MySQL备份、安全、SQL规范与系统规划

2017-07-21 22:12 | 阅读 5k | 评论 0

讲师介绍

> **韩成亮**
>
> 资深DBA
>
> 拥有7年DBA实战经验。目前从事MySQL相关运维及架构工作，擅长MySQL及Oracle设计和调优。

主题简介：

1、MySQL之备份
2、MySQL之安全
3、MySQL之SQL规范
4、MySQL之系统规划

一、MySQL之备份

之所以开头就提这个，主要原因是最近的事故略多，删主机、删库、删表、删字段还有勒索病毒等，太多的不可控因素了，从“删XX到跑路”你缺少的只是一个“机会”。本来是你好、我好、大家好，但是万一呢？所以要未雨绸缪，当然备份的意义不仅仅在于此。

1、可恢复性

首先谈谈备份的可恢复性，或者说是有效性。无论你的备份是在本地、异地、存储、磁带、云上、或者其它不知名处，如果不知道出啥问题丢失了，而且还必须要用这份文件恢复，不能有效的恢复，后果会很严重，所以在系统设计之初就需要规划好。

一般而言我们同一份文件需要存放多个位置，三份是个不错的选择。常规的做法是本地一份，其它地方两份，具体的可以根据实际条件酌情处理，通常一份的效验很难有说服力，而且也并不可靠，一定要确保多份副本，从技术的角度而言，奇数是常用的仲裁配置，而三是最小的单位。

接下来就是需要经常验证，我这边说的是经常，日常不是很现实，如果资源足够，那就完全可以实施了，这样可以做到心里有底，避免真正在生产执行恢复的时候手忙脚乱，而且主要是可以通过预先准备的脚本或者命令，快速恢复。之所以一定要通过实践去检验，因为通过实际的行为去排雷、排坑、排问题，尽可能地摸清楚将来可能会遇到的可能性

问题。虽然可能性为零，不过平时多演练，这样在真正需要的时候才能少掉坑里去。

如果说因为各方面的原因你没法去验证，只能通过系统自带的验证或者采取其他的诸如文件效验的方式，这样回到之前说的多副本情况，如果你拥有多分副本，起码验证起来还是比较简单的，但是也并不是说多份副本就一定是有效的，还有时效性等问题。

2、时效性

这是建立在前面备份可恢复性基础上，首先你的备份是可以支持恢复的，这个接下来需要关注备份的有效性，恢复的效率性。这里面会涉及到RPO、RTO，两者是相互耦合的，RTO要求你越少的时间，RPO要求你恢复到最近时间点的数据，无论哪个方面，有效的解决方案都是更新的备份，更快的恢复效率，最好乃至于瞬间恢复到上一个事务，当前事务。不同的业务级别，需要不同的级别，当然灾备另说。

时效性，说实话是属于比较难界定的。那具体需要恢复到什么时间节点的数据呢？根据相应的RPO级别，需要指定相应的计划，而且这其中还需要考虑RTO，如何缩短时间。如果说真的发生了需要从备份恢复的场景，首先需要确定你的数据库拥有完整的备份，但是往往这个时候可能没有最新的，要么就是最近的备份是不成功或者失败，不要觉得我在危言耸听。

事实往往比这个更复杂、更加严重，如果幸运的是，你不仅仅拥有多份可靠的而且是最新的备份，还有完美得是到宕机前一刻的增量或者日志，接下来就是跟时间赛跑，稳稳地减少宕机时间。

3、安全性

安全性放在最后，并不是说不重要，相反，主要高度依赖以上两点，如果脱离了，就片面而言，没有备份，那就是安全的，这是在没有的情况下，不存在其它的情况。

然后就轮到有备份了，当你有了备份，你不仅仅需要保证备份的多副本，还要保证其安全性，这里面的安全性包括内部安全，外部安全。很多时候外部反而是安全的，一个有效的备份经历加密、访问控制，别人无论是获取还是解密都需要花费大量时间的，而内部往往更容易出问题，千里之堤，毁于蚁穴。

