为什么 MySQL ？

2. 课程环境使用说明

为什么 MySQL ？

- 2017 年 2 月份 DB-Engines 的数据库排名
- 2017 年 1 月份 中国数据库排行榜
- 51job 上 MySQL DBA 的职位提供

Rank			DBMS	Database Model	Score		
Feb 2017	Jan 2017	Feb 2016			Feb 2017	Jan 2017	Feb 2016
1.	1.	1.	Oracle	Relational DBMS	1403.83	-12.89	-72.31
2.	2.	2.	MySQL	Relational DBMS	1380.30	+14.02	+59.18
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1203.45	-17.50	+53.23
4.	↑ 5.	↑ 5.	PostgreSQL	Relational DBMS	353.68	+23.31	+65.02
5.	↓ 4.	↓ 4.	MongoDB	Document store	335.50	+3.60	+29.90
6.	6.	6.	DB2	Relational DBMS	187.90	+5.41	-6.58
7.	7.	↑ 8.	Cassandra	Wide column store	134.38	-2.06	+2.62
8.	8.	↓ 7.	Microsoft Access	Relational DBMS	133.39	+5.94	+0.31
9.	↑ 10.	9.	SQLite	Relational DBMS	115.31	+2.93	+8.53
10.	↓ 9.	10.	Redis	Key-value store	114.03	-4.66	+11.96
11.	11.	↑ 12.	Elasticsearch	Search engine	108.31	+2.14	+30.47
12.	12.	↑ 13.	Teradata	Relational DBMS	75.60	+1.43	+2.22
13.	13.	↓ 11.	SAP Adaptive Server	Relational DBMS	71.74	+2.63	-8.30
14.	14.	14.	Solr	Search engine	67.69	-0.39	-4.59
15.	15.	↑ 16.	HBase	Wide column store	59.24	+0.10	+7.22
16.	16.	↑ 18.	Splunk	Search engine	56.03	+0.54	+13.20
17.	17.	17.	FileMaker	Relational DBMS	55.19	+1.71	+8.16
18.	18.	↑ 19.	SAP HANA	Relational DBMS	52.45	+0.52	+14.37
19.	19.	↓ 15.	Hive	Relational DBMS	47.95	-3.19	-4.83
20.	20.	↑ 23.	MariaDB	Relational DBMS	45.35	+0.31	+16.57

- 2017 年 2 月份 DB-Engines 的数据库排名

- 2017 年 2 月份 DB-Engines 的数据库排名

- 2017 年 1 月份 中国数据库排行榜

- 51JOB 上 MySQL DBA 的职位提供

DBA（MySQL） 上海-浦东新区 **1.5-2.5万/月**

新达达 该公司所有职位

职位描述：

工作职责：

1、负责MySQLNoSQL系统的维护、性能监控和调优；

2、制定安全合理的备份策略，保证数据安全；

3、数据库相关管理系统与自动化工具的研发；

4、参与业务项目设计并提供相应的数据库建议；

5、能够及时处理生产系统突发事件。

任职资格：

1、具备2年及以上数据库相关系统的维护经验；

2、熟练掌握MySQL及InnoDB的体系结构和运行原理；

3、熟练掌握MySQL Replication的运行体系和原理以及数据备份恢复技术；熟悉MySQL高可用技术；

4、熟练掌握Shell脚本，熟悉Awk、Sed等基础工具；

5、熟悉AnsibleSaltStack自动化配置管理工具

6、熟悉并能使用PythonPHPPerl中的一种编程语言；

7、熟悉分布式计算或者存储系统，如HadoopHbaseStorm等，有一定维护经验。

职能类别：数据库工程师/管理员

关键字： Mysql DBA

- 51JOB 上 MySQL DBA 的职位提供

使用 MySQL 考虑的几个关注点：

Q ：它解决了我们的存储需求吗？
A ：没错，我们需要映射、索引、排序和 blob 存储，这些 MySQL 都有。

Q ：它常用吗？ 你可以招聘到相关员工吗？
A ： MySQL 是目前生产线上最常使用的数据库之一。很容易招到使用过 MySQL 的人

Q ：它的社区活跃吗？
A ：非常活跃。有好多非常棒的书籍，和一个强大的在线社区。

Q ：面对故障，它健壮吗？
A ：即使在最恶劣的情况下，我们也从来没有丢失过数据。

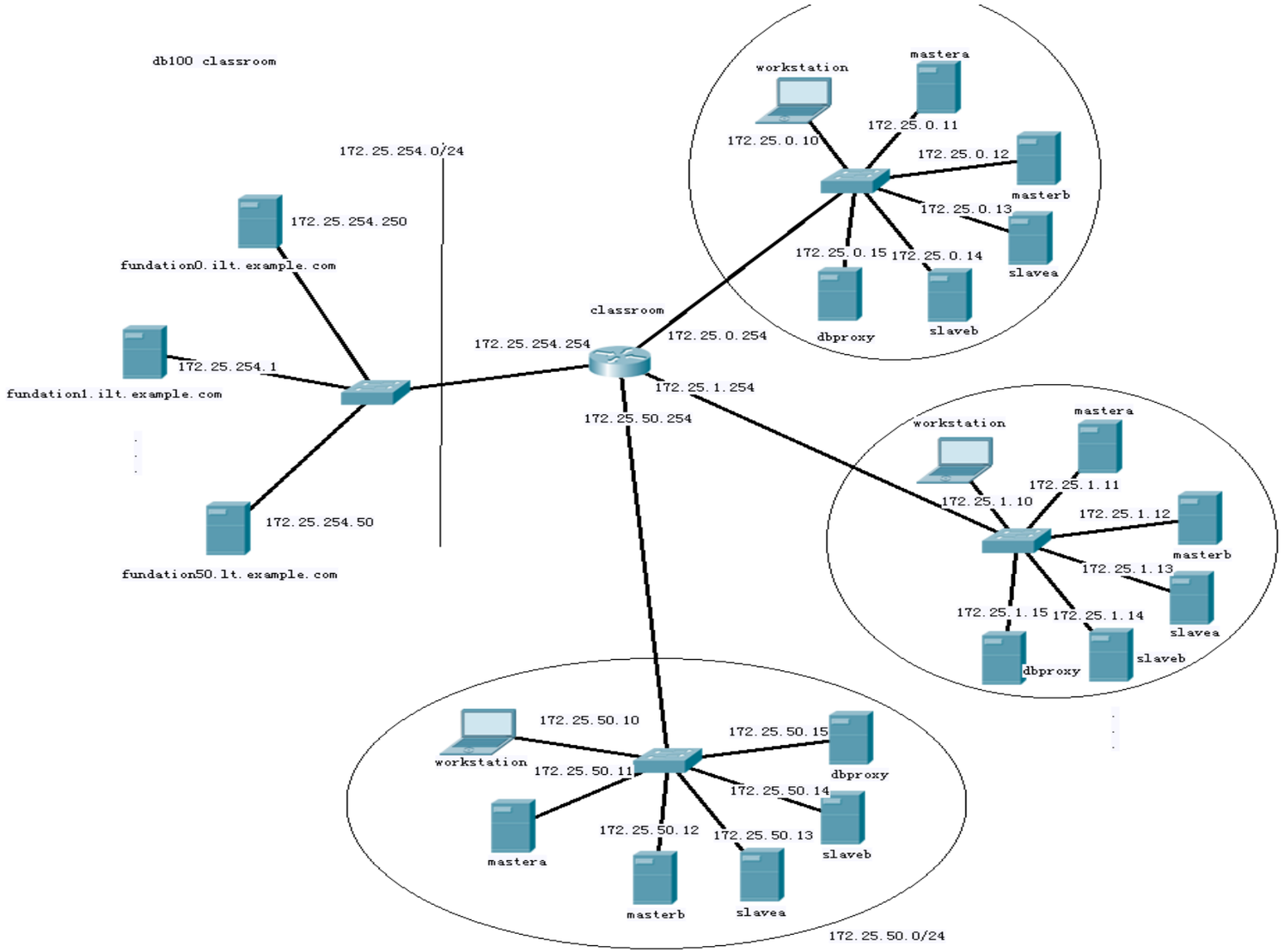
Q ：它的扩展性如何？
A ：就它本身来说，只是一个很小的组件。我们需要一种上层的分片方案（这完全是另一个问题）。

Q ：你会是最大的用户吗？
A ：不，目前不是。最大的用户包括 Facebook、 Twitter 和 Google。除非你能够改进一种技术，否则你不会想要成为它最大的用户。如果你是最大的用户，你会碰到一些新的扩展性问题，而其他人根本没机会遇到。

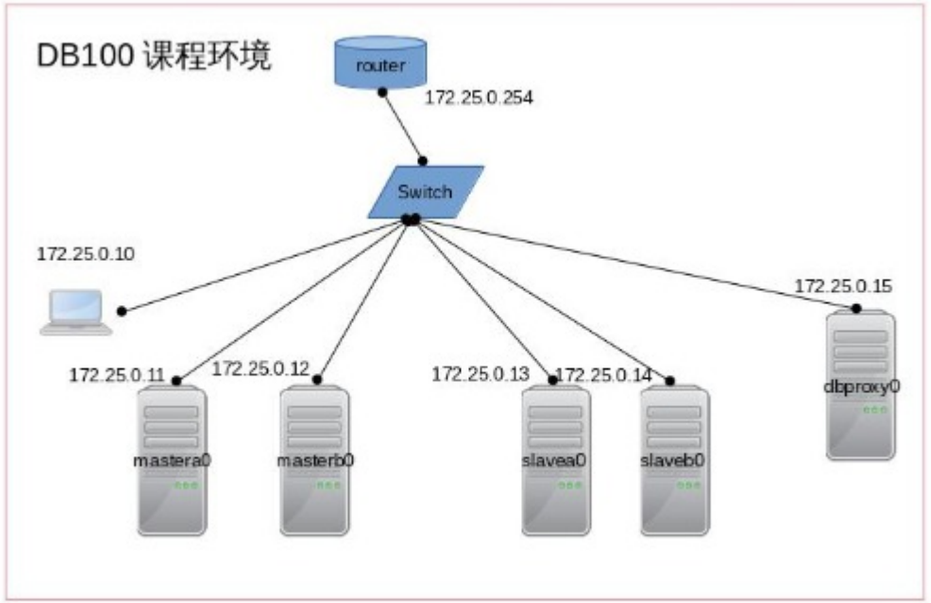
Q ：它的成熟度如何？
A ：真正的区别在于成熟度。根据复杂度的不同，成熟度就好比衡量完成一个程序所需的血、汗和泪。 MySQL 的确复杂，但可比不上那些神奇的自动集群 NoSQL 方案。而且，MySQL 拥有 30 年（ 1996 年第一版）最好和最聪明的贡献，来自于诸如 Facebook 和 Google 那样大规模使用它的公司。根据我们的成熟度定义，在我们审查的所有技术中，MySQL 是一个明智的选择。

Q ：有好的调试工具吗？
A ：作为一个成熟的产品，你当然需要强大的调试和分析工具，因为人们很容易遇到一些类似的棘手情况。比如你可能在凌晨三点遇到问题（不止一次）。相比用另一种技术重写一遍熬到凌晨六点，发现问题的根源然后回去睡觉舒服多了。

MySQL 管理课程环境使用说明



MySQL 管理课程环境使用说明



1.2 MySQL 和 MariaDB 数据库介绍

MySQL 和 MariaDB 数据库介绍

- 数据库的基础概念
- 数据库的分类
- MySQL 的概念
- 如何获取 MySQL 相关资源
- MySQL 在企业中的应用场景
- MySQL 的安装

数据库的基础概念

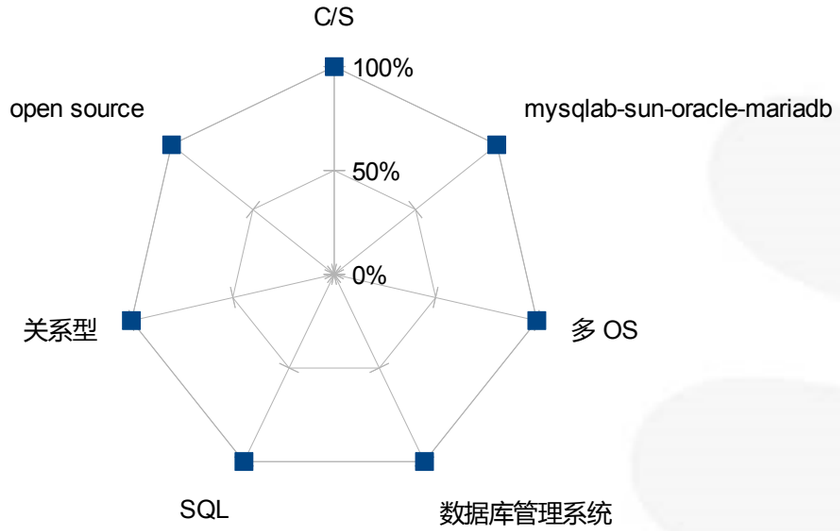
数据库 (database): 保存有组织的数据的容器 (通常是一个文件或一组文件) 。

数据库管理系统 (DBMS): 顾名思义，既然是数据管理系统，就是用来管理数据的数据库软件

数据库的分类

数据库类型	描述	主流产品	有谁在用
关系型数据库	关系型数据库就是由二维表及其之间的联系组成的一个数据组织	Oracle、MySQL、MariaDB、Microsoft SQL Server	alibaba,sina,网易，youtube，google等
键值数据库	键值数据库就像在传统语言中使用的哈希表。你可以通过 key 来添加、查询或者删除数据，鉴于使用主键访问，所以会获得不错的性能及扩展性。	Riak、Redis、Memcached、Amazon's Dynamo、Project Voldemort	GitHub（Riak）Twitter（Redis和Memcached）、StackOverFlow（Redis）、 Youtube（Memcached）
面向文档数据库	面向文档数据库会将数据以文档的形式储存。每个文档都是自包含的数据单元，是一系列数据项的集合。每个数据项都有一个名称与对应的值，值既可以是简单的数据类型，如字符串、数字和日期等；也可以是复杂的类型，如有序列表和关联对象。数据存储的最小单位是文档，同一个表中存储的文档属性可以是不同的，数据可以使用 XML、JSON 或者 JSONB 等多种形式存储。	MongoDB、CouchDB、RavenDB	SAP（MongoDB）、Codecademy（MongoDB）、Foursquare（MongoDB）、NBC News（RavenDB）
列存储数据库	列存储数据库将数据储存在列族（column family）中，一个列族存储经常被一起查询的相关数据。举个例子，如果我们有一个 Person 类，我们通常会一起查询他们的姓名和年龄而不是薪资。这种情况下，姓名和年龄就会被放入一个列族中，而薪资则在另一个列族中。	Cassandra、HBase	Ebay（Cassandra）、Instagram（Cassandra）、NASA（Cassandra）、Twitter（Cassandra and HBase）、Facebook（HBase）、Yahoo!（HBase）
图数据库	图数据库允许我们将数据以图的方式储存。实体会被作为顶点，而实体之间的关系则会被作为边。比如我们有三个实体，Steve Jobs、Apple 和 Next，则会有两个“Founded by”的边将 Apple 和 Next 连接到 Steve Jobs。	Neo4J、Infinite Graph、OrientDB	Adobe（Neo4J）、Cisco（Neo4J）、T-Mobile（Neo4J）

MySQL 的概念



什么是 mysql ？摘自官方的一个解释：

“MySQL 是采用**客户 / 服务器模型**的**开放源码 关系型 SQL 数据库管理系统**，它 可以在**多种操作系统**上运行，它是由 **MySQL Ab** 公司开发、发布并支持的，后被 **sun** 收购，现在在 **oracle** 公司旗下，现在有一个知名的分支 **MariaDB** 。”

如何获取 MySQL 相关资源

MySQL 在企业中的应用场景

MySQL 数据库安装

- 学会 MariaDB 5.5 的安装
- 学会 MariaDB 10.2 的安装
- 学会 MySQL 5.6 的安装
- 学会 MySQL 5.7 的安装
- MySQL 自动安装脚本

MySQL 客户端连接数据库

- 通过 mysql 客户端连接
- 通过 python 程序连接
- 通过 phpMyAdmin 连接
- 通过 MySQL Workbench 连接

1.3 结构化查询语言 SQL 介绍和基本操作

SQL 是 Structure Query Language(结构化查询语言) 的缩写 , 它是使用关系模型的数据库应用语言 , 由 IBM 在 20 世纪 70 年代开发出来 , 作为 IBM 关系数据库原型 System R 的原型关系语言 , 实现了关系数据库中的信息检索。

20 世纪 80 年代初 , 美国国家标准局 (ANSI) 开始着手制定 SQL 标准 , 最早的 ANSI 标准于 1986 年完成 , 就被叫作 SQL-86 。标准的出台使 SQL 作为标准关系数据库语言的地位得到了加强。 SQL 标准目前已几经修改更趋完善。

正是由于 SQL 语言的标准化 , 所以大多数关系型数据库系统都支持 SQL 语言 , 它已经发展成为多种平台进行交互操作的底层会话语言。

SQL 分类

DDL(Data Definition Languages) 语句

DML(Data Manipulation Language) 语句

DCL(Data Control Language) 语句

DQL(Data Query Language) 语句

MySQL 中的语法格式

1. SQL 语句是由简单的英语单词构成的。这些单词称为关键字 , 每个 SQL 语句都是由一个或多个关键字构成的。

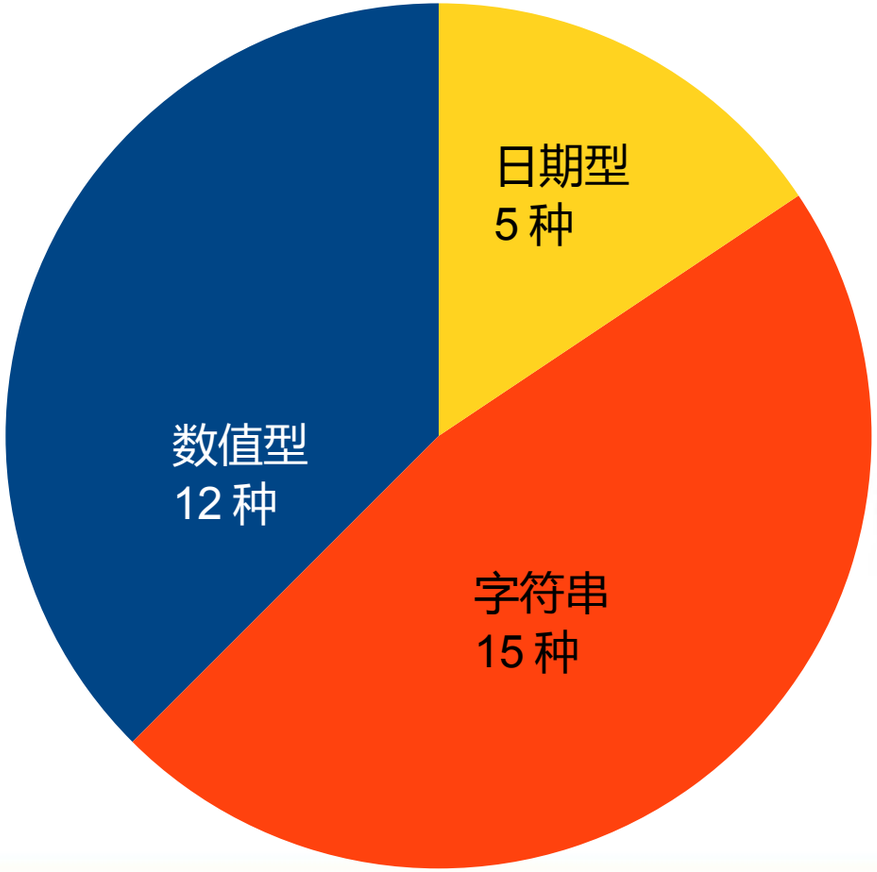
2. 结束 SQL 语句 多条 SQL 语句必须以分号 (;) 分隔。

3. SQL 语句不区分大小写。

4. SQL 语句可以在一行上给出 , 也可以分成许多行。多数 SQL 开发人员认为将 SQL 语句分成多行更容易阅读和调试。

MySQL 数据类型

整数	6
定点数	2
浮点数	3
bit	1



date,datetime,timestamp	3
time	1
year	1

数值
字符串
日期

char 和 varchar	2
binary 和 varbinary	2
blob 和 text	8
enum	1
set	1
json	1

MySQL 数据类型——数字

BIT数据类型可用来保存位字段值。BIT(M)类型允许存储M位值。M范围为1~64，默认为1。
BIT其实就是存入二进制的值，类似010110。如果存入一个BIT类型的值，位数少于M值，则左补0.