提一下泄密，这个还是比较尴尬的，不论等级划分，笼统来讲，被权限以外的人接触都算，那具体怎么界定，而且权限也是比较难划分的，所以该有的预防措施必须要有。

安全无小事，相应的规定章程制定下来就需要严格的执行，剩下的这就需要看从业者的职业操守。

PS：如果需要了解备份恢复可以关注下社群文章《解锁MySQL备份恢复的4种正确姿势》，相信可以帮到你。

二、MySQL之安全

在生产中，安全是相当重要的，毕竟你的核心数据都在里面，MySQL因为其开源的流行性，大量个人、企业、政府单位都采用，但是很多在部署的时候采用都是默认的配置，这就导致了安全的相对欠缺，你需要针对你的安全有所加强。总的来说，数据库一般划分为生产库、压测库、准生产库、测试库、开发库。下面部分主要说的是生产库，但其它库也适用。

1、 mysql_secure_installation

这是数据库基础的安全设置脚本：

- 设置root密码
- 移除匿名用户
- 禁止远程root登录
- 移除test数据库

以上是5.6版本，5.7有所加强但也仅此而已，看看你的环境是否存在上述问题，这个算是最基本的安全吧。

2、连接访问安全

常见创建用户时你需要指定你的IP访问地址范围或者固定IP，一般而言，只有特定唯一的几个IP才会访问，或者说你可以采用代理访问的方式，减少应用直接访问你的数据库，而且现在很多中间件也都有白名单机制，原则上是把非法请求防止在数据库以外的地方。

规范数据库管理软件，实现管理软件的标准、统一化，还有严禁杜绝开启外网访问，如果客户端在远程，就根本不应该直接访问数据库，而应该使用中间件堡垒机或其它替代方案。

为了防止连入数据库的应用程序存在后门，造成数据库安全隐患，检查所有连接数据库程序的安全性。通过使用堡垒机或者其它监控登录数据库，禁止对数据库的直接操作。

对已经连接的IP网段进行规范化、统一化的管理，定期进行权限复核操作，对系统所属IP、用户进行权限梳理工作。

对员工进行安全培训，增强员工的系统安全观念，做到细心操作，安全操作。确保访问数据库的主机为已知用户或者主机，使用专门主机与数据库进行连接。

对重要业务表的所有行为全部审计，审计同时所有包括即使是DBA的DDL操作行为。

3、权限安全

权限这块无可厚非，在建立之初遵循最小权限原则，坚持最小权限原则，是数据库安全的重要步骤。

以上说的是白话，下面说说正题。

很多时候我们不知道具体的最小权限是什么，你说一个账号到底需要什么样权限才合适，才不会影响业务？这个不是很好界定。我们需要知道在设置权限时的信息，要授予的权限级别、库级别、表级别、列级别，或者其它超级权限、要授予的权限类型，增删改查等。
从mysql.user表来看

```
Select_priv/Insert_priv/Update_priv/Delete_priv/Create_priv/Drop_priv
Reload_priv/Shutdown_priv/Process_priv/File_priv/Grant_priv/References_priv
Index_priv/Alter_priv/Show_db_priv/Super_priv/Create_tmp_table_priv/Lock_tables_priv
Execute_priv/Repl_slave_priv/Repl_client_priv/Create_view_priv/Show_view_priv/Create_routine_priv/
Alter_routine_priv/Create_user_priv/Event_priv/Trigger_priv/Create_tablespace_priv
```

4、账户安全

用户账户划分原则：

- 超级管理员账号
- 系统应用账号（比如备份，监控，审计等）
- 应用业务账号
- 业务人员账号
- 开发人员账号
- 测试人员账号
- 其它专用账号

主要是防止泄漏，非必要人员不需要知道账号的名称，同时需要制定相应的命名规则，还有就是合理使用自己的账号密码，保护好你的账号密码，对于绝无必要的用户，先禁用，后期删除，要做到无匿名账户和无废弃账户。

5、目录文件安全

提高本地安全性，主要是防止MySQL对本地文件的存取，会对系统构成威胁，还有Load DATA LOCAL INFILE，禁用该功能。

这个主要是防止误删除，非权限用户禁止访问目录，还有就是数据文件禁止访问，或者采用更改常用的目录路径，或者通过chroot，要保证该目录不能让未经授权的用户访问后把数据库打包拷贝走了，所以要限制对该目录的访问。

- 密码强度复杂性

尽量并且不要使用固定密码，实行每个用户单独密码，长度在16位以上 0-9a-zA-Z~!@#$%^&*()-+ 随机组合。

- 密码过期机制

根据公司的情况设定密码过期时间，定期更改，同时不可使用重复密码。

- 密码保存机制

为了方便管理，可能会采用一个密码表，要加强对于密码表的维护更新，最重要的是保证不泄漏。

7、漏洞安全

常规的方式是安装补丁，不过这个往往比较麻烦，主要是版本升级，还有就是防护策略。

8、被忽视的SSL

由于性能或者其它方面原因，很多生产环境并没有使用，不过从5.7+开始，已经好很多了，有需要的加强安全防范其实可以尝试下了。
https://dev.mysql.com/doc/refman/5.7/en/mysql-ssl-rsa-setup.html

9、防火墙安全

一般化数据库前面都会有主备的墙，不过从成本上考虑，很多企业都是单个或者裸奔的，有自己的硬件防火墙最好，没有的话也可以使用系统自带的防火墙，然后在加上其它白名单和中间件白名单过滤辅助措施，也能防止一部分问题。

10、端口安全

默认端口是3306，这个最好修改下，为了方便记忆，你可以根据的IP地址来加密动态调整，不过如果生产网络允许，也可以定期修改，最好不要影响研发进度。

11、记录安全

删除操作系统记录的敏感数据，比如.mysql_history、.bash_history 等，及时清理，移除和禁用.mysql_history文件。

人是安全的主导，管理的对象要从两个角度来看，从信息角度来说就是MySQL本身的安全，要防止数据的丢失和免遭破坏；从技术角度来说就是整个系统的安全，要防止系统的瘫痪和免遭破坏。

最后说句题外话，监控和审计，安全主要是防患于未然，没有谁希望一天到晚接到各种警报，最好根据公司的实际情况订个详细的规章制度，不要觉得这个麻烦，有些你可能并不觉得有用，但是呢？我希望是没有但是。

三、MySQL之SQL规范

不以规矩，不成方圆。

无可否认，很多时候由于项目的开发迭代过于频繁，实时的需求反馈，可以及时调整产品的方向，不过由于各种大大小小的功能的耦合交错，还有研发人员对数据库的了解参差不齐，他们可能更加关注的是功能的实现，其次可能才是响应速度，这就导致了由于数据量小时看不出问题的SQL，一旦遇到大数据量，查询性能或者说系统的响应速度会变慢，这是个值得关注的地方，为了防止出现这种情况，你需要做点什么。

1、使用什么

- 使用 InnoDB 存储引擎，默认使用utf8mb4 字符集
- 使用自增主键，尽量每个表都有而且是非业务键值
- 使用合适的字段类型，比如 VARCHAR 代替 CHAR 等，最好确定好长度
- 使用尽可能小的 VARCHAR 字段
- 使用合适的 INTEGER、INT、SMALLINT、TINYINT、MEDIUMINT、 BIGINT 数据类型。https://dev.mysql.com/doc/refman/5.7/en/integer-types.html
- 使用 UNSIGNED 存储非负数值，比如自增等
- 使用 INT UNSIGNED 存储 IPV4 INET_ATON、INET_NTOA。更多 https://dev.mysql.com/doc/refman/5.7/en/miscellaneous-functions.html
- 必须使用 DECIMAL 代替 FLOAT 和 DOUBLE 存储精确浮点数，比如与货币、金融相关的数据，防止运算丢失精度问题
- 使用注释，所有表都需要添加注释；除自增主键外的其他字段都需要增加注释
- 使用默认值，所有字段都必须拥有默认值，不在表中存储 NULL
- 使用 PREPARED STATEMENT，防止SQL注入
- 使用合理的 LIMIT 分页方式以提高分页效率
- 使用 EXISTS 适用于子查询不返回实际数据，而表较大的情况，如果子查询表较少可以考虑用in
- 使用 IN 代替 OR，同时 SQL 语句中IN包含的值不应过多，应少于1000个，否则使用转为字符串LIKE
- 使用 UNION ALL代替 UNION，减少不必要的去重消耗
- 使用 COUNT(1) 统计行数，如果使用字段的话可能存在空值或NULL不准的情况
- 使用 INDEX，所有 WHERE 条件必须有索引，特别是 DELETE / UPDATE