类型	大小	范围（有符号）	范围（无符号）	用途
TINYINT	1 字节	(-128，127)	(0，255)	小整数值
SMALLINT	2 字节	(-32 768，32 767)	(0，65 535)	大整数值
MEDIUMINT	3 字节	(-8 388 608，8 388 607)	(0，16 777 215)	大整数值
INT或INTEGER	4 字节	(-2 147 483 648，2 147 483 647)	(0，4 294 967 295)	大整数值
BIGINT	8 字节	(-9 233 372 036 854 775 808，9 223 372 036 854 775 807)	(0，18 446 744 073 709 551 615)	极大整数值
FLOAT	4 字节	(-3.402 823 466 E+38，-1.175 494 351 E-38)，0，(1.175 494 351 E-38，3.402 823 466 351 E+38)	0，(1.175 494 351 E-38，3.402 823 466 E+38)	单精度浮点数值
DOUBLE	8 字节	(-1.797 693 134 862 315 7 E+308，-2.225 073 858 507 201 4 E-308)，0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	双精度浮点数值
DECIMAL	对DECIMAL(M,D)，如果M>D，为M+2否则为D+2	依赖于M和D的值	依赖于M和D的值	小数值

REAL就是DOUBLE，定义方式为：FLOAT(M,D) 、 REAL(M,D) 、 DOUBLE PRECISION(M,D)。

定点数
DECIMAL和NUMERIC类型在MySQL中视为相同的类型。它们用于保存必须为确切精度的值。

MySQL 数据类型——字符串

CHAR 和 VARCHAR 类型类似，但它们保存和检索的方式不同。它们的最大长度和是否尾部空格被保留等方面也不同。在存储或检索过程中不进行大小写转换。
BINARY 和 VARBINARY 类类似于 CHAR 和 VARCHAR ，不同的是它们包含二进制字符串而不要非二进制字符串。也就是说，它们包含字节字符串而不是字符字符串。这说明它们没有字符集，并且排序和比较基于列值字节的数值值。
BLOB 是一个二进制大对象，可以容纳可变数量的数据。有 4 种 BLOB 类型： TINYBLOB 、 BLOB 、 MEDIUMBLOB 和 LONGBLOB 。它们只是可容纳值的最大长度不同。
有 4 种 TEXT 类型： TINYTEXT 、 TEXT 、 MEDIUMTEXT 和 LONGTEXT 。这些对应 4 种 BLOB 类型，有相同的最大长度和存储需求。

类型	大小	用途
CHAR	0-255字节	定长字符串
VARCHAR	0-65535 字节	变长字符串
TINYBLOB	0-255字节	不超过 255 个字符的二进制字符串
TINYTEXT	0-255字节	短文本字符串
BLOB	0-65 535字节	二进制形式的长文本数据
TEXT	0-65 535字节	长文本数据
MEDIUMBLOB	0-16 777 215字节	二进制形式的中等长度文本数据
MEDIUMTEXT	0-16 777 215字节	中等长度文本数据
LONGBLOB	0-4 294 967 295字节	二进制形式的极大文本数据
LONGTEXT	0-4 294 967 295字节	极大文本数据

MySQL 数据类型——字符串

MySQL 5.7.7 labs 版本开始 InnoDB 存储引擎已经原生支持 JSON 格式，该格式不是简单的 BLOB 类似的替换。原生的 JSON 格式支持有以下的优势：

- JSON 数据有效性检查： BLOB 类型无法在数据库层做这样的约束性检查
- 查询性能的提升：查询不需要遍历所有字符串才能找到数据
- 支持索引：通过虚拟列的功能可以对 JSON 中的部分数据进行索引

MySQL 数据类型——日期

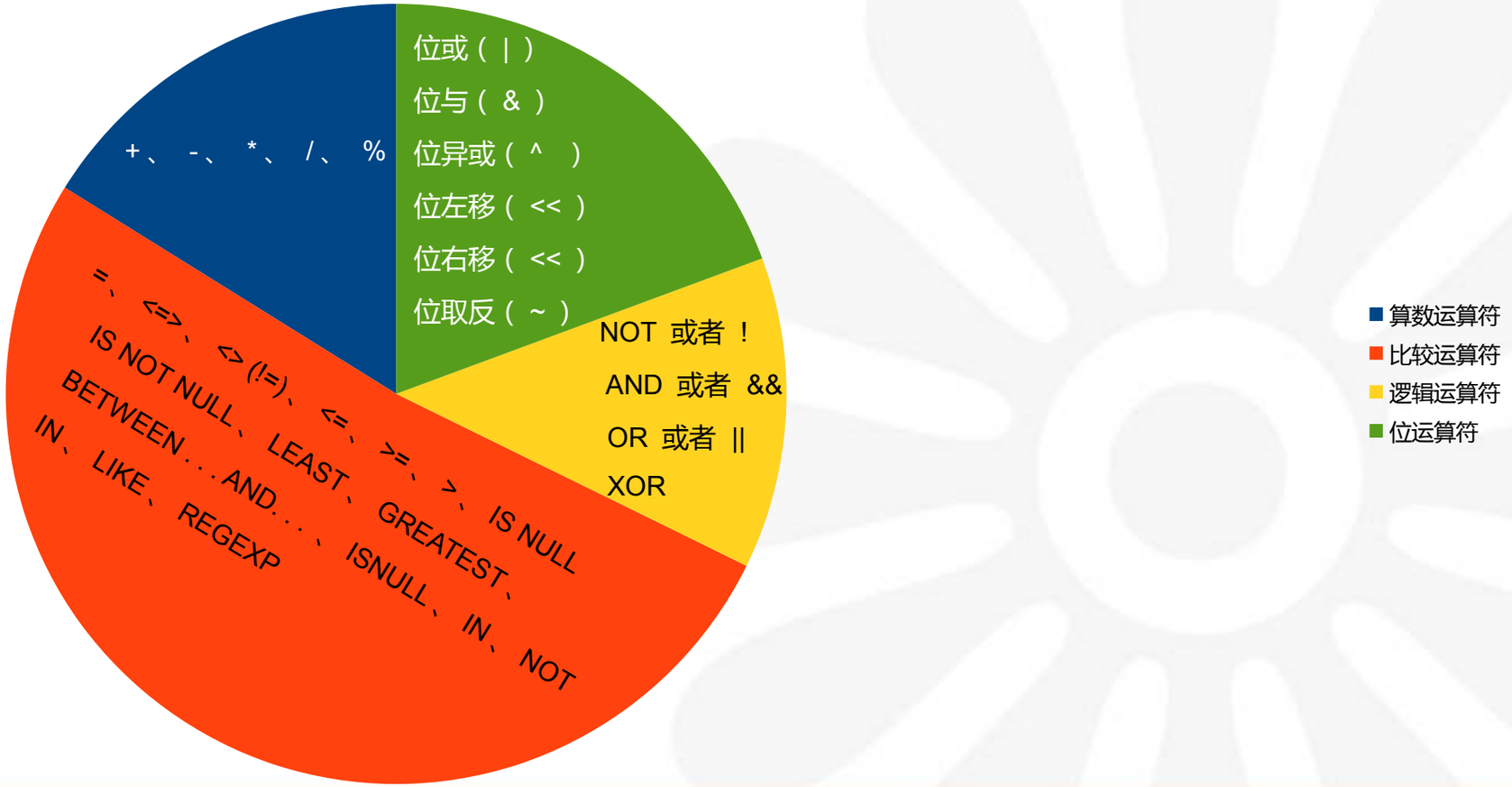
表示时间值的日期和时间类型为DATETIME、DATE、TIMESTAMP、TIME和YEAR。

每个时间类型有一个有效值范围和一个"零"值，当指定不合法的MySQL不能表示的值时使用"零"值。

TIMESTAMP类型有专有的自动更新特性，将在后面描述。

类型	大小(字节)	范围	格式	用途
DATE	3	1000-01-01/9999-12-31	YYYY-MM-DD	日期值
TIME	3	'-838:59:59'/'838:59:59'	HH:MM:SS	时间值或持续时间
YEAR	1	1901/2155	YYYY	年份值
DATETIME	8	1000-01-01 00:00:00/9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	混合日期和时间值
TIMESTAMP	4	1970-01-01 00:00:00/2037 年某时	YYYYMMDD HHMMSS	混合日期和时间值，时间戳

MySQL 运算符


+、 -、 *、 /、 %
位或（ | ）
位与（ & ）
位异或（ ^ ）
位左移（ << ）
位右移（ << ）
位取反（ ~ ）
NOT 或者 ！
AND 或者 &&
OR 或者 ||
XOR
=、 <=>、 <> (!=)、 <=、 >=、 >、 IS NULL
IS NOT NULL、 LEAST、 GREATEST、
BETWEEN . . . AND. . . 、 ISNULL、 IN、 NOT
IN、 LIKE、 REGEXP
算数运算符
比较运算符
逻辑运算符
位运算符

比较运算符

- 一个比较运算符的结果总是 1,0 或者是 NULL 。

  = 、 <=> 、 <> (!=) 、 <= 、 >= 、 > 、 IS NULL IS NOT NULL 、 LEAST 、 GREATEST 、 BETWEEN . . . AND. . . 、 ISNULL 、 IN 、 NOT IN 、 LIKE 、 REGEXP

逻辑运算符

- 逻辑运算符的求值所得结果均为 TRUE 、 FALSE 或 NULL 。

  NOT 或者 ！

  AND 或者 &&

  OR 或者 ||

  XOR

位运算符

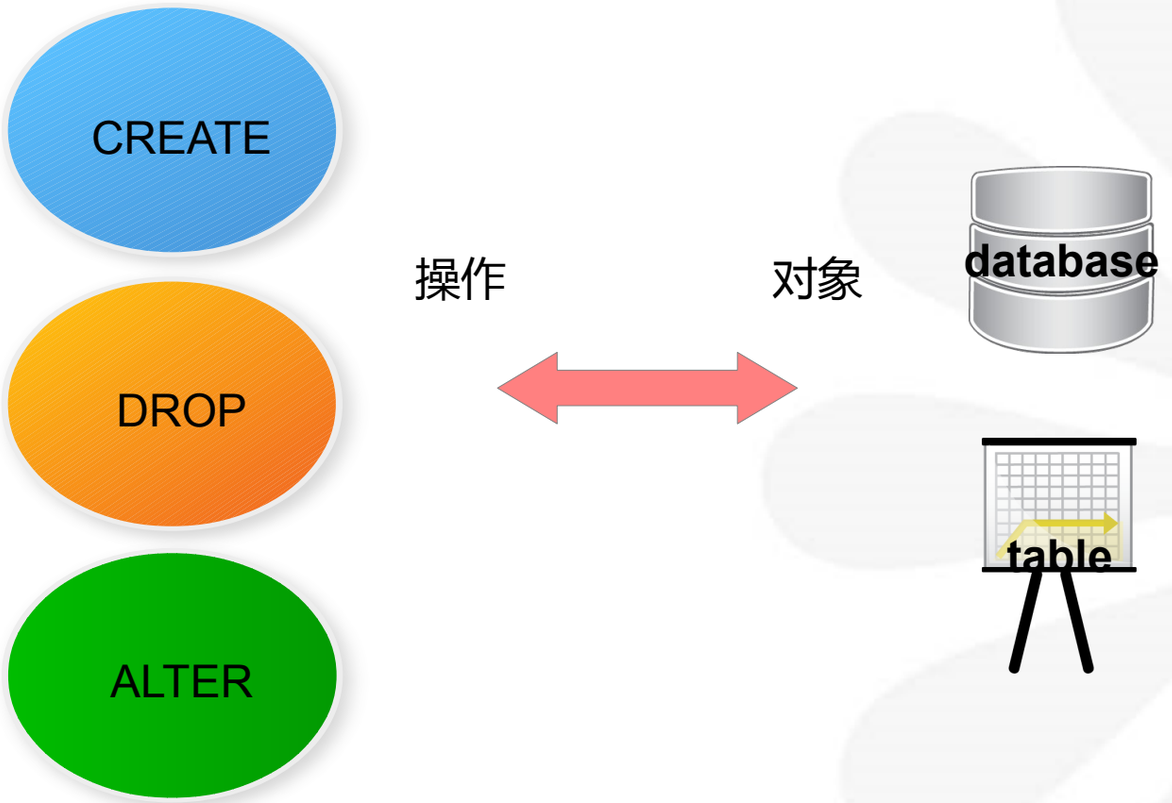
- 位运算符是用来对二进制字节中的位进行测试、移位或者测试处理。位或（ | ）
  - 位与（ & ）
  - 位异或（ ^ ）
  - 位左移（ << ）
  - 位右移（ << ）
  - 位取反（ ~ ）

运算符的优先级

- 运算的优先级决定了不同的运算符在表达式中计算的先后顺序。
- 级别高的运算符先进行计算，如果级别相同， MySQL 按表达式的顺序从左到右依次计算。当然，在无法确定优先级的情况下，可以使用圆括号“（ ）”来改变优先级。

SQL DDL

- DDL 语法

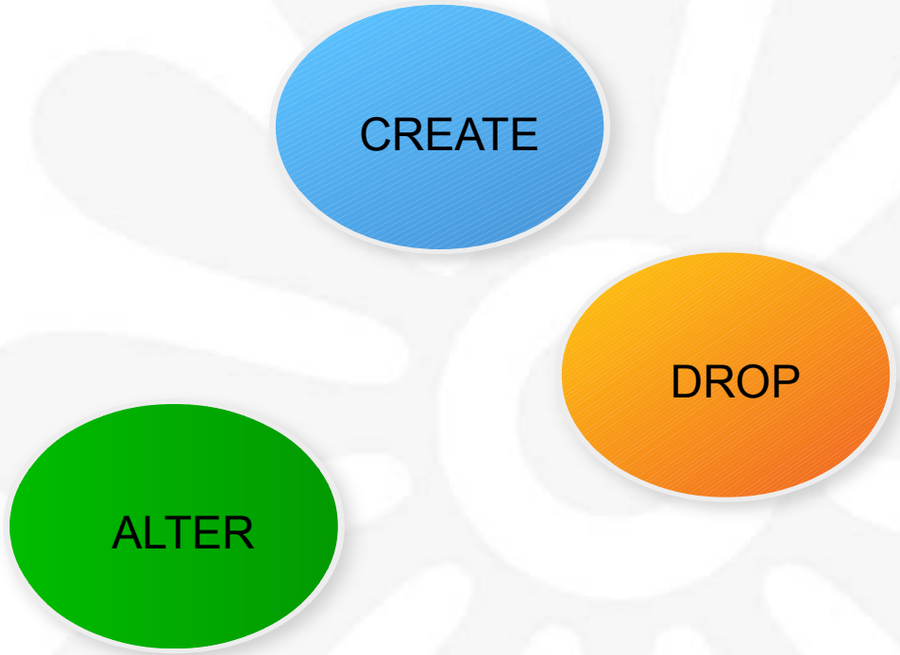


SQL DDL

创建数据库	create database 库名 ;
删除数据库	drop database 库名 ;
修改数据库校验规则	alter database 库名 [default] character set [=] 字符编码 | [default] collate [=]
使用数据库	use 库名 ;

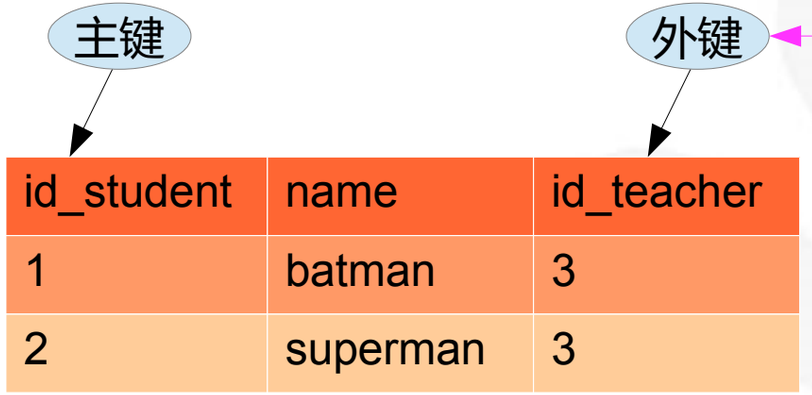
SQL DDL 练习

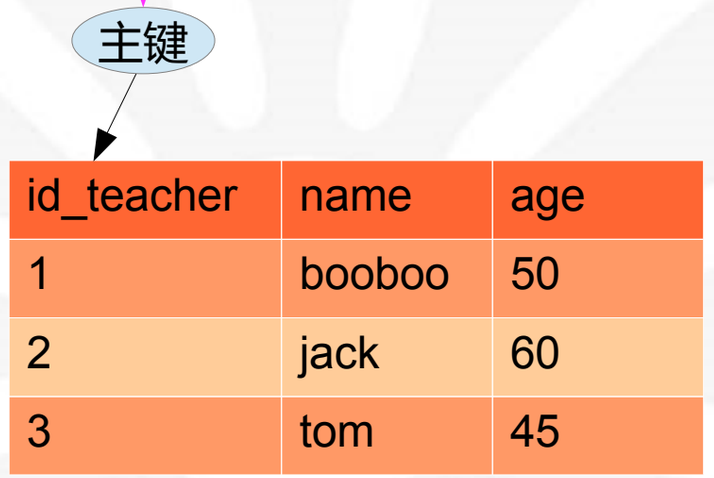
1. 创建库 db1
2. 修改 db1 的字符编码为 utf8
3. 删除库 db1



SQL DDL

- 主键约束 PRIMARY
- 外键约束
- 非空约束 NOT NULL | NULL
- 唯一性约束 UNIQUE
- 默认约束
- 自动增加 AUTO_INCREMENT





SQL DDL

创建表　　create table 表名 like 已存在的表名 ;

create table 表名 (列名 数据类型 约束条件 , 列名 数据类型 约束条件 ...) engine= 存储引擎 default charset=utf8 collate=utf8_bin;

修改表　　alter table 表名 add 列名 数据类型 约束条件 ;

alter table 表名 drop column 列名 ;

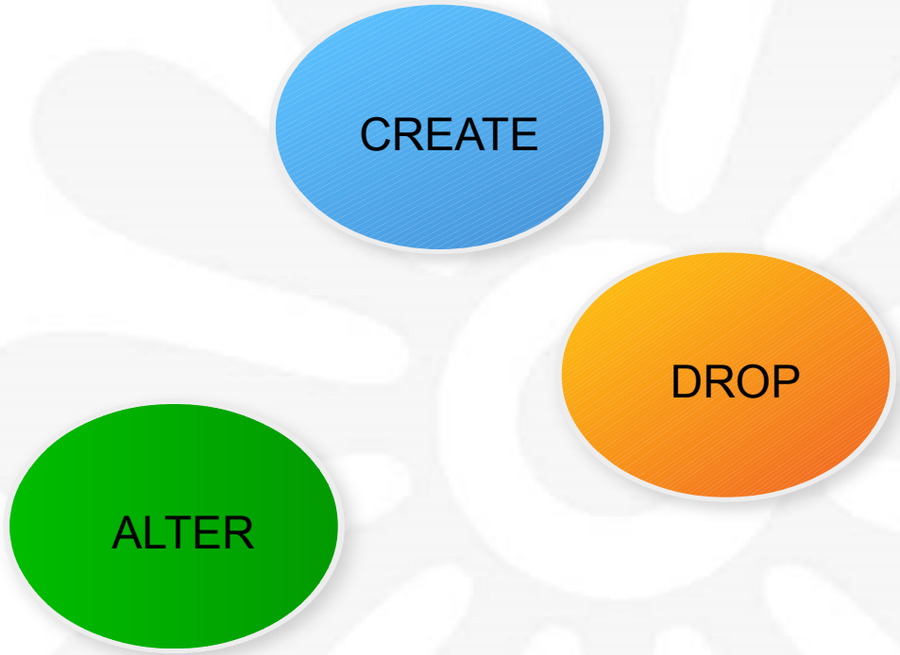
定义外键　　alter table 表 1 add [constraint [symbol]] foreign key (列名) references 表 2 (列名);

重命名表　　rename table 表名 to 新表名 ;

删除表　　drop table 表名 ;

SQL DDL 练习

1. 创建库 uplooking
2. 创建表 student ，有 id_student （主键）和 name 列
3. 创建表 teachers ，有 id_teacher （主键）和 name 列
4. 创建表 t1 ，有 id （主键，自增）和 values 列
5. 重命名表 student 为 students
6. 给 student 表新增 id_teacher 列
7. 给 teachers 表新增 age 列
8. 给 teachers 表删除 age 列
9. 给 students 表添加外键 id_teacher 列
10. 删除 t1 表
11. 创建库 db1
12. 删除库 db1

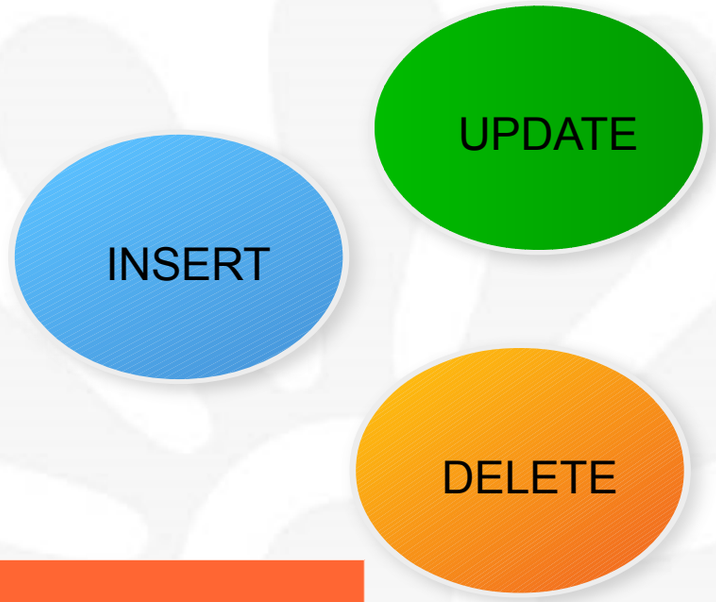


SQL DML

- 插入 insert　　insert into 表名 set 列名 = 值 ...;

  insert into 表名（列 ,...) values （值 ,...),(...),...;
- 更新 update　　update 表名　set 列名 = 值 ... [where where_condition];
- 删除 delete　　delete from 表名 [where where_condition];

SQL DML 练习



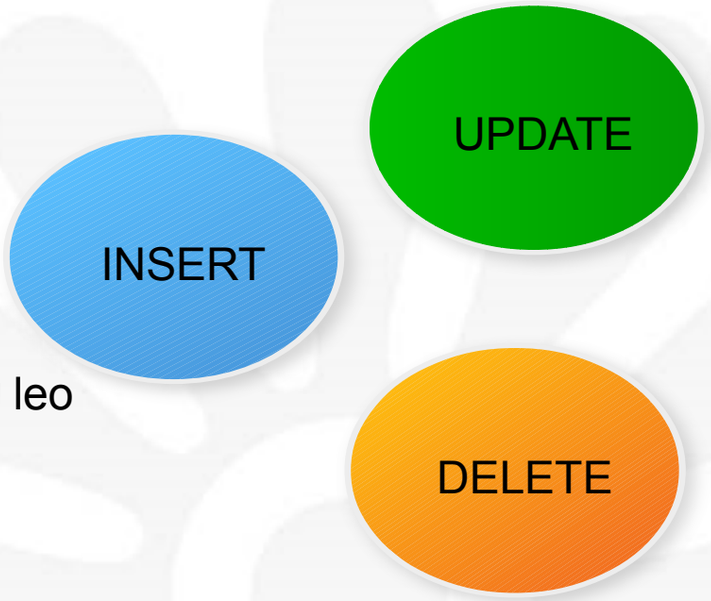
1. 插入以下数据到 teachers 表中：

id_teacher	name
1	Booboo
2	Kevin
3	Mark

2. 插入以下数据到 students 表中：

id_student	name	id_teacher
1	superman	1
2	batman	1
3	wonderwoman	1

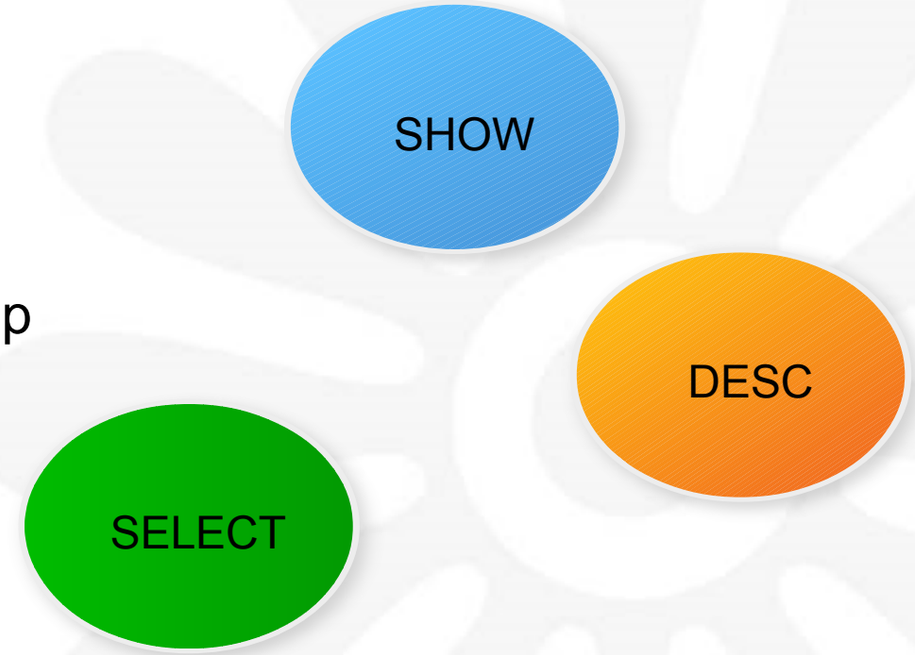
SQL DML 练习



3. 修改 teachers 表中 id_teacher 列为 3 的 name 列的值为 John
4. 修改 students 表中 name 列为 batman 并且 id_teacher 为 1 的行，将 name 改为 leo
5. 修改 students 表中 id_teacher=1 或者 id_teacher=2 的行，将 id_teacher 改为 3
6. 删除 students 表中 id_student=3 的行
7. 删除 students 表和 teachers 表中所有的行