- 使用 EXPLAIN EXTENDED 判断SQL语句是否合理使用索引
- 使用 SHOW PROFILES 跟踪资源使用
- 使用事务，增删改必须有，程序应有捕获SQL异常的处理机制
- 使用从库，不是必要的查询一律使用从库查询，减轻主库压力

2、减少什么

- 减少不必要的固化查询，合理使用 Memcached、Redis、MongoDB等
- 减少不需要的空连接，及时关闭
- 减少不必要的连接，尽量采用批量SQL语句
- 减少不必要的大事务，尽量做拆分
- 减少查询的数据量，大量查询分批次
- 减少游标和临时表的使用，如果有的话

3、避免什么

- 避免使用TEXT、BLOB类型等
- 避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理
- 避免使用子查询 IN ，必要时使用JOIN。
- 避免出现NOW()、RAND()、SYSDATE()、CURRENT_USER()等不确定结果的函数
- 避免在索引列上使用IS NULL和IS NOT NULL
- 避免同一个表上索引过多，重复索引，太多的索引可能提高了查询效率，但是也增加了增删改的开销

4、禁止什么

- 禁止在数据库中存储图片、文件等大数据
- 禁止使用order by rand()，类似的有很多替代方案
- 禁止使用SELECT *，根据需要获取相对应需要的字段
- 禁止使用预留字段，无论从流程和规范上都说不过去
- 禁止在MySQL中进行数学运算和函数运算
- 禁止隐式转换，数值类型禁止加引号，字符串类型必须加引号
- 禁止使用 INSERT INTO TABLE()，需要指定相应的字段，最好是顺序的
- 禁止使用前置百分号，例如：WHERE A LIKE ‘%B’
- 禁止使用 NOT 查询，例如 NOT IN、!=、NOT LIKE
- 禁止单条SQL语句同时更新多个表

- 禁止使用存储过程、触发器、视图、自定义函数等
- 禁止在从库上执行负责统计类查询，使用专用从库
- 禁止在数据库中存储明文密码

5、注意什么

- 合并对于单表多次的操作
- 表结构变更必须通知DBA审核参与
- 重要项目的数据库方案选型和设计必须提前通知DBA参与
- 禁止在线上做数据库压力测试
- 禁止从测试、开发环境直连线上数据库
- 禁止有超级权限的应用程序账号存在
- 禁止有DDL、DCL权限的应用程序账号存在
- 批量导入、导出数据必须通过DBA审核，并在执行过程中观察服务
- 批量更新数据，如UPDATE、DELETE操作，必须DBA进行审核，并在执行过程中观察服务
- 产品出现非数据库导致的故障时，如被攻击，必须及时通DBA，便于维护服务稳定
- 业务部门程序出现BUG等影响数据库服务的问题，必须及时通知DBA，便于维护服务稳定
- 业务部门推广活动或上线新功能，必须提前通知DBA进行服务和访问量评估，并留出必要时间以便DBA完成扩容
- 出现业务部门人为误操作导致数据丢失，需要恢复数据的，必须第一时间通知DBA，并提供准确时间点、 误操作语句等重要线索
- 提交线上建表改表需求，必须详细注明涉及到的所有SQL语句(包括INSERT、DELETE、UPDATE)，便于DBA进行行审核和优化