SQL DQL

1. 下载学习样例 mysql_scripts.zip
2. 导入 SQL 练习表



vendors 表

存储销售产品的供应商。每个供应商在这个表中有一个记录,供应商ID(vend_id)列用来匹配产品和供应商。

vendors 表的列	说明
vend_id	唯一的供应商ID
vend_name	供应商名
vend_address	供应商的地址
vend_city	供应商的城市
vend_state	供应商的州
vend_zip	供应商的邮政编码
vend_country	供应商的国家

products 表

包含产品目录,每行一个产品。每个产品有唯一的ID(prod_id 列),通过 vend_id (供应商的唯一ID)关联到它的供应商。

products 表的列	说明
prod_id	唯一的产品ID
vend_id	产品供应商ID(关联到vendors表中的vend_id)
prod_name	产品名
prod_price	产品价格
prod_desc	产品描述

customers 表

存储所有顾客的信息。每个顾客有唯一的ID(cust_id列)。

customers 表的列	说明
cust_id	唯一的顾客ID
cust_name	顾客名
cust_address	顾客的地址
cust_city	顾客的城市
cust_state	顾客的州
cust_zip	顾客的邮政编码
cust_country	顾客的国家
cust_contact	顾客的联系名
cust_email	顾客的联系email地址

orderitems 表

存储每个订单中的实际物品,每个订单的每个物品占一行。对 orders 中的每一行, orderitems 中有一行或多行。每个订单物品由订单号加订单物品(第一个物品、第二个物品等)唯一标识。订单物品通过 order_num 列(关联到 orders 中订单的唯一ID)与它们相应的订单相关联。此外,每个订单项包含订单物品的产品ID(它关联物品到products 表)。

orderitems 表的列	说明
order_num	订单号(关联到orders表的order_num)
order_item	订单物品号(在某个订单中的顺序)
prod_id	产品ID(关联到products表的prod_id)
quantity	物品数量
item_price	物品价格

productnotes 表

存储与特定产品有关的注释。并非所有产品都有相关的注释,而有的产品可能有许多相关的注释。

productnotes 表的列	说明
note_id	唯一注释ID
prod_id	产品ID(对应于products表中的prod_id)
note_date	增加注释的日期
note_text	注释文本

SQL DQL

- 检索数据 select 列名 | 列，列 |* from 表名；
  - 去重 distinct
  - 限定行数 limit
  - 完全限定表名 dbname.tbname
- 排序检索
  - order by
  - Desc|asc

DQL 检索排序习题

1. products 表中检索一个名为prod_name的列
2. Products 表中检索3个列(prod_id,prod_name,prod_price)
3. Products 表中检索所有列
4. products 表中产品的所有供应商ID，去除重复
5. products 表中检索一个名为prod_name的列，只显示5行
6. products 表中检索一个名为prod_name的列，从第5行开始显示4行
7. products 表中检索一个名为prod_name的列，对 prod_name 列以字母顺序排序。
8. products表中检索3个列(prod_id,prod_name,prod_price)，并按其中两个列对结果进行排序——首先按价格prod_price,然后再按名称prod_name排序
9. products表中检索3个列(prod_id,prod_name,prod_price)，按价格prod_price以降序排序产品(最贵的排在最前面)
10. products表中检索3个列(prod_id,prod_name,prod_price)，按价格prod_price以降序排序产品(最贵的排在最前面)，然后再对产品名排序prod_name

SQL DQL

- 过滤数据
  - Where
  - WHERE 子句操作符
- 数据过滤
  - and | or
  - in | not in

操作符	说明
=	等于
<>	不等于
!=	不等于
<	小于
<=	小于等于
>	大于
>=	大于等于
between	在指定的两个值之间

DQL 过滤数据习题

1. products 表中检索两个列(prod_name,prod_price),但不返回所有行,只返回 prod_price 值为 2.50 的行
2. products 表中检索两个列(prod_name,prod_price),但不返回所有行,只返回 prod_name 的值为 Fuses 的一行
3. 列出价格小于10美元的所有产品
4. 检索价格小于等于10美元的所有产品
5. 不是由供应商 1003 制造的所有产品
6. 检索价格在5美元和10美元之间的所有产品
7. 检查products表中prod_price列具有 NULL 值的行
8. 检查products表中prod_price列具有 NULL 值的行
9. 检查customers表中cust_email列具有 NULL 值的行
10. 检查customers表中cust_email列不具有 NULL 值的行

DQL 数据过滤习题

1. 检索products表由供应商 1003 制造且价格小于等于10美元的所有产品的名称和价格
2. 检索products表由供应商 1002或1003 制造的所有产品的名称和价格
3. 检索products表，列出价格为10美元(含)以上且由 1002 或 1003 制造的所有产品
4. 列出除 1002和1003 之外的所有供应商制造的产品
5. 检索products表，列出由 1002 或 1003 制造的所有产品

SQL DQL

- 通配符过滤
  - like 近似匹配
  - % 任何字符出现任意次数
  - _ 匹配单个字符
- 正则表达式过滤
  - regexp 正则匹配
  - 正则表达式默认不区分大小写
  - regexp binary 区分大小写

DQL 通配符和正则过滤习题

对 products 表进行检索：

1. 找出所有以词 jet 起头的产品
2. 找出所有包含anvil的产品
3. 找出以 s 起头以 e 结尾的所有产品
4. 找出以 开头有一位，之后是空格，然后为ton anvil的所有产品
5. 产品名称中匹配1000的
6. 产品名称中匹配000的
7. 产品名称中匹配jetpack的，不区分大小写
8. 产品名称中匹配JetPack的，区分大小写
9. 产品名称中匹配1000或者2000的
10. 产品名称中匹配 1000 或 2000 或 3000

1. 产品名称中匹配 1 或 2 或 3后面空格ton
2. 产品名称中匹配 1 到 5 后面空格接ton
3. 产品名称中包含 . 字符的值
4. 产品名称中匹配包括括号，并且括号中的内容依次为 一位数字，一个空格，stick，s有或者没有
5. 产品名称中匹配4个数字
6. 以一个数(包括以小数点开始的数)开始的所有产品

SQL DQL

- 创建计算字段
  - 删除空白　trim() ltrim() rtrim()
  - 拼接字段　concat()
  - 别名　as
- 数据处理函数
  - 字符串函数
  - 数字函数
  - 日期和时间函数
  - 控制流程函数
  - 系统函数

常用的文本处理函数	说 明
Left()	返回串左边的字符
Right()	返回串右边的字符
Lower()	将串转换为小写
Upper()	将串转换为大写
LTrim()	去掉串左边的空格
RTrim()	去掉串右边的空格
Length()	返回串的长度
Soundex()	返回串的SOUNDEX值
Locate()	找出串的一个子串
SubString()	返回子串的字符

常用日期和时间处理函数	说 明
AddDate()	增加一个日期(天、周等)
AddTime()	增加一个时间(时、分等)
CurDate()	返回当前日期
CurTime()	返回当前时间
Date()	返回日期时间的日期部分
DateDiff()	计算两个日期之差
Date_Add()	高度灵活的日期运算函数
Date_Format()	返回一个格式化的日期或时间串
Day()	返回一个日期的天数部分
DayOfWeek()	对于一个日期,返回对应的星期几
Hour()	返回一个时间的小时部分
Minute()	返回一个时间的分钟部分
Month()	返回一个日期的月份部分
Now()	返回当前日期和时间
Second()	返回一个时间的秒部分
Time()	返回一个日期时间的时间部分
Year()	返回一个日期的年份部分

常用数值处理函数	说 明
Abs()	返回一个数的绝对值
Cos()	返回一个角度的余弦
Exp()	返回一个数的指数值
Mod()	返回除操作的余数
Pi()	返回圆周率
Rand()	返回一个随机数
Sin()	返回一个角度的正弦
Sqrt()	返回一个数的平方根
Tan()	返回一个角度的正切

控制流程函数	说明
if(expr,v1,v2)	如果表达式expr为真则返回v1，否则返回v2
ifnull(v1,v2)	如果v1不为null，则返回v1，否则返回v2
case expr when v1 then r1 [when v2 then r2] [else rn] end	如果expr的值等于某个vn，则放回对应的then的结果，否则返回else后面的rn

```
MariaDB [(none)]> select if(2>1,'yes','no'),if(1>2,1,2);
+------------------+------------+
| if(2>1,'yes','no') | if(1>2,1,2) |
+------------------+------------+
| yes              |          2 |
+------------------+------------+
1 row in set (0.00 sec)

MariaDB [(none)]> select ifnull(1,2),ifnull(NULL,'wrong');
+------------+--------------------+
| ifnull(1,2) | ifnull(NULL,'wrong') |
+------------+--------------------+
|          1 | wrong              |
+------------+--------------------+
1 row in set (0.01 sec)
```

```
MariaDB [(none)]> select case 11 when 1 then 'a' when 10 then 'b' else 'c' end;
+-----------------------------------------------------+
| case 11 when 1 then 'a' when 10 then 'b' else 'c' end |
+-----------------------------------------------------+
| c                                                   |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

系统信息函数	说明
version()	mysql版本信息
connection_id()	连接数
user()	用户名
charset()	字符集
collation()	验证集
last_insert_id()	最后一个自动生成的ID值
password()	加密

DQL 计算字段习题

1. vendors 表包含供应商名和位置信息。生成一个供应商报表,需要在供应商的名字中按照 name (location) 这样的格式列出供应商的位置
2. orderitems 表包含每个订单中的各项物品。检索订单号 20005 中的所有物品，并汇总物品的价格(单价乘以订购数量)
3. SELECT 3*2; 将返回 6
4. SELECT Trim(' abc '); 将返回 abc
5. SELECT Now() 利用 Now() 函数返回当前日期和时间

DQL 计算字段习题

1. 检索供应商表vendors中的供应商名vend_name，并都转换为大写，排序。
2. customers 表中有一个顾客 Coyote Inc. ,其联系名为Y Lee 。但如果这是输入错误,此联系名实际应该是 Y Lie ,怎么办?显然,按正确的联系名搜索不会返回数据。
3. 从orders表中检索出一个订单记录,该订单记录的 order_date 为 2005-09-01
4. 从orders表中检索出2005年9月下的所有订单
5. 获取pi的值
6. 获取随机数

SQL DQL

- 汇总数据
  - 聚集函数
  - 组合聚集函数
- 分组数据
  - 创建分组　group by
  - 汇总分组　with rollup
  - 过滤分组　having
  - 分组和排序 group by 和 order by

SQL聚集函数	说明
AVG()	返回某列的平均值
COUNT()	返回某列的行数
MAX()	返回某列的最大值
MIN()	返回某列的最小值
SUM()	返回某列值之和

DQL 聚集函数习题

1. 使用 AVG() 返回 products 表中所有产品的平均价格 avg_Price
2. 使用 AVG() 返回 products 表中产品供应商1003提供的商品的平均价格 avg_Price
3. 返回 customers 表中客户的总数
4. 对 customers 表中具有电子邮件地址的客户计数,别名num_cust
5. 返回products 表中最贵的物品的价格
6. orderitems表 包含订单中实际的物品,每个物品有相应的数量(quantity)。检索所订购物品的总数(所有quantity 值之和)
7. orderitems表 合计每项物品的item_price*quantity ,得出总的订单金额total_price
8. orderitems表 返回vend_id = 1003的供应商提供的产品的平均价格，只考虑各个不同的价格
9. 检索products 表中物品的数目num_items,产品价格的最高price_max、最低price_min以及平均值price_avg

DQL 分组数据习题

1. products表按 vend_id 排序并分组数据计算每个供应商的商品总数 num_prods
2. products表按 vend_id 排序并分组数据计算每个供应商的商品总数 num_prods，并将每列汇总
3. orders表过滤两个以上的订单的那些分组
4. products表列出具有 2 个(含)以上、价格为 10 (含)以上的产品的供应商
5. orderitems表检索总计订单价格大于等于 50 的订单的订单号和总计订单价格

下表是在 SELECT 语句中使用时必须遵循的次序，列出迄今为止所学过的子句

SELECT子句	说明	是否必须使用
SELECT	要返回的列或表达式	是
FROM	从中检索数据的表	仅在从表选择数据时使用
WHERE	行级过滤	否
GROUP BY	分组说明	仅在按组计算聚集时使用
HAVING	组级过滤	否
ORDER BY	输出排序顺序	否
LIMIT	要检索的行数	否

SQL DQL 高级语法

- 子查询
- 联结表
- 组合查询
- 全文本搜索

SQL DCL

- 添加认证授权 grant all on *.* to booboo@'172.25.0.11' identified by 'uplooking';
  - all 权限
  - *.* 所有库 . 所有表
- 刷新授权表 flush privileges;
- 回收权限 revoke [权限] on [库].[表] from booboo@'172.25.0.11';
- 删除用户 drop user 'jeffrey'@'localhost';
- 修改密码

实战项目：完成数据库用户权限操作项目
>> 要求 : 添加授权用户测试客户机优先使用的哪个密码 (X 为学生机号)

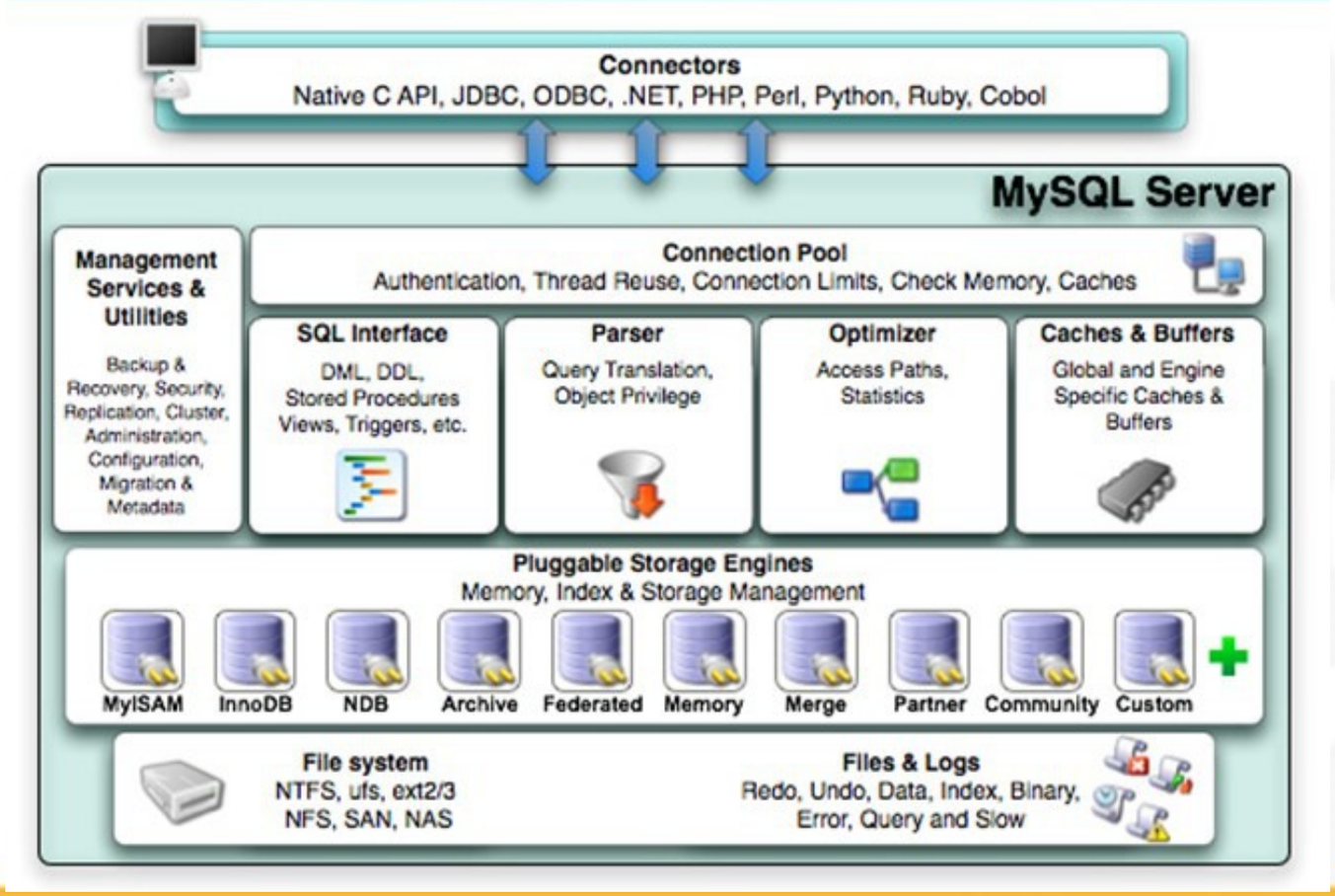
1. user1@'172.25.X.%' uplooking
2. user1@'172.25.X.12' uplooking123

SQL 拓展内容

- 视图
- 存储过程
- 游标
- 触发器
- 全球化和本地化

1.4 MySQL 逻辑架构和 Innodb 存储引擎

MySQL 逻辑架构



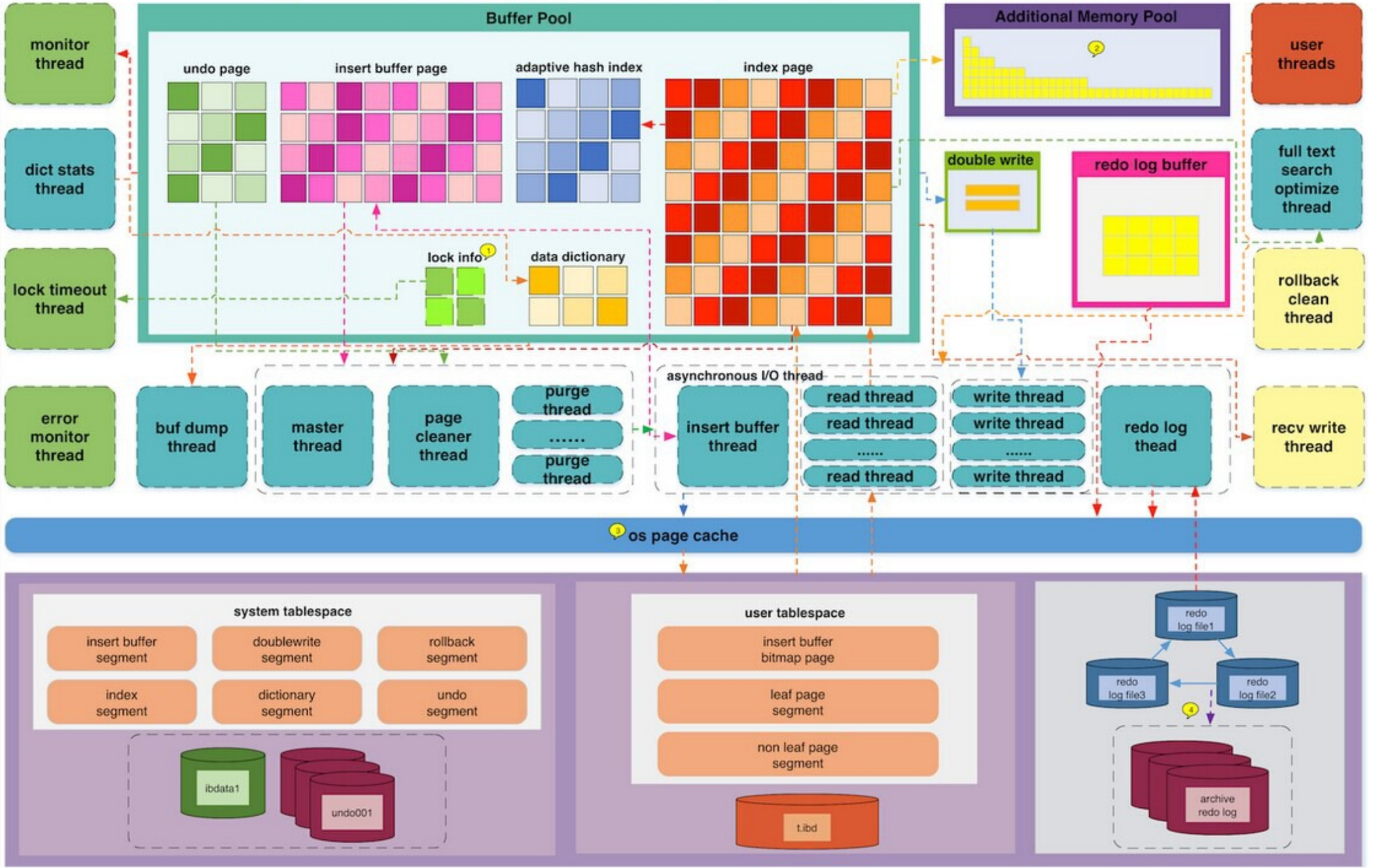
MySQL 存储引擎

Feature	MyISAM	BDB	Memory	InnoDB	Archive	NDB
Storage Limits	No	No	Yes	64TB	No	Yes
Transactions (commit, rollback, etc.)		✔		✔		
Locking granularity	Table	Page	Table	Row	Row	Row
MVCC/Snapshot Read				✔	✔	✔
Geospatial support	✔					
B-Tree indexes	✔	✔	✔	✔		✔
Hash indexes			✔	✔		✔
Full text search index	✔					
Clustered index				✔		
Data Caches			✔	✔		✔
Index Caches	✔		✔	✔		✔
Compressed data	✔				✔	
Encrypted data (via function)	✔	✔	✔	✔	✔	✔
Storage cost (space used)	Low	Low	N/A	High	Very Low	Low
Memory cost	Low	Low	Medium	High	Low	High
Bulk Insert Speed	High	High	High	Low	Very High	High
Cluster database support						✔
Replication support	✔	✔	✔	✔	✔	✔
Foreign key support				✔		
Backup/Point-in-time recovery	✔	✔	✔	✔	✔	✔
Query cache support	✔	✔	✔	✔	✔	✔
Update Statistics for Data Dictionary	✔	✔	✔	✔	✔	✔

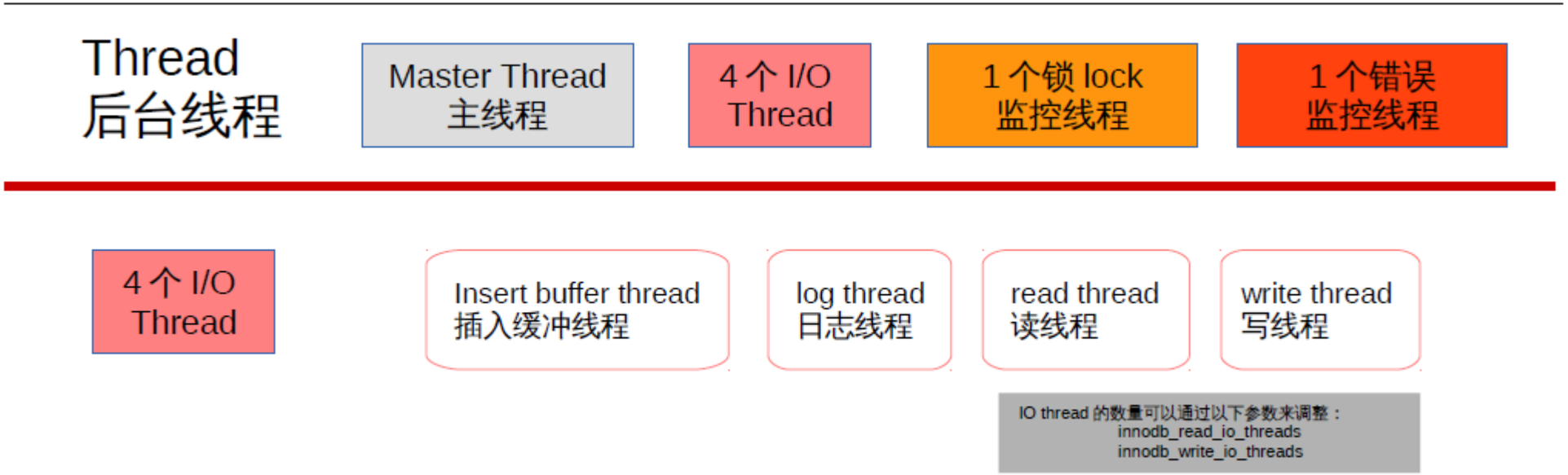
MYISAM 和 INNODB

存储引擎	事务	锁精度	使用场景
MYISAM	NO	表锁	OLAP
INNODB	YES	行锁	OLTP

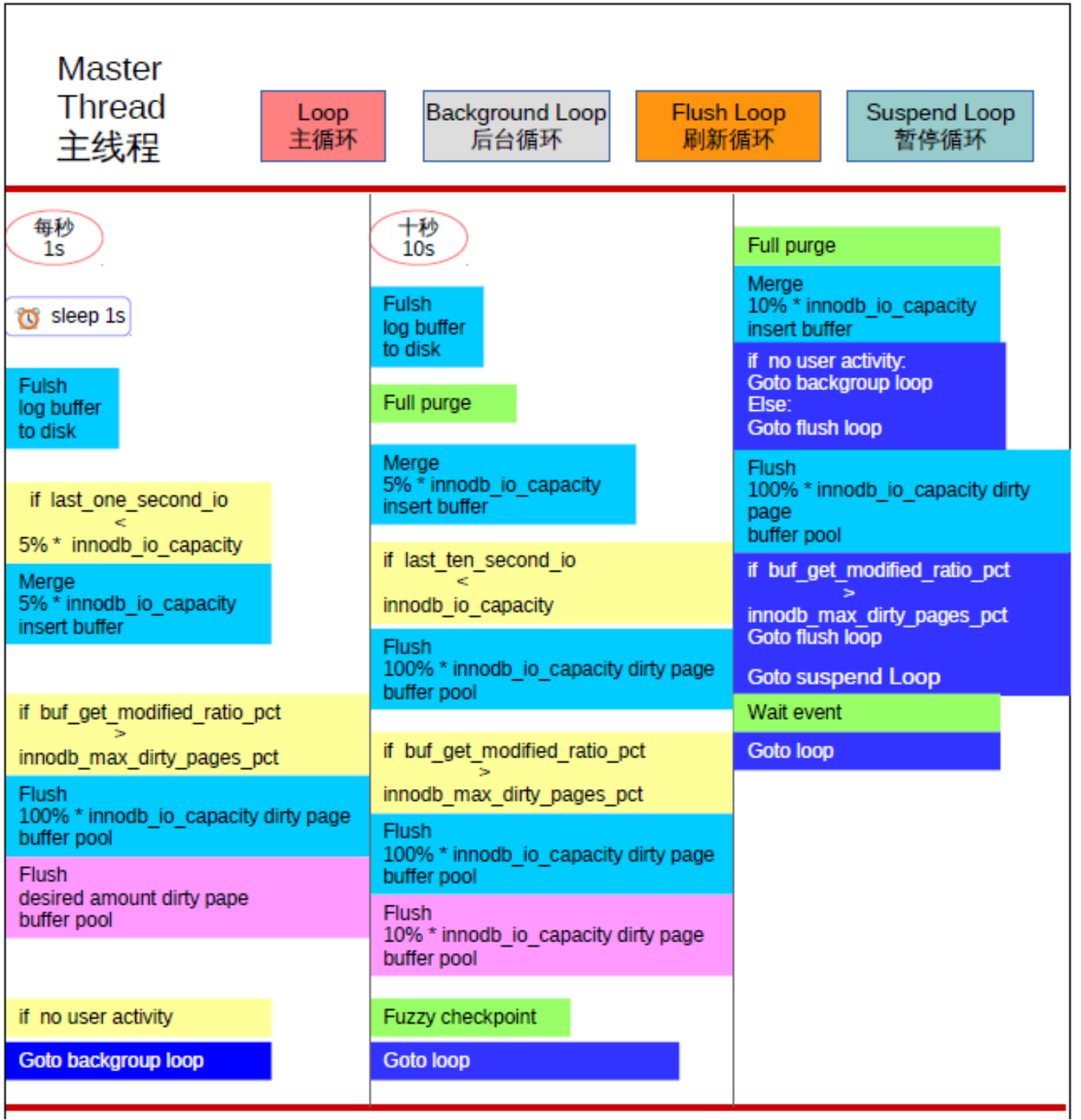
MySQL 5.6 | InnoDB Storage Engine Architecture



INNODB 后台线程



主体系结构：默认 7 个后台线程， 4 个 io thread(insert buffer 、 log 、 read 、 write),1 个 master thread(优先级最高),1 个锁 (lock) 监控线程， 1 个错误监控线程。可以通过 show engine innodb status 来查看。新版本已对默认的 read thread 和 write thread 分别增大到 4 个，可通过 show variables like 'innodb_io_thread%' 查看。


Master Thread
主线程
Loop
主循环
Background Loop
后台循环
Flush Loop
刷新循环
Suspend Loop
暂停循环
每秒
1s
sleep 1s
Fulsh
log buffer
to disk
if last_one_second_io
<
5% * innodb_io_capacity
Merge
5% * innodb_io_capacity
insert buffer
if buf_get_modified_ratio_pct
>
innodb_max_dirty_pages_pct
Flush
100% * innodb_io_capacity dirty page
buffer pool
Flush
desired amount dirty pape
buffer pool
if no user activity
Goto backgroup loop
十秒
10s
Fulsh
log buffer
to disk
Full purge
Merge
5% * innodb_io_capacity
insert buffer
if last_ten_second_io
<
innodb_io_capacity
Flush
100% * innodb_io_capacity dirty page
buffer pool
if buf_get_modified_ratio_pct
>
innodb_max_dirty_pages_pct
Flush
100% * innodb_io_capacity dirty page
buffer pool
Flush
10% * innodb_io_capacity dirty page
buffer pool
Fuzzy checkpoint
Goto loop
Full purge
Merge
10% * innodb_io_capacity
insert buffer
if no user activity:
Goto backgroup loop
Else:
Goto flush loop
Flush
100% * innodb_io_capacity dirty
page
buffer pool
if buf_get_modified_ratio_pct
>
innodb_max_dirty_pages_pct
Goto flush loop
Goto suspend Loop
Wait event
Goto loop

master thread:

loop 主循环每秒一次的操作：

1. 睡觉
2. 将日志缓冲刷新到磁盘，即使事务未提交
3. 如果【前 1s 的 I/O 次数】小于【磁盘 I/O 吞吐量的 5% （默认 2005%=10 个页）】，则合并插入缓冲（数量为磁盘 I/O 吞吐量 5% 即 10 个页）
4. 如果【脏页比例】大于【阀值（默认为 75 ，脏页比例为百分之七十五）】，则刷新缓冲池中脏页到磁盘，否则只刷新合适的脏页数量（ innodb_adaptive_flushing 参数决定）
5. 当前没有用户活动，切换到后台循环

loop 主循环每十秒一次的操作：

1. 将日志缓冲刷新到磁盘，即使事务未提交
2. 删除无用的 undo 页
3. 合并插入缓冲，缓冲数量为【磁盘 I/O 吞吐量的 5% ，如果默认 200 ，则合并插入缓冲 10 个】
4. 如果【前 10s 的 I/O 次数】小于【磁盘 I/O 吞吐量（默认 200100%=200 个页）】，则刷新缓冲池中脏页到磁盘（数量为磁盘 I/O 吞吐量 100% 即 200 个页）
5. 如果【脏页比例】大于【阀值（默认为 75 ，脏页比例为百分之七十五）】，则刷新缓冲池中脏页到磁盘（数量为磁盘 I/O 吞吐量 100% 即 200 个页），否则脏页刷新数量为（磁盘 I/O 吞吐量 10% 即 20 个页）
6. 产生一个检查点（模糊检查点）

backgroud loop ，若当前没有用户活动 (数据库空闲时) 或者数据库关闭时，就会切换到这个循环：

1. 删除无用的 undo 页
2. 合并插入缓冲，缓冲数量为【磁盘 I/O 吞吐量的 10% ，如果默认 200 ，则合并插入缓冲 20 个】
3. 当前有用户活动，切换到主循环
4. 当前没有用户活动，切换到刷新循环

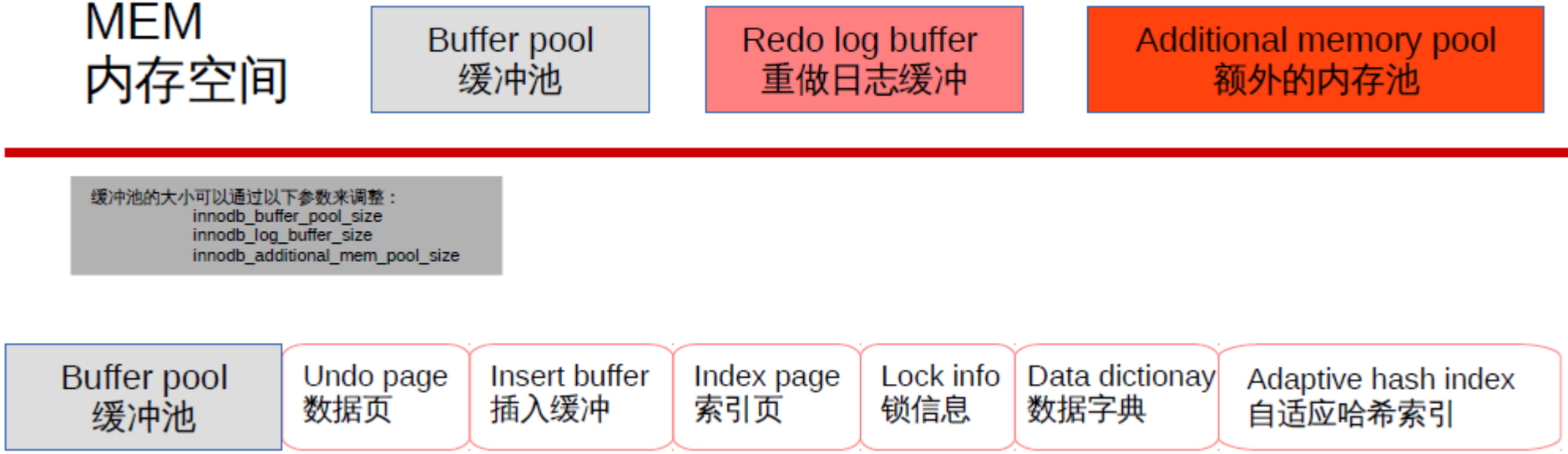
flush loop 刷新循环，包括：

1. 刷新缓冲池中脏页到磁盘（数量为磁盘 I/O 吞吐量 *100% 即 200 个页）
2. 如果【脏页比例】大于【阀值（默认为 75 ，脏页比例为百分之七十五）】，则继续进入刷新循环，否则进入暂停循环

suspend_loop 暂停循环，包括：

1. 等待事件发生，进入主循环

INNODB 内存空间

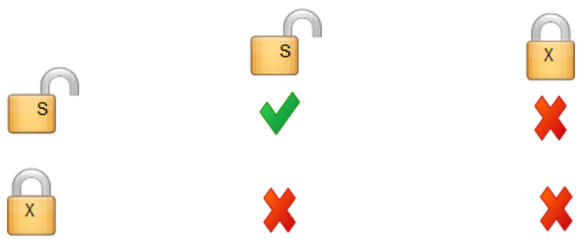


内存空间：缓冲池 (buffer pool) 、重做日志缓冲池 (redo log buffer) 以及额外的内存池 (additional memory pool). 具体配置可由
show variables like 'innodb_buffer_pool_size' 、
show variables like 'innodb_log_buffer_size' 、
show variables like 'innodb_additional_mem_pool_size' 来查看。

INNODB 的特性

- 事务
- 并发控制
- 锁机制
- 多版本并发控制 MVCC
- 事务的隔离级别

	LOCK 锁	LATCH 闩
管理对共享资源的并发访问		
对象	事务	线程
保护	数据库内容	内存数据结构
持续时间	整个事务过程	临界资源
模式	行锁、表锁、意向锁	读写锁、互斥量
死锁	通过 wait-for graph 、 time out 等机制进行死锁检测与处理	无死锁检测与处理机制。仅通过应用程序加锁的顺序 lock leveling 保证无死锁
存在于	Lock Manager 的哈希表	每个数据结构的对象中



Latch 轻量级闩锁，锁定时间必须非常短。若持续时间长则性能会差。目的是为了保证并发线程操作临界资源的正确性，并且没有死锁检测机制。
Lock 我们一般说的锁，使用对象为事务，锁定对象为数据库中的对象，例如表、页、行。一般 lock 的对象仅在 commit 或 rollback 后进行释放。有死锁机制。

共享锁 S ：允许事务读
排他锁 X ：允许事务写（ update 或 delete ）

查看 innodb 存储引擎中的 latch ：