以上相对比较常规，一个完善合理的规范当然还需要一个比较长的过程。

四、MySQL之系统规划

1、环境规划

MySQL作为流行的开源数据库拥有多个重要分支，每个分支都有各自的优缺点，这里不做过多评价，总的来说，MySQL仍然是一款非常出色的产品，是一个非常适合大多数场景下使用的数据库，无论是从可用性、可扩展性、性能和管理各方面都是不错的选择。

当然说为了追求某些方面的新的功能性需求，或者MySQL版本觉得太臃肿，你也可以尝试其他版本，主要是符合你的业务场景才是最合适的。

根据研发的阶段来划分主要分为开发阶段、测试阶段、准生产阶段、压测阶段、上线阶段。

- 开发阶段-开发环境

数据库和系统的版本可能都比较新，这里面有一部分是尝鲜的概念在里面，同时，也为了以后的版本更新提供了一定的基础，其次是研发人员对于数据库和操作系统的权限是比较大的，基本上会all in，主要保证开发的进度。

- 测试阶段-测试环境

这个环境的版本可能也是比较新的，不过已经很贴近生产了，然后是对于权限的话，由于主要是测试阶段，研发人员和测试人员的权限基本上限定到库表的DML权限，很难执行其它特别是DDL操作了，当然还是以保证开发和测试为主。

- 准生产阶段-准生产环境

原则上这个环境跟生产环境基本上1：1，版本跟生产是一样的，其次配置比生产会低很多，权限的话，已经不允许DDL语句了，并且DML也会严格控制。

- 压测阶段–压测环境

一般化而言，很多公司会把准这个压测环境放到准生产上面，对此，我不反对不建议，完全看你的想法和公司的规划，压测环境跟生产环境保证100%完全一致性，因为为了追求那个极致，同时全面禁止的DDL，只读环境。

PS：很多时候，你的准生产可能就是压测环境，这边就不做过多说明了，主要还是看你的规划。

- 上线阶段–线上环境

这个就不说了，完全禁止的DML、DDL操作，任何操作必须有记录，比如审计、工单等等。

2、容量规划

容量规划主要是如何有效评估需求，如果说没有统一综合的管理机制，各种项目的资源申请各自为政，没有考虑到综合实际使用的资源情况，会造成严重的分配问题，或者说资源不足，有可能部分项目的提前预支超过实际情况的资源，其它项目完全没有资源，或者项目资源是有时间期限的，过了期限就需要即使的回收利用。

对于数据库本身容量规划更加重要，毕竟你的数据都在里面，首先项目初始阶段可能会预估下使用量，然后还需要定期的实际评估，根据资源的使用情况及时调整，综合规划各个项目的资源配置。

MySQL的容量规划主要是库表的大小，这个主要是看业务量，很多时候业务部门会说我的计划目标是1kw用户量，那么可能很难有一个估计的大小，因为你的库表一直是在变化的，自然而言，比较困

难评估一个大概的容量，而且如果数据量太大的情况下，从数据的访问响应速度就需要考虑分库分表了，这个主要还是看的业务场景和需要的响应速度。

3、文件规划

MySQL数据文件目录还是比较简单的，可以分成数据文件目录、日志文件目录、参数文件目录、其它文件目录等。

- 数据文件目录

数据文件主要按照库表单位划分的，由于引擎的不同可能分成结构定义文件、数据文件、索引文件。

- 日志文件目录

这里面包括了启动日志、报错日志、查询日志、慢查询日志、binlog日志、binlog索引文件、relay_log中继日志、relay_log中继日志索引文件等。

- 参数文件目录

包括pid、sock、cnf 文件等。

- 其他文件目录

包括redo、undo、ibdata1、ibtmp1文件等。

归档，这边说的归档规划是定期把以上各类文件备份归档，数据文件自不必说，其它文件也是需要定期归档的，归档的方法有很多，你可以把相应的文件根据不同的规律存放在结构化或者非结构化的地方。