```
> show engine innodb mutex;
```

查看 innodb 存储引擎中的 lock:

```
> show engine innodb status;
> select * from information_schema.innodb_trx;
> select * from information_schema.innodb_locks;
> select * from information_schema.innodb_lock_waits;
```

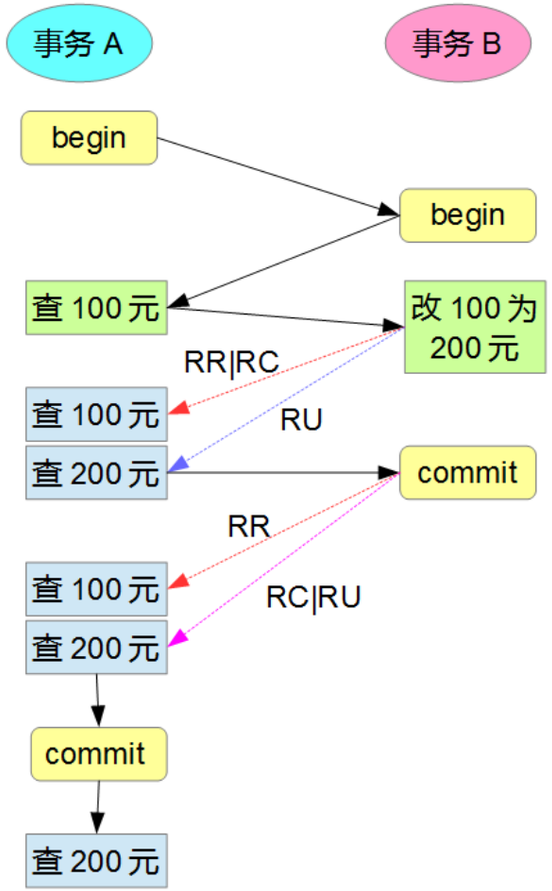
设置 INNODB 事务隔离级别

- 设置 INNODB 事务隔离级别
  - 实践 1 ：查看 innodb 默认的事务隔离级别
  - 实践 2 ：改变单个会话的隔离级别
  - 实践 3 ：改变单个实例的隔离级别
  - 实践 4 ：改变所有实例的隔离级别

区分 INNODB 事务隔离级别

- InnoDB 中的隔离级详细描述
- 实践 1 ： SERIALIZABLE 隔离级别查询自动加共享锁
- 实践 2 ： RU 、 RC 、 RR 隔离级别的对比

ISOLATION 隔离级别	READ UNCOMMITTED	READ COMMITTED	REPEATABLE READ	SERIALIZABLE



InnoDB 默认是可重复读的（ REPEATABLE READ ），MVCC 多版本并发控制，实现一致性地非锁定读操作。

1. 命令行用— transaction-isolation 选项
2. 配置文件，为所有连接设置默认隔离级别。

```
[mysqld]
transaction-isolation = {READ-UNCOMMITTED | READ-COMMITTED
                         | REPEATABLE-READ | SERIALIZABLE}
```

3. 用户可以用 SET TRANSACTION 语句改变单个会话或者所有新连接的隔离级别。

```
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL
                    {READ UNCOMMITTED | READ COMMITTED
                     | REPEATABLE READ | SERIALIZABLE}
```

4. 查询全局和会话事务隔离级别：

```
SELECT @@global.tx_isolation;
SELECT @@tx_isolation;
```

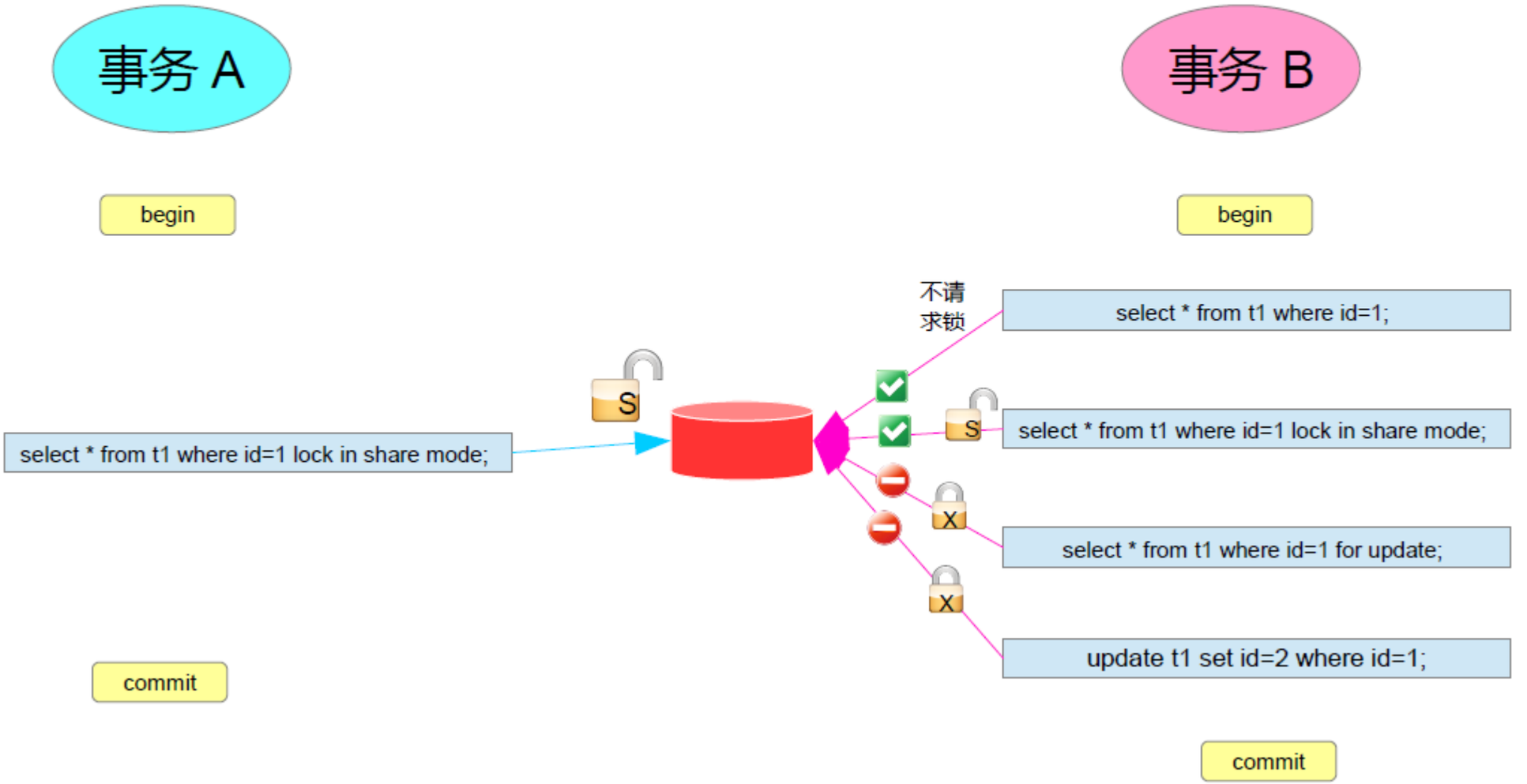
实现一致性锁定读

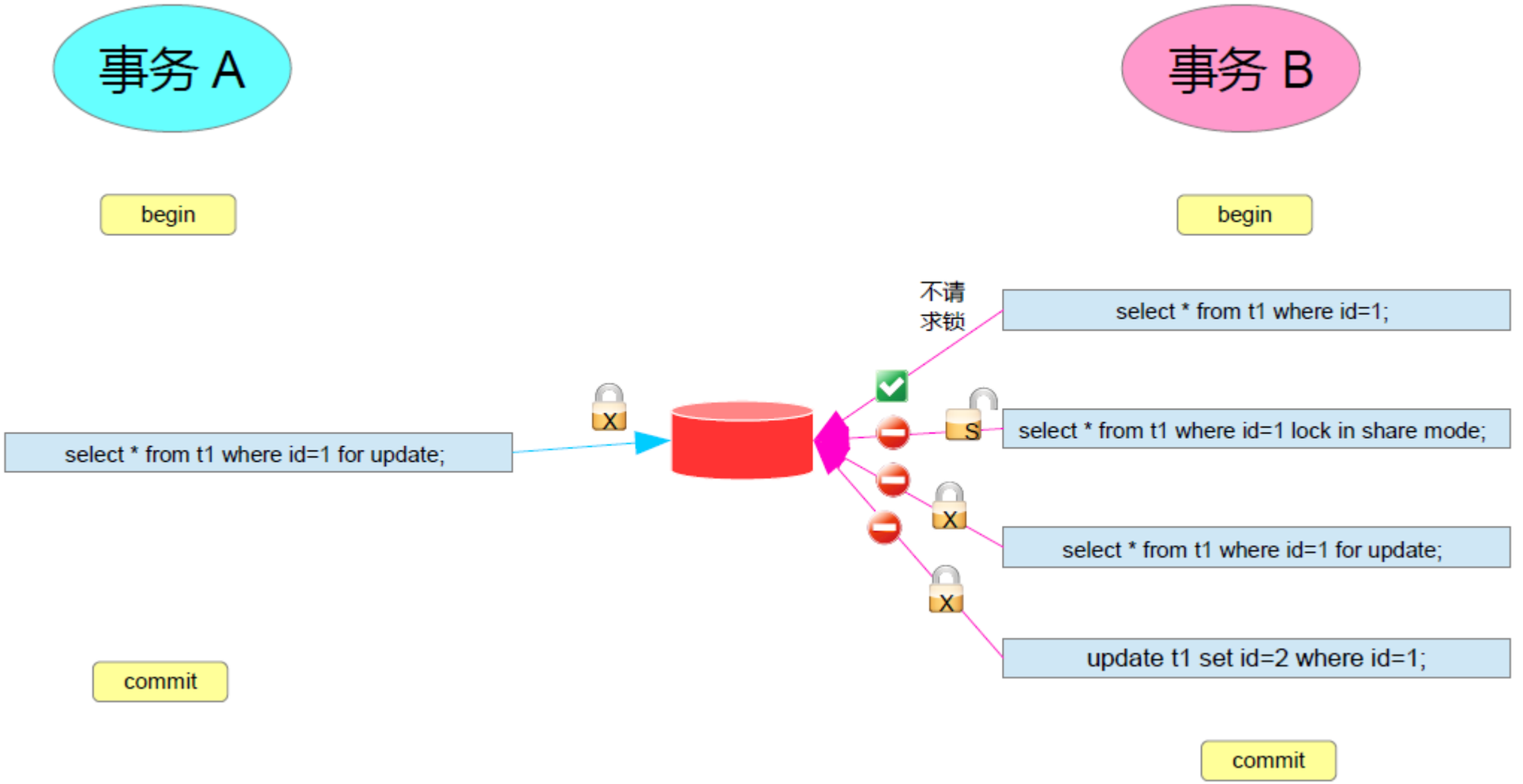
- 实践 1 ：设置 innodb 申请锁等待超时时间
- 实践 2 ：设置一致性锁定读，加共享锁测试
- 实践 3 ：设置一致性锁定读，加排他锁测试

锁定读
非锁定读

一致性锁定读

一致性非锁定读




事务 A
事务 B
begin
begin
不请
求锁
select * from t1 where id=1;
X
S
X
X
select * from t1 where id=1 for update;
select * from t1 where id=1 lock in share mode;
select * from t1 where id=1 for update;
update t1 set id=2 where id=1;
commit
commit

认识锁的算法

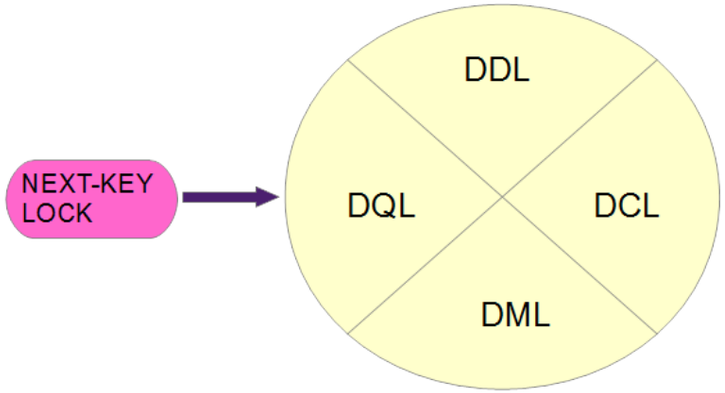
- 实践 1 ： 验证 next-key lock 降级为 record key
- 实践 2 ： 关闭 GAP 锁 _RC
- 实践 3 ： 关闭 GAP 锁 _innodb_locks_unsafe_for_binlog
- 实践 4 ： next-key locking 是如何解决幻读问题的

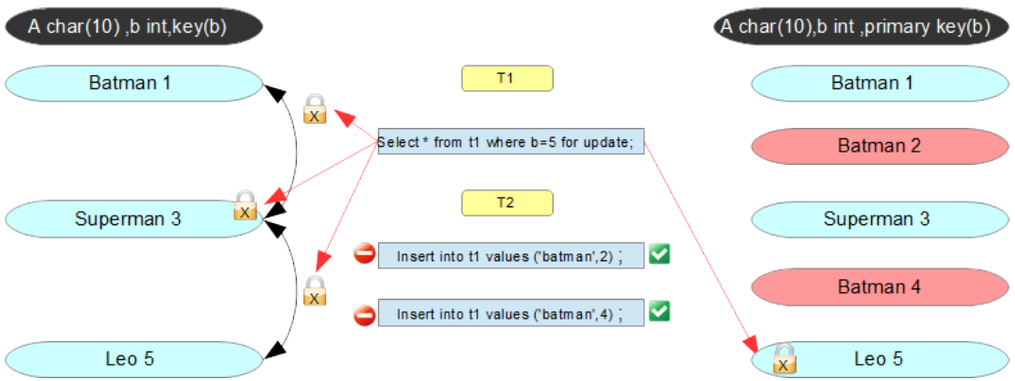
LOCK ALGORITHM
锁的算法

READ LOCK
单个行锁

GAP LOCK
间隙锁

NEXT-KEY LOCK
GAP+RECORD

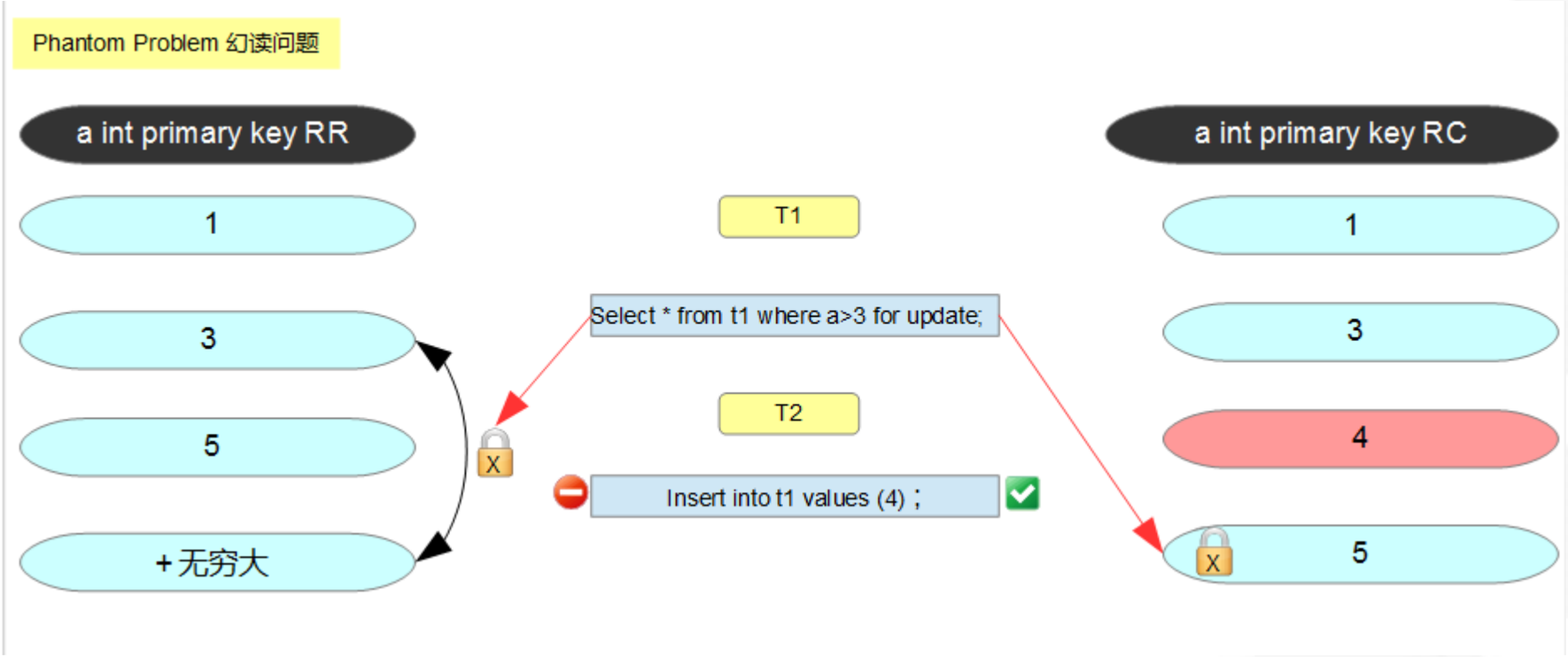




InnoDB 存储引擎的锁的算法有三种：
- Record lock：单个行记录上的锁
- Gap lock：间隙锁，锁定一个范围，不包括记录本身
- Next-key lock：record+gap 锁定一个范围，包含记录本身

Lock 的精度（type）分为 行锁、表锁、意向锁
Lock 的模式（mode）分为
- 锁的类型 ——【读锁和写锁】或者【共享锁和排他锁】即 【X or S】
- 锁的范围 ——【record lock、gap lock、Next-key lock】

1. innodb 对于行的查询使用 next-key lock
2. Next-locking keying 为了解决 Phantom Problem 幻读问题
3. 当查询的索引含有唯一属性时，将 next-key lock 降级为 record key
4. Gap 锁设计的目的是为了阻止多个事务将记录插入到同一范围内，而这会导致幻读问题的产生
5. 有两种方式显式关闭 gap 锁：（除了外键约束和唯一性检查外，其余情况仅使用 record lock）
   - A. 将事务隔离级别设置为 RC
   - B. 将参数 innodb_locks_unsafe_for_binlog 设置为 1



1.5 MySQL 备份与恢复

为什么要备份？
什么是备份？
备份的分类
备份的两大要素
备份和恢复工具
设计 MySQL 备份计划
实战项目

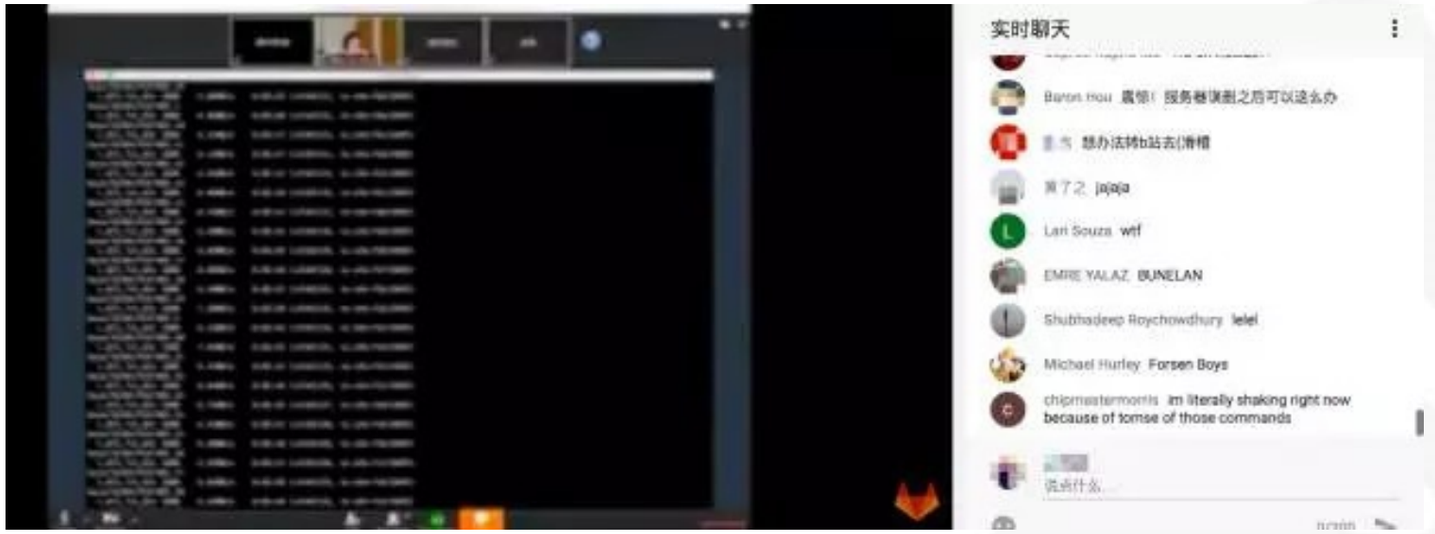
为什么要备份？

2 月 1 日，著名的代码资源托管网站 Gitlab.com 的一位工程师在维护数据时不慎删除约 300GB 的数据，至发文时仍在恢复工作中。

由于不是所有资料都遗失了，所以对用户来说还是稍感安慰，但是该文件在“遇到的问题”（ Problems Encountered ）小节里，最后总结：

“ 因此，换句话说，部署的 5 个不同备份 / 还原技术中，没有一个能可靠地工作或第一时间还原回来，我们只能从 6 小时前有效的备份还原。”

为什么要备份？



亡羊补牢为时不晚， GitLab 展现诚意以 YouTube 直播与 Twitter 将讯息公诸于网络，但是看来 GitLab 必须非常努力，才能挽回客户与投资者对该公司的信心。对其他依赖资讯科技的公司而言，相信这也是很好的借镜。

什么是备份

简单的讲：备份，就是把数据保存一份备用；恢复，就是把保存的数据还原回去。

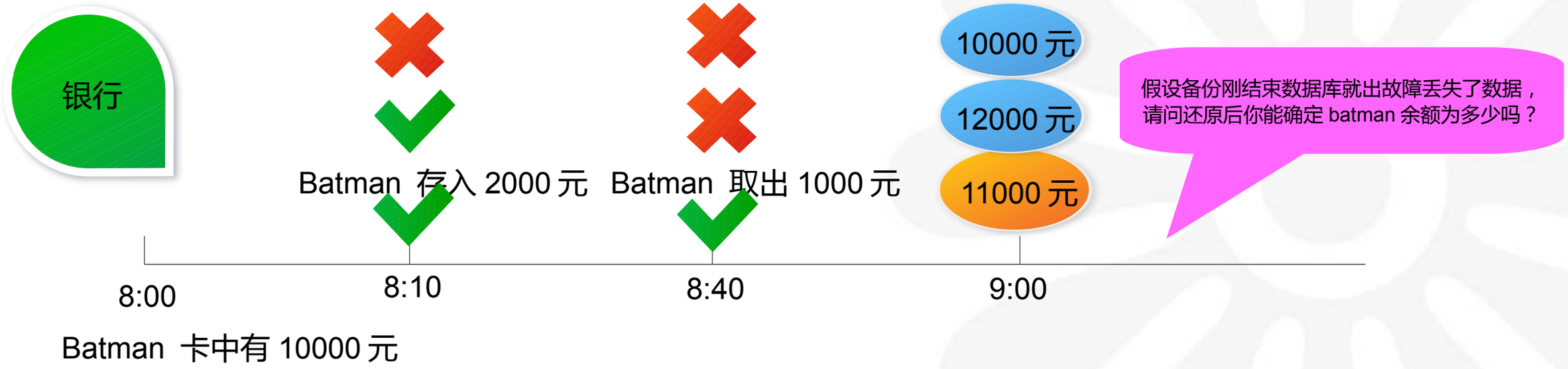
备份的分类

冷备和热备的区别

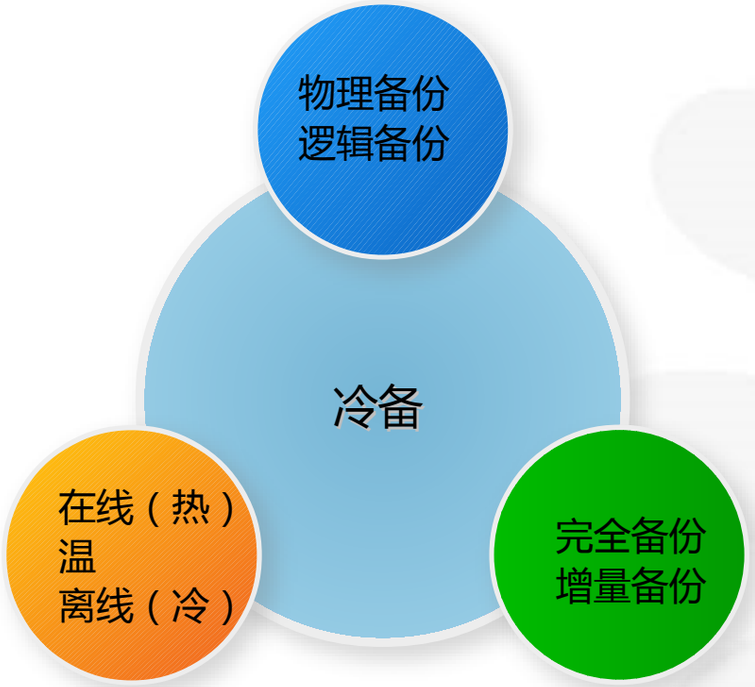
备份	定义	优点	缺点
备份	将数据以隔离的方式保存	解决硬件故障，误操作	不是瞬间还原
冗余	人为地增加重复部分，其目的是用来对原本的单一部分进行备份，以达到增强其安全性的目的,构建冗余（主服务器从服务器）的环境	恢复速度快	解决硬件故障，误操作无法解决

备份的两大要素

数据一致性：备份数据在指定时间点一致
服务可用性：数据库是否可以读写，既能读也能写才是服务可用



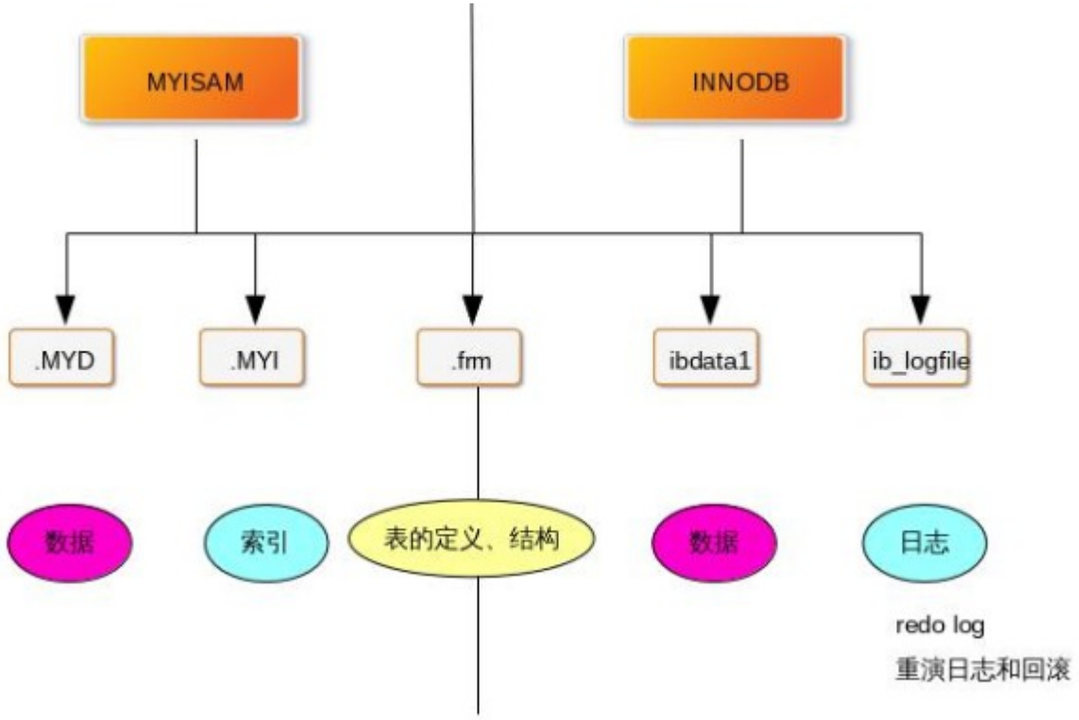
冷备的分类



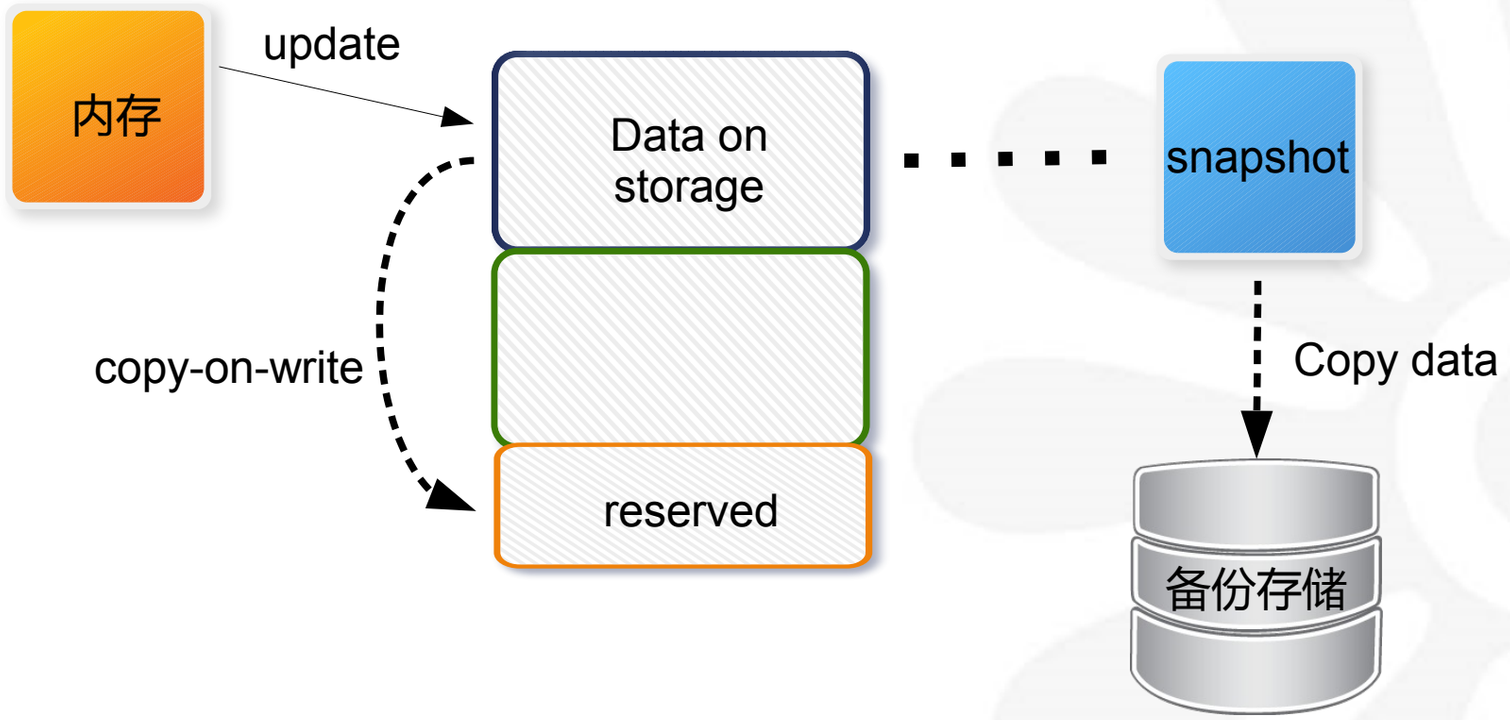
备份和恢复工具

备份工具	物理|逻辑	在线热|离线冷|温	全备|增备	备份速度	还原速度	备份数据量	并发机制
tar	物理	离线冷	全备	快	快	大	
lvm	物理	温，几乎在线热	全备	最快	快	大	
percona xtrabackup	物理	innodb在线热，myisam温	全备和增备	快	快	大	
mysqldump	逻辑	innodb在线热，myisam温	全备	慢	慢	小	单线程无并发
mysqlpump	逻辑	innodb在线热，myisam温	全备	较慢	慢	小	基于表的并发
mydumper	逻辑	innodb在线热，myisam温	全备	稍慢	慢	小	基于行的并发

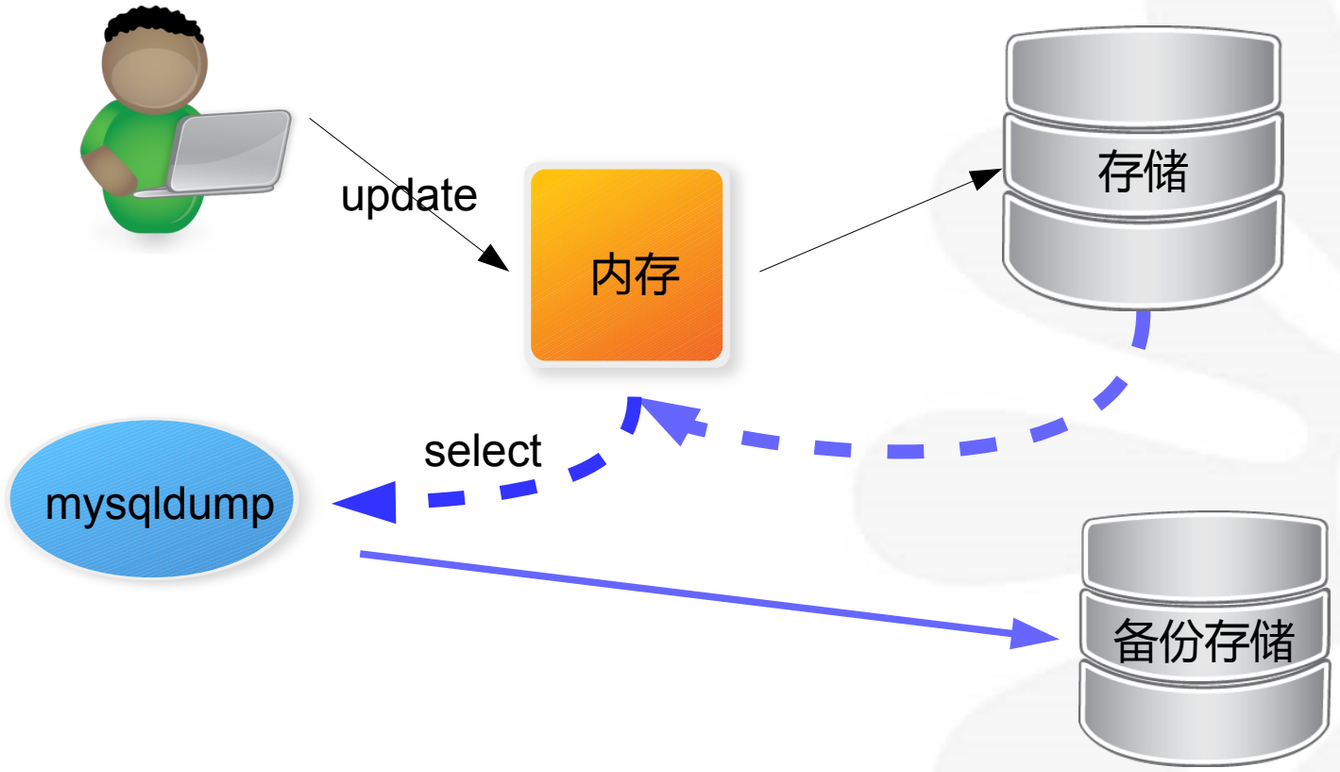
课堂实战 1：利用 tar 实现物理备份并还原

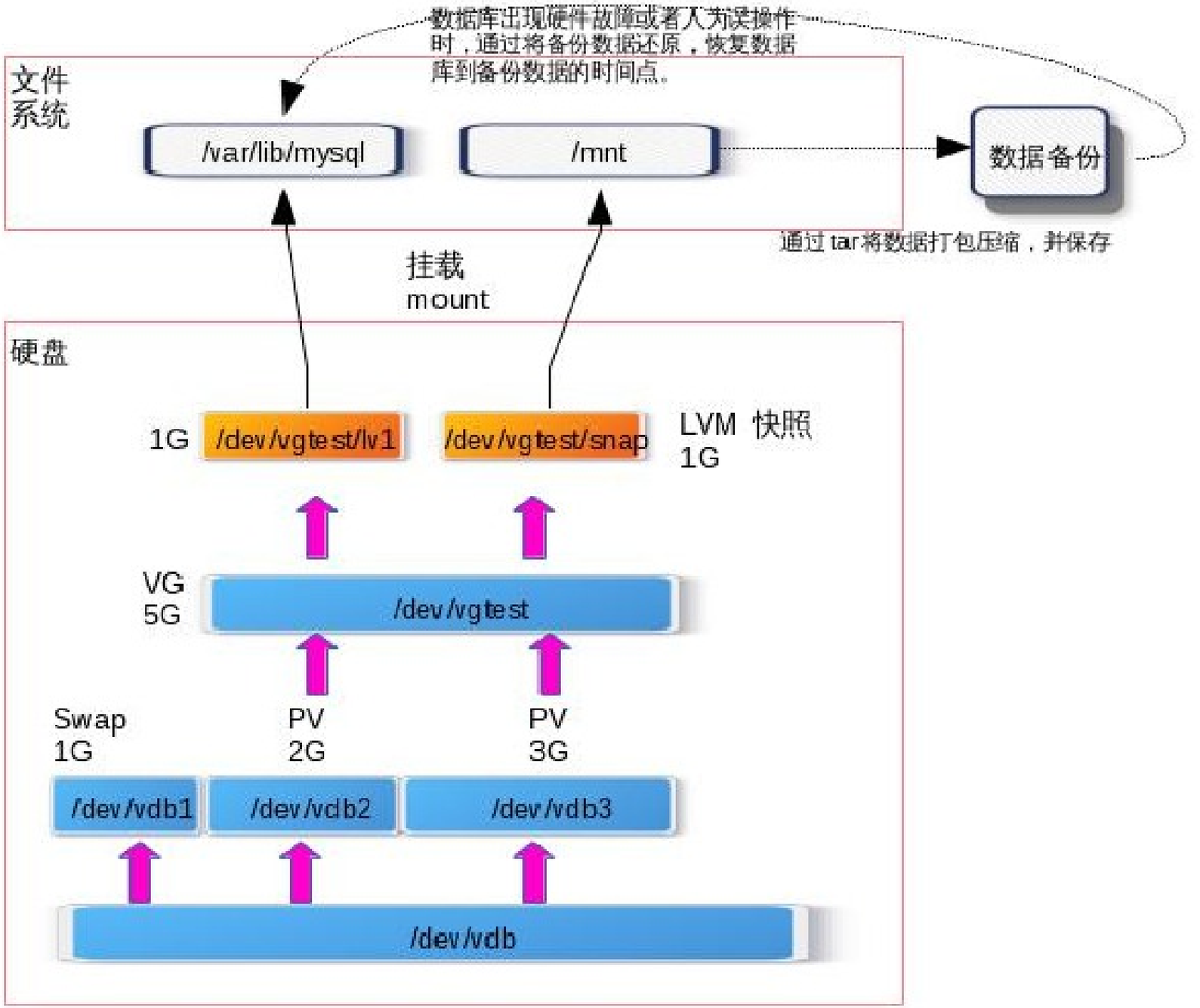


课堂实战 2：利用 LVM 快照实现物理备份并还原

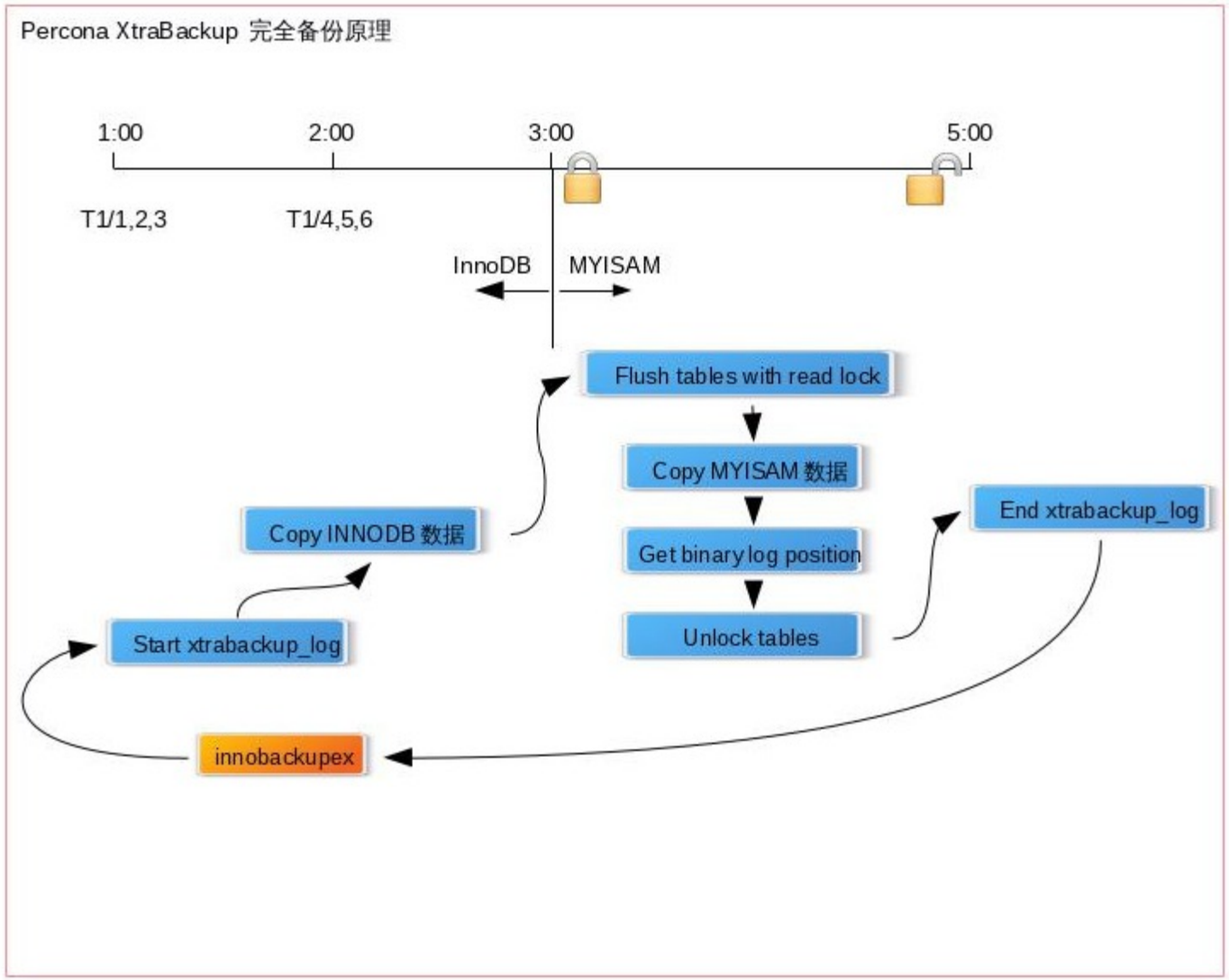


课堂实战 3：利用 mysqldump 实现逻辑备份并还原

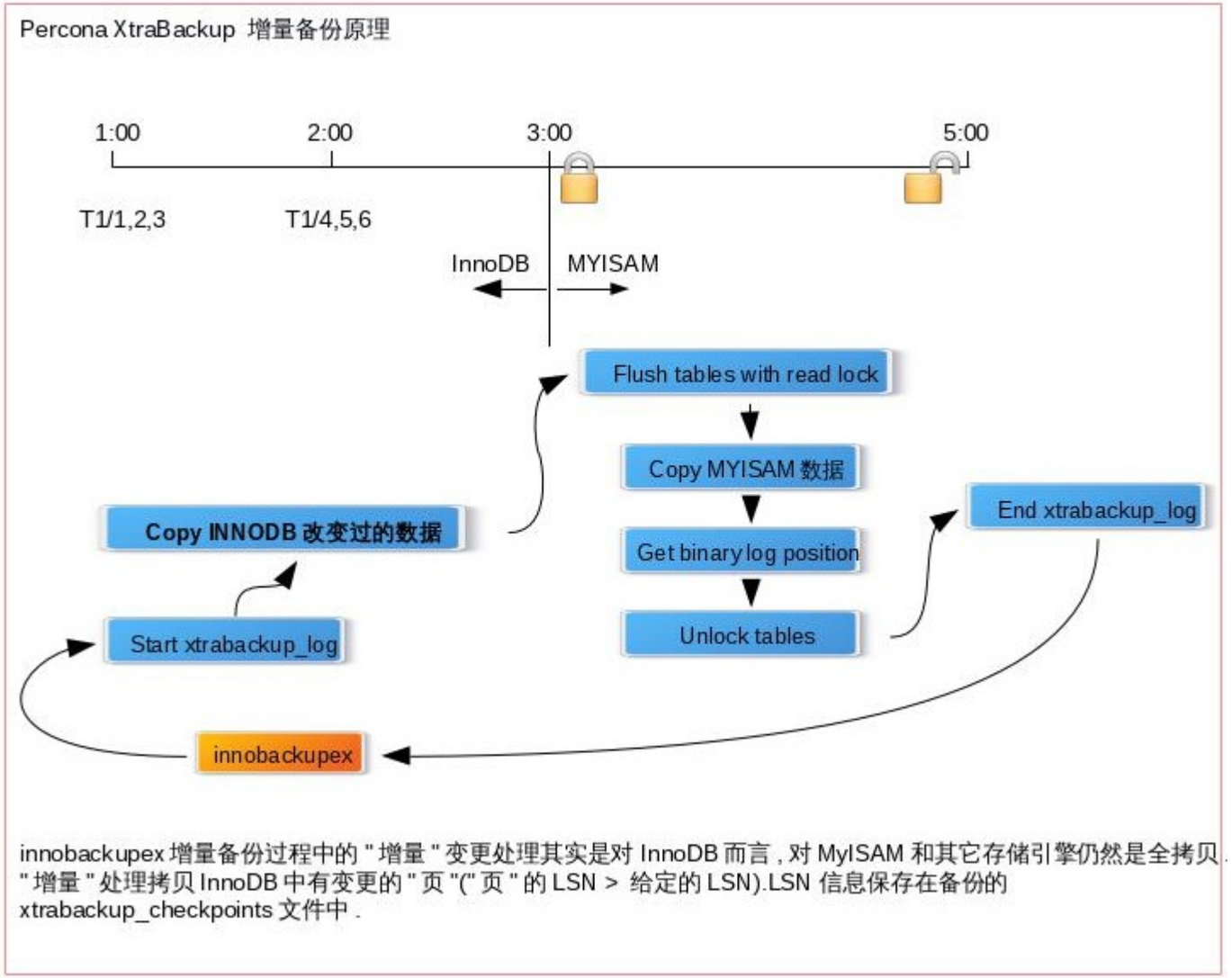


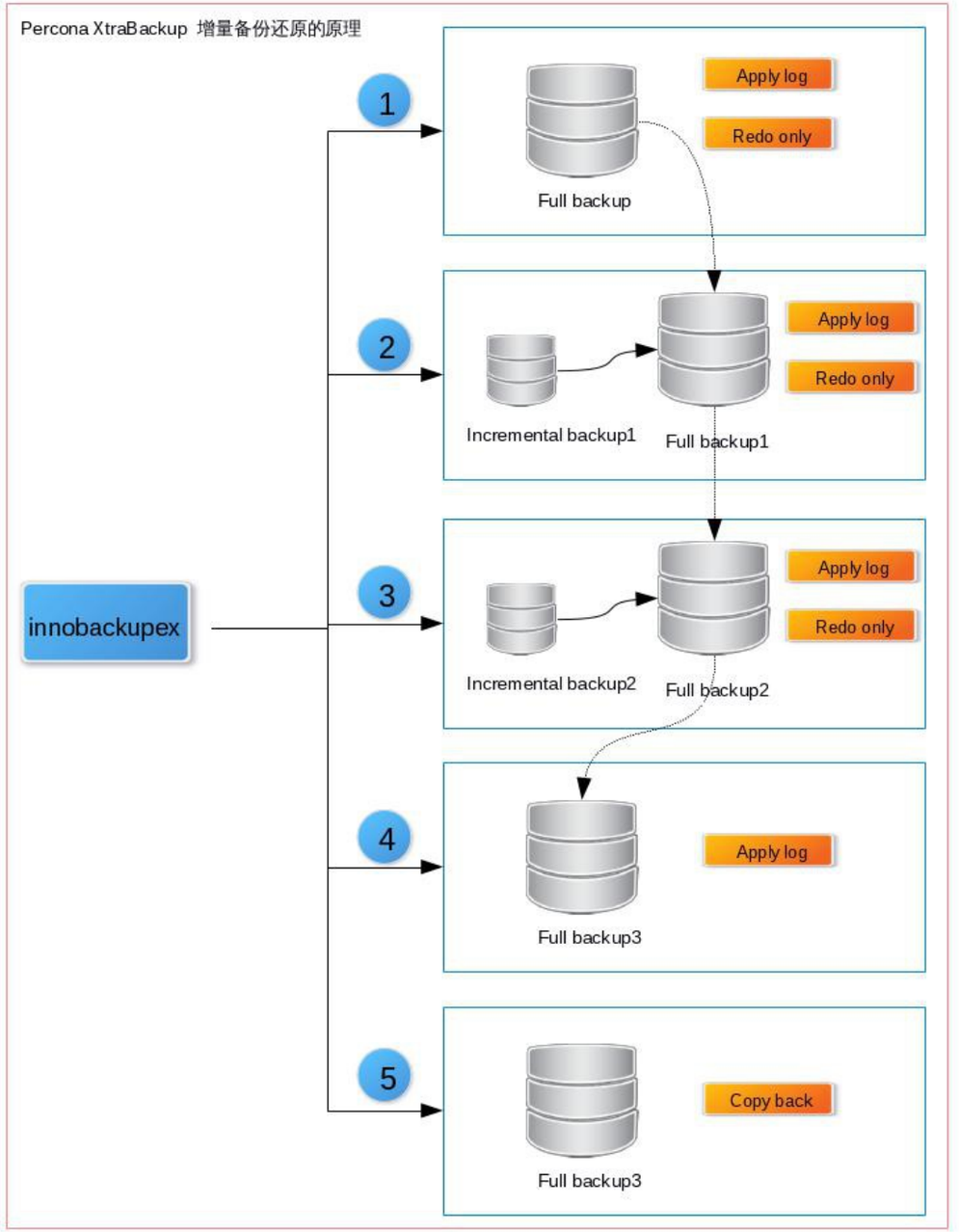

数据库出现硬件故障或者人为误操作时，通过将备份数据还原，恢复数据库到备份数据的时间点。
文件系统
/var/lib/mysql
/mnt
数据备份
通过 tar 将数据打包压缩，并保存
挂载
mount
硬盘
1G
/dev/vgtest/lv1
/dev/vgtest/snap
LVM 快照
1G
VG
5G
/dev/vgtest
Swap
1G
PV
2G
PV
3G
/dev/vdb1
/dev/vdb2
/dev/vdb3
/dev/vdb

课堂实战 4 ： innobackupex 实时增量备份和还原



课堂实战 4 ： innobackupex 实时增量备份和还原





课堂实战 4 ： innobackupex 实时增量备份和还原

MySQL 日志的分类

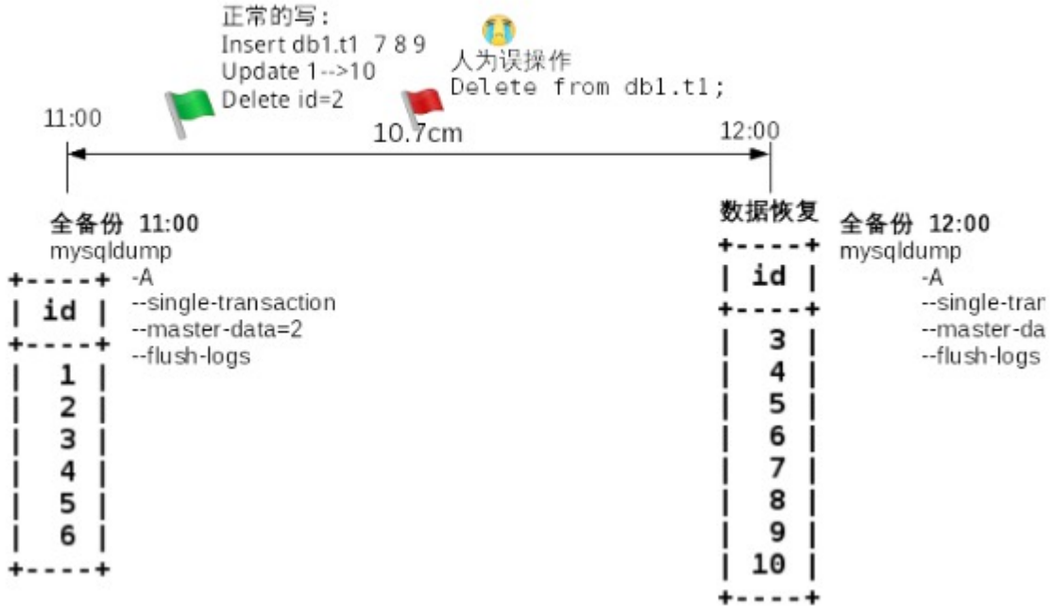
日志分类	解释	说明	作用	配置字段
启动日志	排错日志	/var/log/mariadb/mariadb.log	排错的	log-error
写日志	二进制日志	默认不打开，记录写操作ddl dcl dml	备份	log-bin
读日志	慢查询日志	默认不打开，记录读操作dql	性能调优	log-slow-queries

二进制日志的管理和备份

- 如何打开二进制日志
- 如何查看二进制日志
- 基于二进制日志的实时增量还原

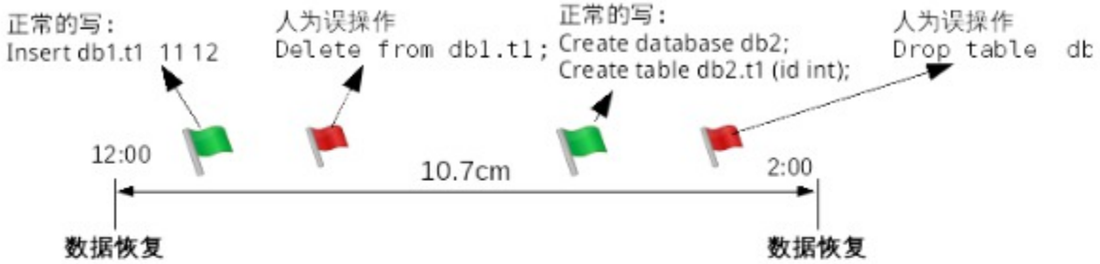
课堂实战 5 ：基于二进制日志时间点和位置的数据库备份恢复模拟

数据库备份恢复演习 1



1) 11:00 mysqldump db1.t1 1 2 3 4 5 6
2) 11:00-12:00 insert 7 8 9 update 1--》 10 delete 2
3) 12:00 人为误操作 delete db1.t1

数据库备份恢复演习 2



全备份 12:00

```
mysqldump
        -A
        --single-transaction
        --master-data=2
        --flush-logs
```

```
+----+
| id |
+----+
|  3 |
|  4 |
|  5 |
|  6 |
|  7 |
|  8 |
|  9 |
| 10 |
+----+
```

```
MariaDB [(Slide 1)]> select * from db1.t1;
+----+
| id |
+----+
|  3 |
|  4 |
|  5 |
|  6 |
|  7 |
|  8 |
|  9 |
| 10 |
| 11 |
| 12 |
+----+
10 rows in set (0.01 sec)

MariaDB [(none)]> desc db2.t1;
+-------+---------+------+-----+---------+-
| Field | Type    | Null | Key | Default |
+-------+---------+------+-----+---------+-
| id    | int(11) | NO   | PRI | NULL    |
+-------+---------+------+-----+---------+-
1 row in set (0.00 sec)
```

1) 12:00 mysqldump db1.t1 3 4 5 6 7 8 9 10
2) 12:00-14:00

正常的写： insert db1.t1 11 12

人为误操作 delete from db1.t1;

正常的写：

Create database db2;

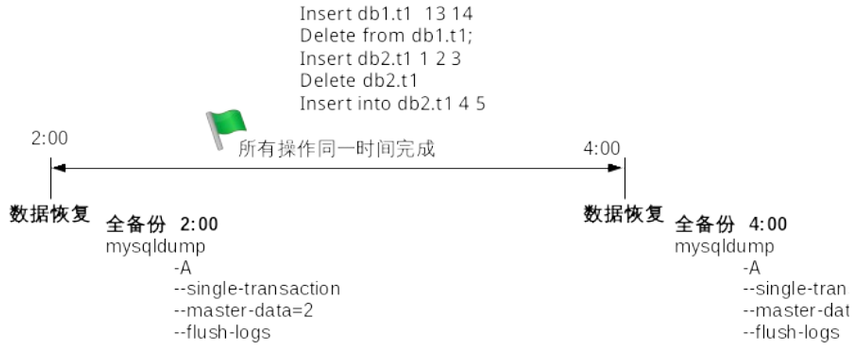
Create table db2.t1 (id int);

人为误操作

Drop table db2.t1;

3) 14:00 恢复

数据库备份恢复演习 3




```
MariaDB [(Slide 1)]> select * from db1.t1;
+----+
| id |
+----+
|  3 |
|  4 |
|  5 |
|  6 |
|  7 |
|  8 |
|  9 |
| 10 |
| 11 |
| 12 |
+----+
10 rows in set (0.01 sec)

MariaDB [(none)]> desc db2.t1;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| id    | int(11) | NO   | PRI | NULL    |       |
+-------+---------+------+-----+---------+-------+
1 row in set (0.00 sec)

MariaDB [(none)]> select * from db2.t1;
Empty set (0.00 sec)
```

```
MariaDB [(none)]> select * from db1.t1;
+----+
| id |
+----+
|  3 |
|  4 |
|  5 |
|  6 |
|  7 |
|  8 |
|  9 |
| 10 |
| 11 |
| 12 |
| 13 |
| 14 |
+----+
12 rows in set (0.00 sec)

MariaDB [(none)]> select * from db2.t1;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
|  4 |
|  5 |
+----+
5 rows in set (0.00 sec)
```

1) 14:00 mysqldump db1.t1 3 4 5 6 7 8 9 10 11 12 db2.t1
2) 14:00-16:00
   insert into db1.t1 values (13),(14);
   delete from db1.t1;
   insert into db2.t1 values (1),(2),(3);
   delete from db2.t1;
   insert into db2.t1 values (4),(5);
3) 16:00 恢复

设计 MySQL 备份计划

接下来你还要考虑

- 在线（热）备份还是离线（冷）备份，又或者温备份？
- 逻辑备份还是物理备份？
- 备份什么？
- 存储引擎和一致性？

MySQL 备份与恢复总结

重点掌握

1. 备份概念
2. 备份的过程
3. Mysqldump 命令
4. 二进制日志
5. 不同的备份方法的原理以及备份过程中会遇到的问题
6. 清楚每一种备份方法的优缺点
7. 掌握备份数据的还原

难点

1. 备份与冗余的区别
2. 备份中服务可用性与数据一致性
3. MVCC 版本控制机制
4. 备份中数据一致性和服务可用性的理解
5. Percona XtraBackup 增量备份和还原的原理

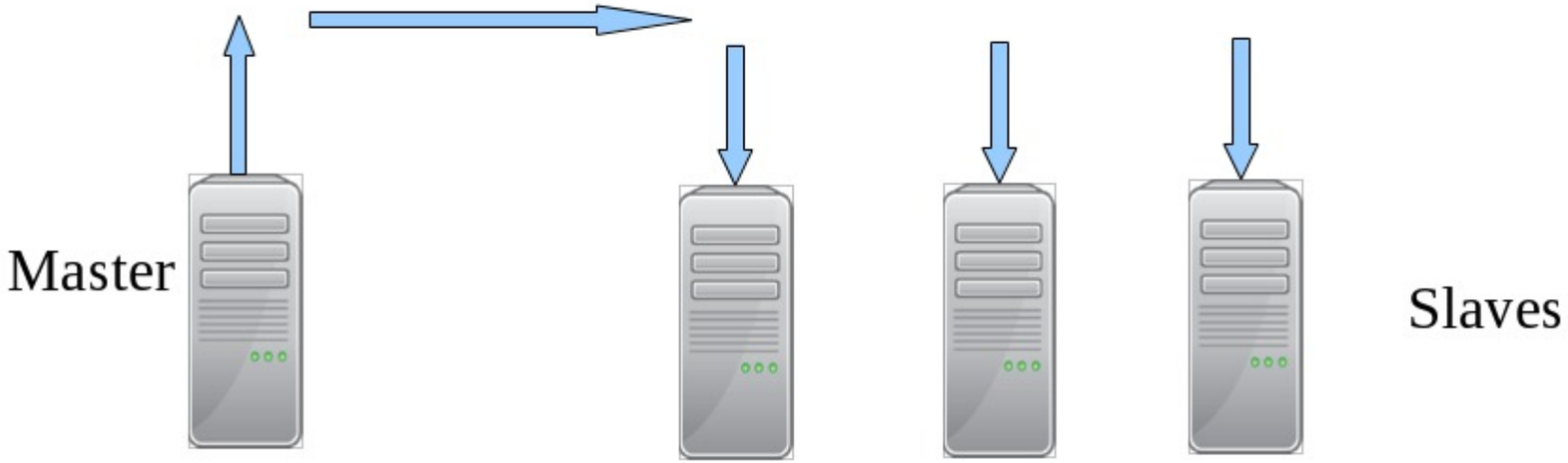
1.6 MySQL 复制 replication

MySQL 复制 replication

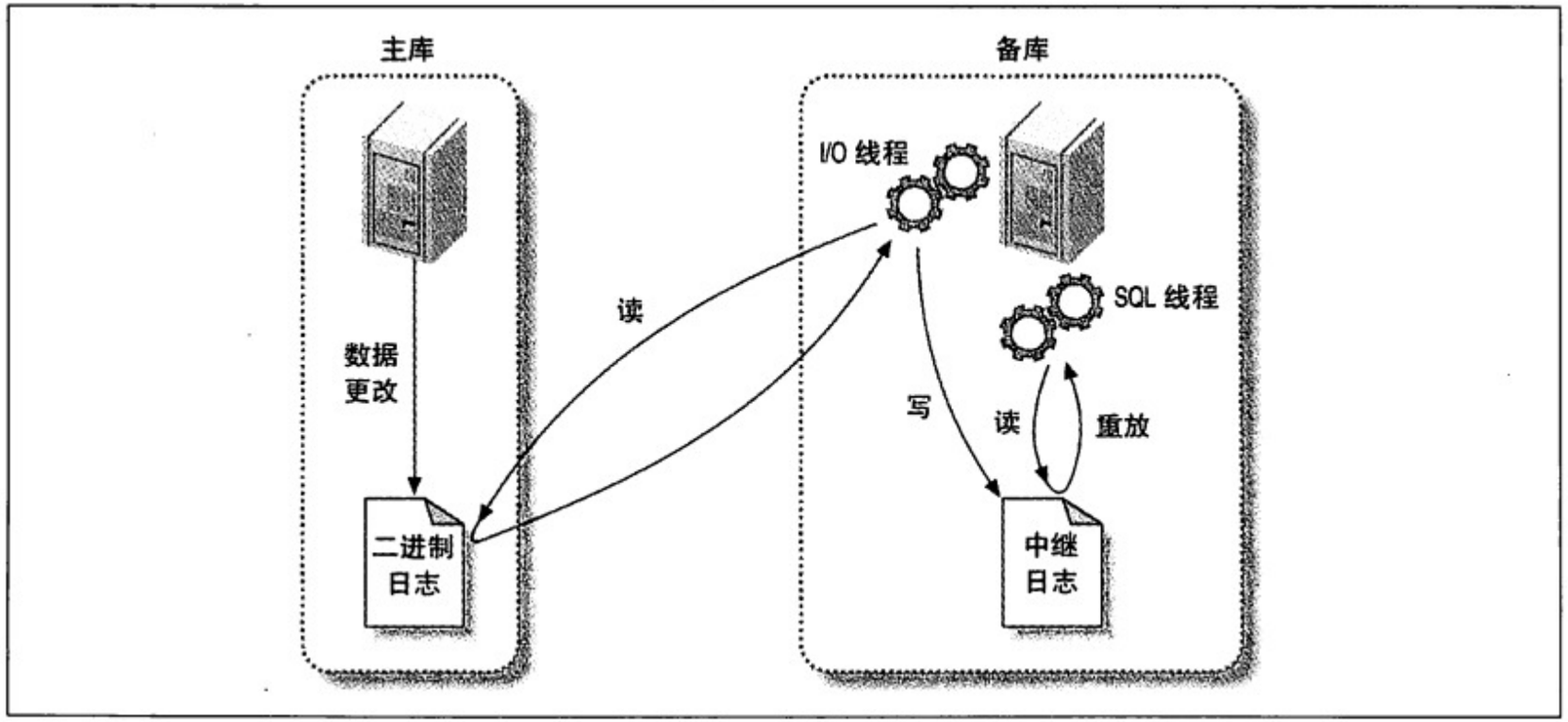
1.复制解决的问题
2.复制的原理
3.复制中的延迟问题
4.复制中的单点故障问题
5.复制实战项目

Replication 解决的问题

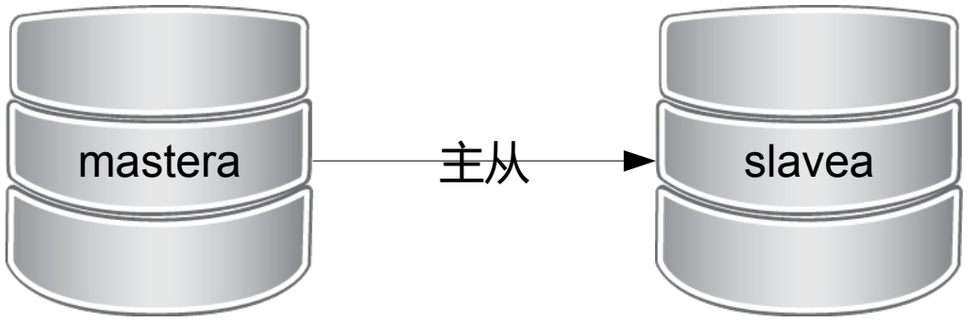
Replication 解决的问题



Replication 的原理



项目实战 1 ： mariadb server 5.5 单主从架构



172.25.0.11
Mariadb-server 5.5

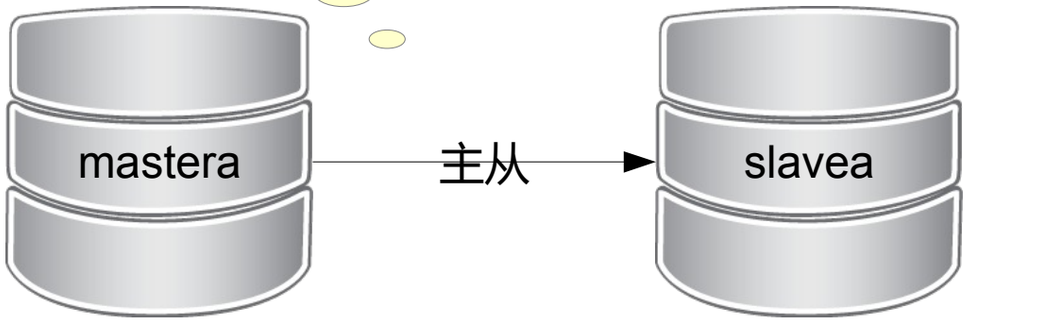
172.25.0.13
Mariadb-server 5.5

Master:
[mysqld]
server-id=1
log-bin=/var/lib/mysql-log/mastera
gitd_mode=on
enforce_gtid_consistency=1

Slave:
[mysqld]
server-id=2
gtid_mode=on
enforce_gtid_consistency=1

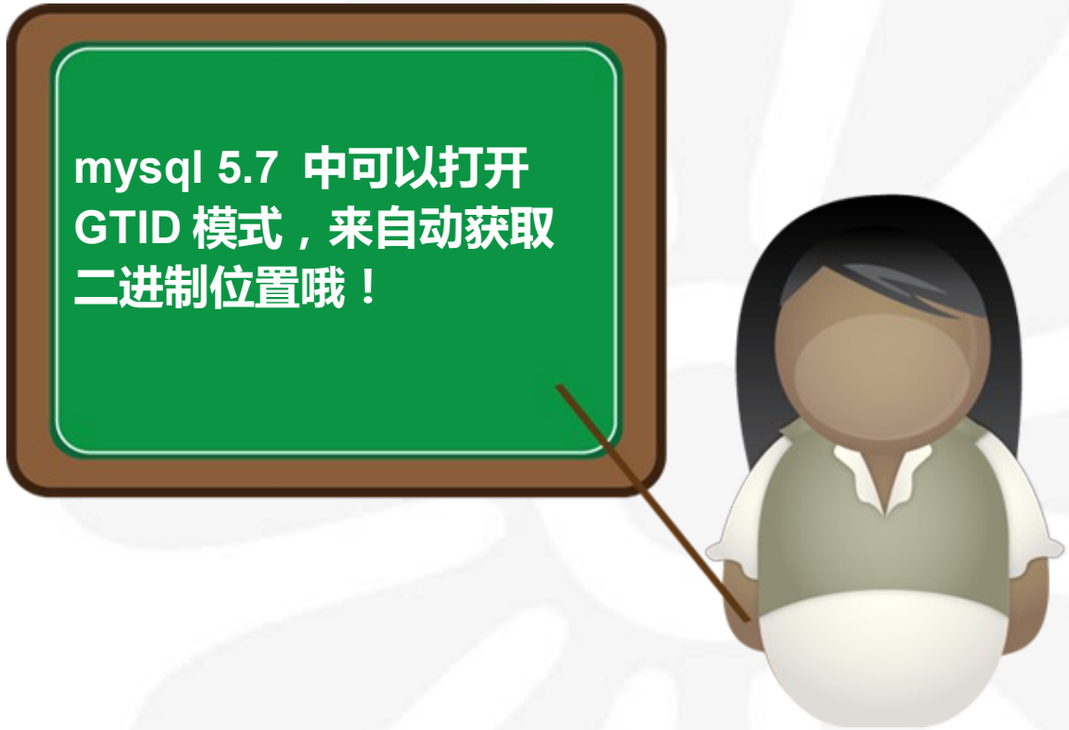
项目实战 1 ： mariadb server 5.5 单主从架构

搭建过程中， slave 需要手动获取 master 的二进制日志的位置，是否能自动获取呢？

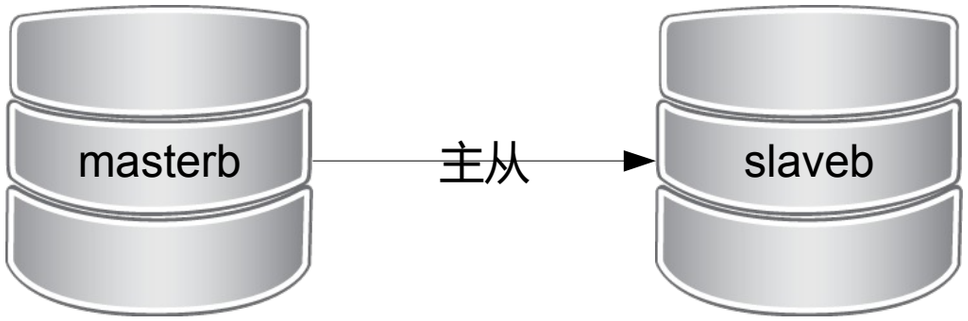


172.25.0.11
Mariadb-server 5.5

172.25.0.13
Mariadb-server 5.5



项目实战 2 ： mysql server 5.7 基于 GTID 的单主从架构



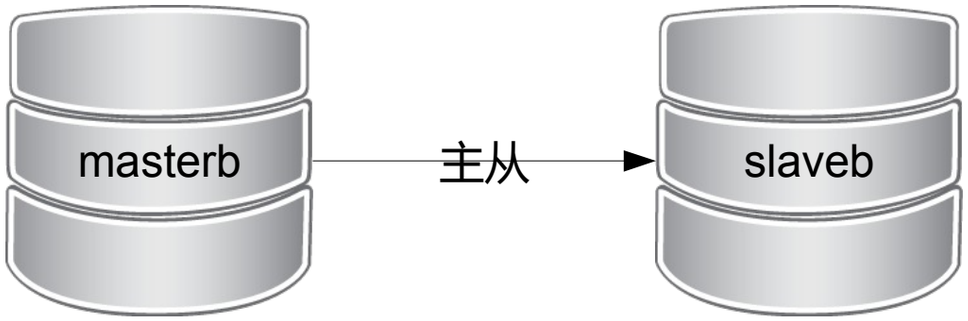
172.25.0.12
Mysql-server 5.7

172.25.0.13
Mysql-server 5.7

Master:
[mysqld]
server-id=1
log-bin=/var/lib/mysql-log/masterb
gitd_mode=on
enforce_gtid_consistency=1

Slave:
[mysqld]
server-id=2
gtid_mode=on
enforce_gtid_consistency=1

项目实战 2 ： mysql server 5.7 基于 GTID 的单主从架构



172.25.0.12
Mysql-server 5.7

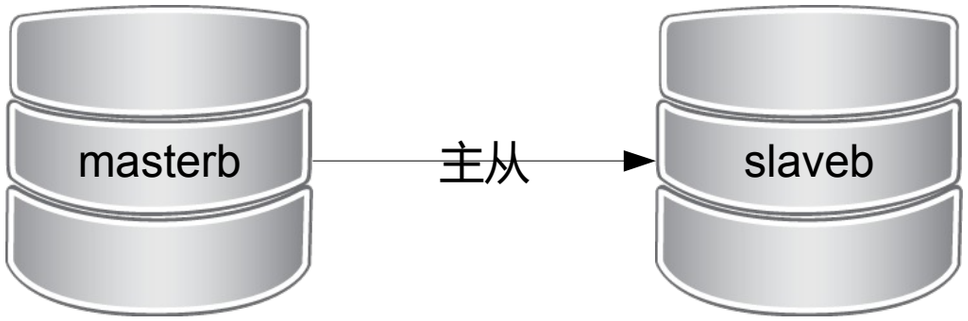
172.25.0.13
Mysql-server 5.7

Replication 的延迟问题（一）

- mysql5.5 之前没有解决方案——单线程
- mysql5.6（ mariadb10 ）开始——一库一线程
- mysql5.7（ mariadb10.1 ）真正解决了——一组一线程

Replication 的延迟问题

项目实战 3 ： mysql server 5.7 基于 GTID 的并行 MTS 单主从架构



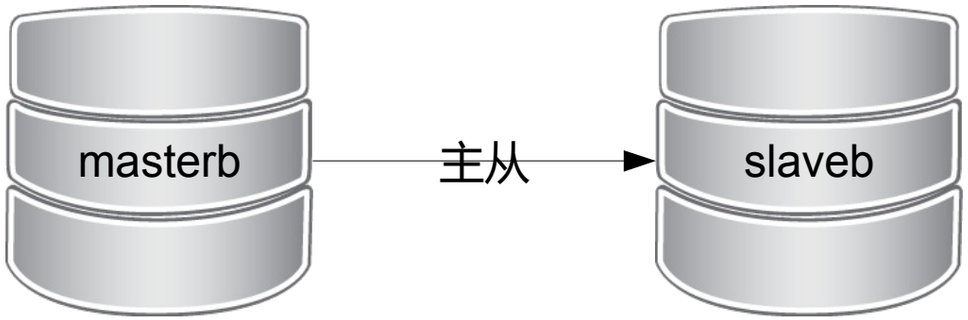
172.25.0.12
Mysql-server 5.7

172.25.0.13
Mysql-server 5.7

Master:
[mysqld]
server-id=1
log-bin=/var/lib/mysql-log/masterb
gitd_mode=on
enforce_gtid_consistency=1

Slave:
[mysqld]
server-id=2
gtid_mode=on
enforce_gtid_consistency=1
#MTS
slave-parallel-type=logical_clock
slave-parallel-workers=16

项目实战 3 ： mysql server 5.7 基于 GTID 的并行 MTS 单主从架构



172.25.0.12
Mysql-server 5.7

172.25.0.13
Mysql-server 5.7

MTS 参数是在 slave 上设置：
slave-parallel-type=logical_clock
slave-parallel-workers=16

salve-parallel-type 的选项有：
1. logical_clock　一组一线程
2. database　一库一线程

项目实战 4 ： mysql server 5.7 基于 GTID 的并行 MTS 单主从架构 crash safe 参数调优

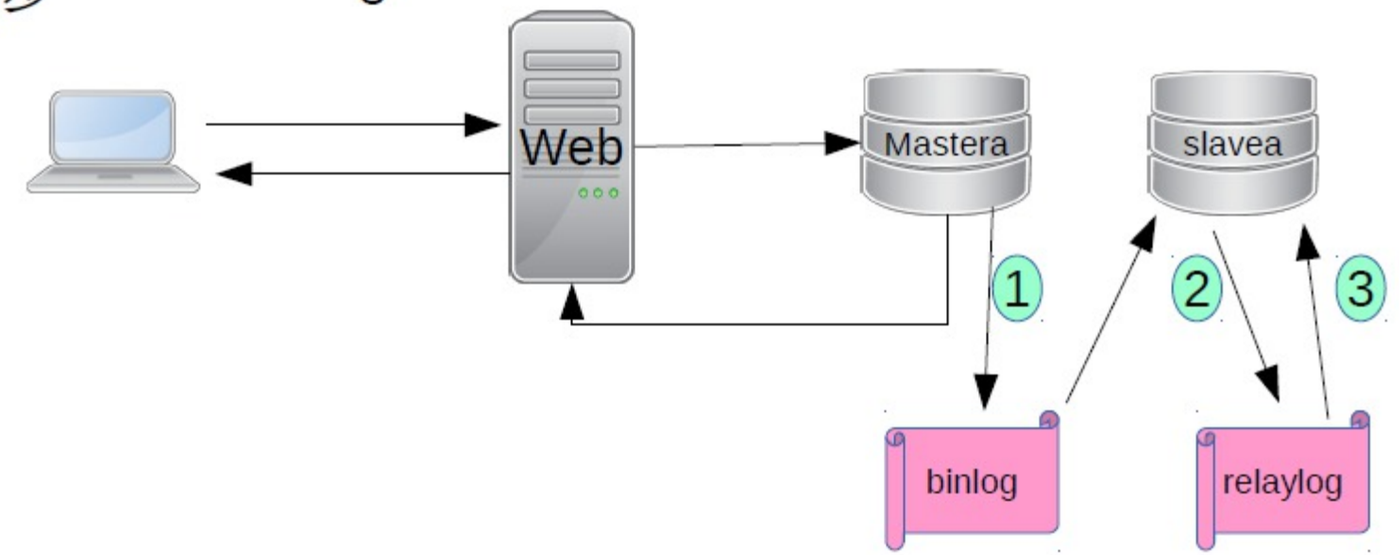
Master
sync_binlog=1 # 强制刷新 binlog 到磁盘
innodb_flush_log_at_trx_commit=1 # 强制刷新 redolog 到磁盘

Slave
relay_log_recovery=1 # 如果 slave 的中继日志出问题，能够再次自动获取 master 的二进制日志

Replication 的延迟问题（二）

读写分离导致的延迟：

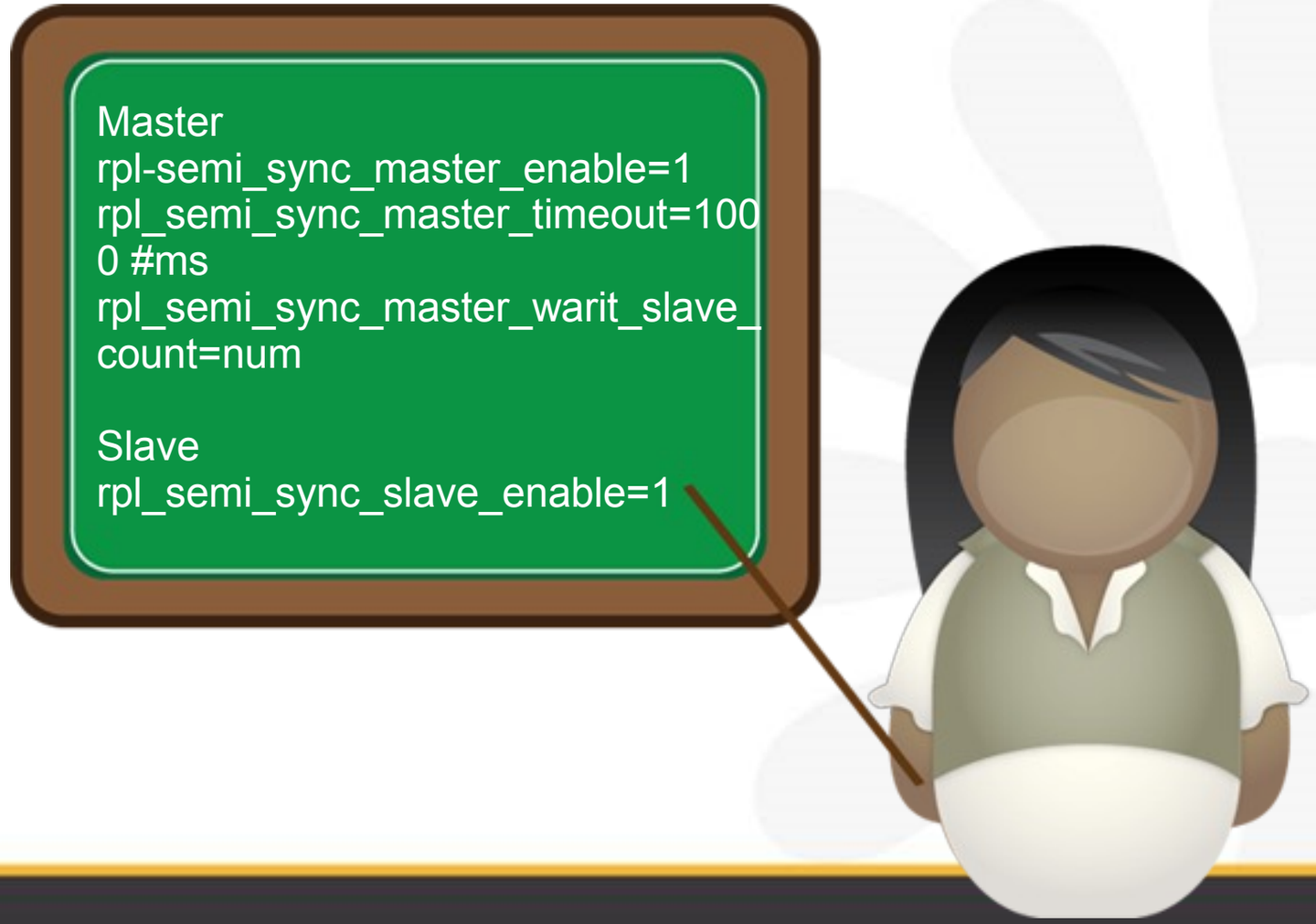
异步	1
半异步、半同步	2
同步	3

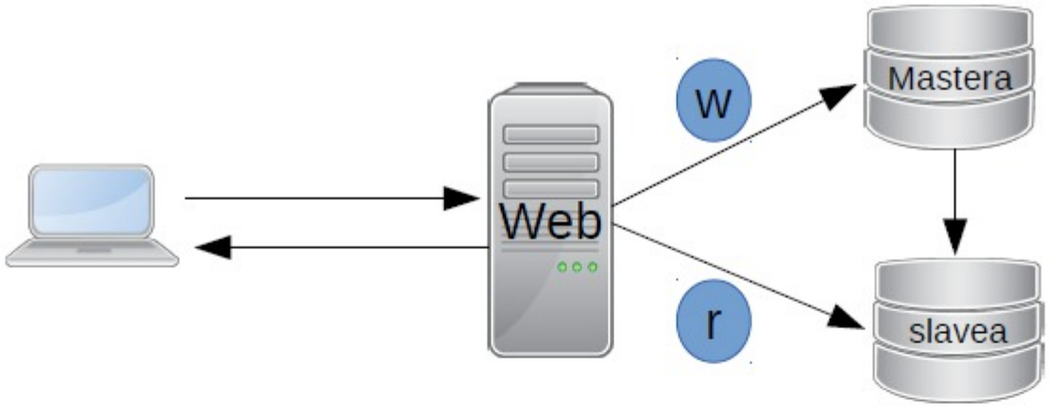


Replication 的延迟问题（二）

读写分离的方式	定义	缺点	适用场景
异步	client 写完后，Master 立刻告知用户，操作成功	容易丢数据	适用于不是金钱的
同步	client 写完后，Slave 重演完成后，Master 再告知用户操作成功	性能低	银行结算适合
半同步	client 写完后，只要 Master 的日志过去了，Slave 是否重演不关心，Master 就会告知用户写操作完成	基于异步与同步之间	
常用的是异步和半同步			

项目实战 5 ： mysql server 5.7 基于 GTID 的并行 MTS 单主从半同步架构





修改配置文件之前需要安装半同步模块

sql> install plugin rpl_semi_sync_slave soname 'semisync_slave.so';

mysql> install plugin rpl_semi_sync_master soname 'semisync_master.so';

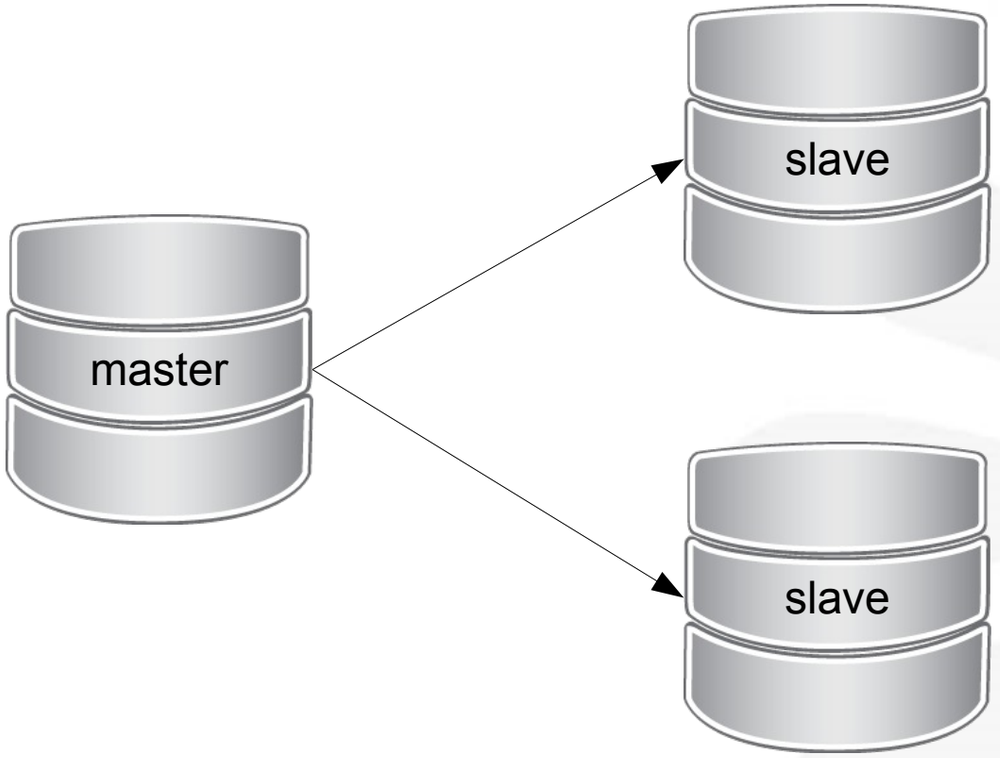
Master:

```
[mysqld]
server-id=1
log-bin=/var/lib/mysql-log/masterb
gtid_mode=on
enforce_gtid_consistency=1
sync_binlog=1 # 强制刷新 binlog 缓冲到磁盘
innodb_flush_log_at_trx_commit=1 #redolog 同上
# sync mode
rpl_semi_sync_master_enabled=1
rpl_semi_sync_master_timeout=1000 #ms
```

slave ：

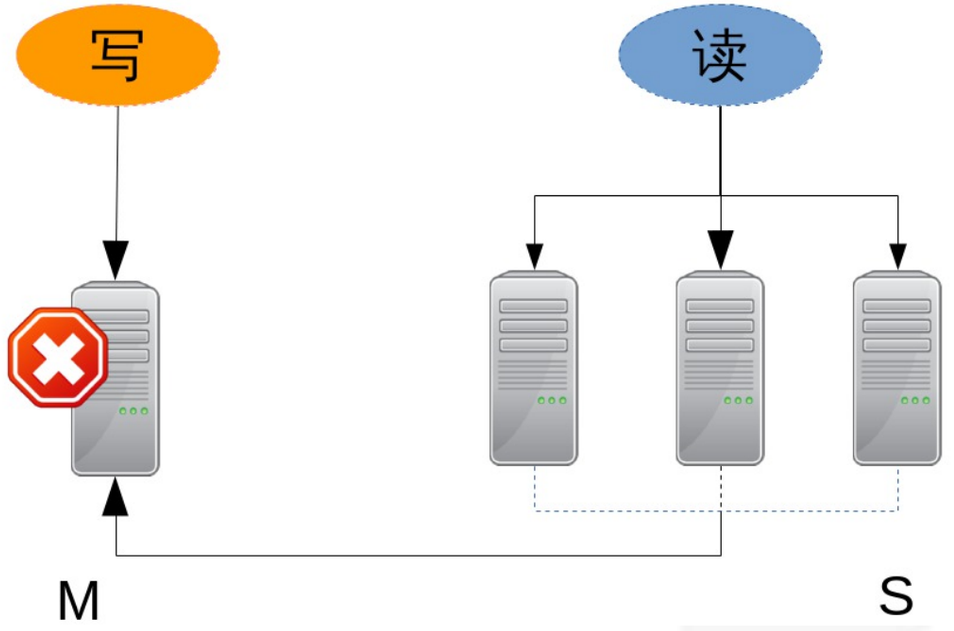
```
[mysqld]
server_id=2
gtid_mode=on
enforce_gtid_consistency=1
relay_log_recovery=1
master-info-repositry=table
relay-log-info-repositry=table
# MTS
slave-parallel-type=logical_clock #database
slave-parallel-workers=16
# sync_mode
rpl_semi_sync_slave_enabled=1
```

Replication 的单点故障问题



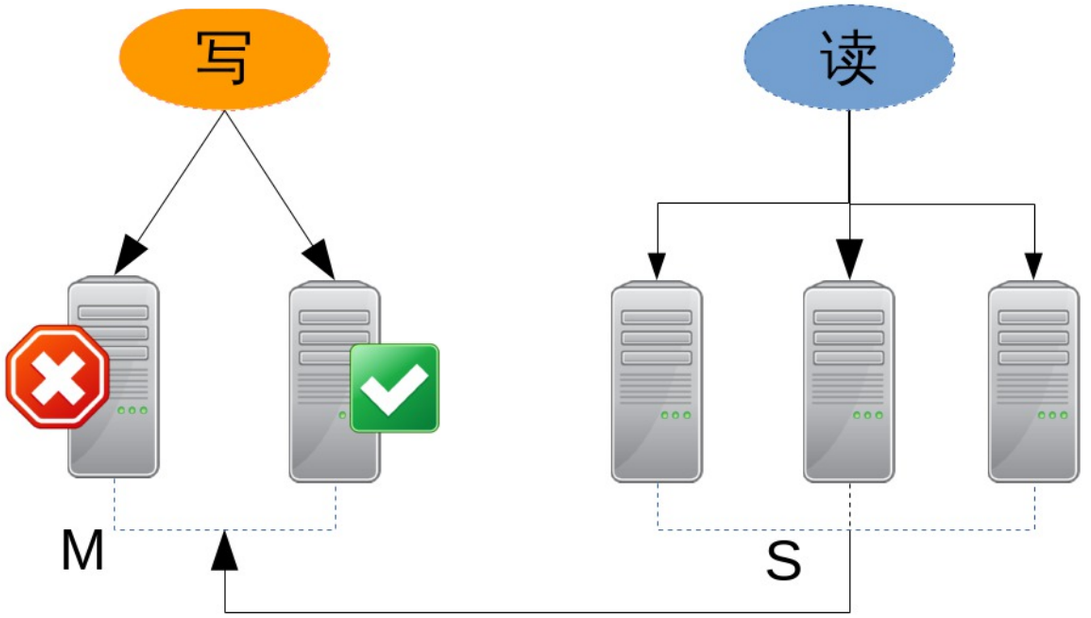
Replication 的单点故障问题

如果 master 是单点故障，如何解决？



Replication 的单点故障问题

双主
Multisource 多源复制

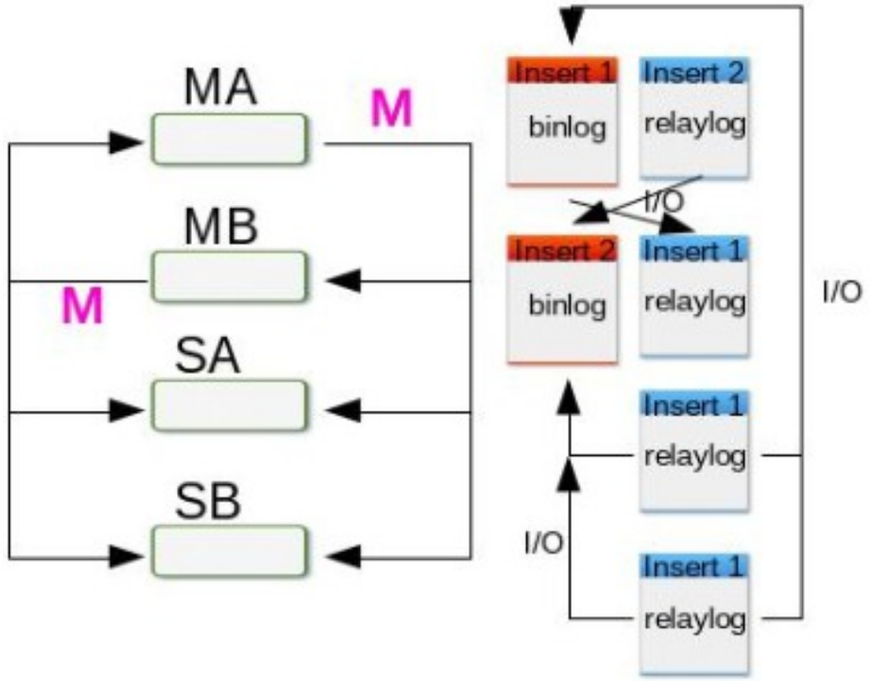


项目实战 6 ： mysql server 5.7 基于 GTID 的并行 MTS 多级主从 Multisource 架构

Multisource 多源复制
让一台 slave 有多个 master

只需要在 slave 上配置如下参数：
master_info_repository=table
relay_log_info_repository=table

项目实战 6 ： mysql server 5.7 基于 GTID 的并行 MTS 多级主从 Multisource 架构



项目实战 6 ： mysql server 5.7 基于 GTID 的并行 MTS 多级主从 Multisource 架构

Master:
[mysqld]
server-id=1
log-bin=/var/lib/mysql-log/masterb
 # GTID
gitd_mode=on
enforce_gtid_consistency=1
 # crash safe
sync_binlog=1 # 强制刷新 binlog 到磁盘
innodb_flush_log_at_trx_commit=1 # 强制刷新 redolog 到磁盘
 # semi sync
rpl-semi_sync_master_enable=1
rpl_semi_sync_master_timeout=1000 #ms
rpl_semi_sync_master_warit_slave_count=num

Slave:
[mysqld]
Server-id=2
 # GTID
gtid_mode=on
enforce_gtid_consistency=1
 # MTS
slave-parallel-type=logical_clock
Slave-parallel-workers=16
 # crash safe
relay_log_recovery=1
 # semi sync
rpl_semi_sync_slave_enable=1

Rreplication 总结

- 重点掌握
  - 1. 主从同步的原理以及同步过程中的延迟和数据一致性问题
  - 2. mariadb5.5 和 mysql5.7 的主从同步步骤
- 难点
  - 1. 同步过程中的延迟问题和数据一致性问题

1.7 MySQL 高可用 HA

什么是高可用？
如何实现高可用？

什么是高可用

高可用性实际上有点像神秘的野兽。它通常是以百分比表示，这本身也是一种暗示：高可用性不是绝对的，只有相对更高的可用性。 100% 的可用性是不可能达到的。可用性的 “ 9” 规则是表示可用性目标最普遍的方法。你可能也知道， “ 5 个 9” 表示 99.999% 的正常可用时间。换句话说，每年只允许 5 分钟的宕机时间。对于大多数应用这已经是令人惊叹的数字，尽量还有一些人试图获得更多的 “ 9” 。

每个应用对可用性的需求各不相同。在设定一个可用时间的目标之前，先问问自己，是不是确实需要达到这个目标。可用性每提高一点，所花费的成本都会远超之前；可用性的效果和开销的比例并不是线性的。需要保证多少可用时间，取决于能够承担多少成本。

高可用实际上是在宕机造成的损失与降低宕机时间所花费的成本之间取一个平衡。换句话说，如果需要花大量金钱去获取更好的可用时间，但所带来的收益却很低，可能就不值得去做。总的来说，应用在超过一定的点以后追求更高的可用性是非常困难的，成本也会很高，因此我们建议设定一个更现实的目标并且避免过渡设计。幸运的是，建立 2 个 9 或 3 个 0 的可用时间的目标可能并不困难，具体请款取决于应用。

如何实现高可用

- 降低故障率
- 优化架构

降低故障率

导致宕机的原因

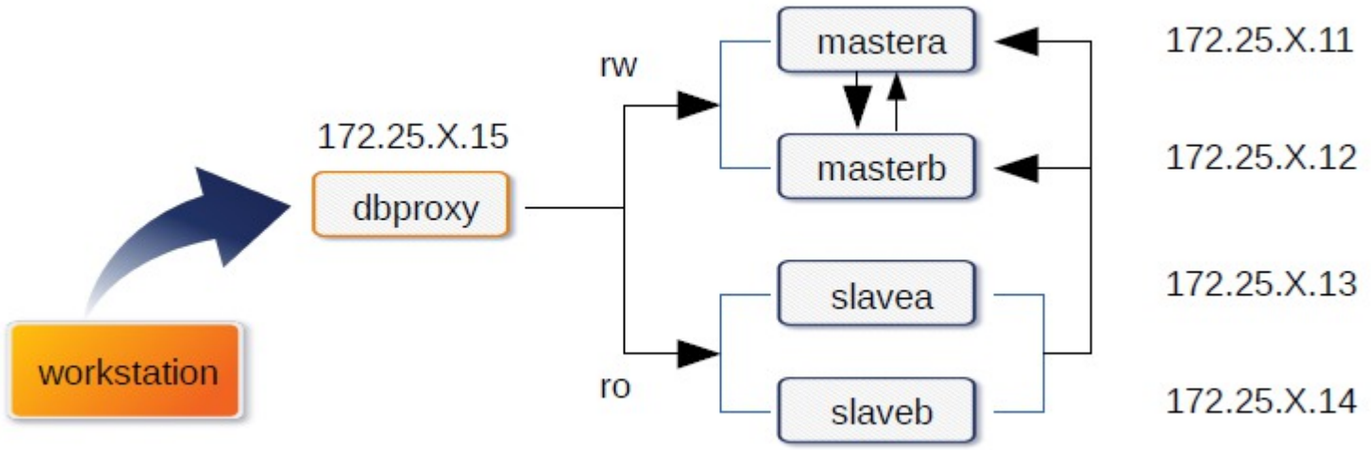
- 在运行环境的问题中，最普遍的问题是磁盘空间耗尽。
- 在性能问题中，最普遍的宕机原因确实是运行很糟糕的 SQL ，但也不一定都是这个原因，比如也有很多问题是由于服务器 Bug 或错误的行为导致的。
- 糟糕的 Schema 和索引设计是第二大影响性能的问题。
- 复制问题通常是由于主备数据不一致导致。
- 数据丢失问题通常由于 DROP TABLE 的误操作导致，并总是伴随着缺少可用备份的问题。

优化架构

- 避免单点故障
  - 基于复制的冗余
  - MySQL 同步复制
    - MySQL Cluster 集群
    - Galera Cluster 集群
- 故障转移和故障恢复
  - 数据库代理
    - Mysql-proxy
    - mycat

项目实践 1 ：搭建 4 个节点的 Mariadb 10.2 Galera Cluster 集群

项目实践 2 ：数据库中间件 Mycat 实现 读写分离负载均衡，后端数据库服务器为 MySQL 5.7 M-M-S-S Multisource Replication 高可用架构



项目实践 3: 数据库中间件 Mycat 实现 读写分离负载均衡，后端数据库服务器为 Mariadb 10.2 Galera Cluster 同步复制高可用架构

高可用总结

- 1. Galera Cluster 是 MariaDB 的一个双活多主集群 , 其可以使得 MariDB 的所有节点保持同步 ,Galera 为 MariaDB 提供了同步复制 (相对于原生的异步复制), 因此其可以保证 HA, 且其当前仅支持 XtraDB/InnoDB 存储引擎 (扩展支持 MyISAM), 并且只可在 Linux 下使用。
- 2. mariadb10.2 的 galera 版本和之前版本有所区别，需要注意
- 3. mysql-proxy 还没有正式版，问题比较多，安装需要源码编译，使用 lua 脚本实现读写分离；
- 4. mycat 是目前开源里面做的最好的数据库代理服务器， tar 包解压到制定目录就能使用，但是注意需要安装 jdk ，并宣告 java 家目录的位置。
- 5. 高可用还有其他方案，例如 MHA 、 LVS 等，可以在高级课程中再学习！