4、审计规划

数据库审计能够实时记录网络上的数据库活动，对数据库操作进行细粒度审计的合规性管理，对数据库遭受到的风险行为进行告警，对攻击行为进行阻断。

它通过对用户访问数据库行为的记录、分析和汇报，用来帮助用户事后生成合规报告、事故追根溯源，同时加强内外部数据库网络行为记录，提高数据安全。

审计对数据库记录和行为进行独立的审查和估计，其主要作用和目的包括以下几个方面：

- 对可能存在的潜在攻击者起到威慑和警示作用，核心是风险评估。
- 测试系统的控制情况，及时进行调整，保证与安全策略和操作规程协调一致。
- 对已出现的破坏事件，做出评估并提供有效的灾难恢复和追究责任的依据。
- 对系统控制、安全策略与规程中的变更进行评价和反馈，以便修订决策和部署。

- 协助系统管理员及时发现网络系统入侵或潜在的系统漏洞及隐患。

有个不知道算奇怪还是正常的事情，不知道多少人给生产恢复过数据，这里面不谈因由，只有做过的人才知道这其中的坑有多少，所以也就需要大家对于各种恢复手段都有所了解，乃至于熟练掌握。

虽然防患未然还是比较困难，毕竟各种情况都会发生，但至少可以未雨绸缪，从安全上讲，有了审计就可以大大减少这类事情的发生。

对于数据库而言，无论是硬件设备还是软件审计都会加大数据库的压力，从性能的损耗上讲，事后审计是比较折中的策略，这边先讲下软件部分的，无论是MySQL官方还是MariaDB或者Percona还是其它的都有一套自己的审计产品。

下面列下常用的几种：

- MariaDB Audit Plugin(https://mariadb.com/kb/en/mariadb/about-the-mariadb-audit-plugin/),
- Percona(https://www.percona.com/doc/percona-server/5.6/management/audit_log_plugin.html),
- MySQL(https://dev.mysql.com/doc/refman/5.7/en/audit-log.html), mcafee(https://github.com/mcafee/mysql-audit)

主要用哪个还得看你适合哪个，如果你有足够的资源可以考虑采用硬件的模式，尽量选择专门审计设备的设备，然后根据定期的报表检查你的相关配置。
前面列出的几个感兴趣的小伙伴可以具体测试下，当然如果你已经在使用了，那我们可以私下交流。

5、备份恢复规划

备份和恢复是两个相互关联的，至于备份恢复的种种前面已经有说过了，关于备份恢复也有一系列的软硬件，这边主要说下规划。

你首选要根据自己的实际情况做需求分析，你需要有当前使用数据库的类型、版本信息、配置信息、数据量总大小、每日新增大小、备份方式（物理备份/逻辑备份）、业务高峰期，当然还有数据库的主机的系统情况等信息。

备份策略，根据不同的业务还有其它需要确定，常用的是全量、增量、差异备份三种，实际情况下很多都是三种策略的结合使用。

备份大小，这个取决于你采用的备份方式，会有一个大致的增长值，这个可能需要跟业务那边的规划做详细的统计，给出一个大概的范围。

备份保留时间，这个跟备份的大小、备份介质的大小、性价比有关系。

如果你的介质空间太小，自然而然也就不能保留太长时间，这个时候，通常会根据业务的核心程度来划分，应确定重要业务备份的保存期以及其它需要永久保存的备份，总的来说，保留半年之类的有效数据是基本的要求。

备份介质大小，根据你的保留时间合理地规划你的备份介质大小。

备份计划，这个跟恢复策略有关系，基本上是你需要一个专门备份的数据库，这个主要是为了减少对主库的影响，对此大部分是数据库都支持该方案，当然主库执行备份也不是不可以，只要合理使用。

根据业务繁忙的情况，在合理的时间和空间下能全备的尽量全备，虽然全量可能会增加数据的重复性和空间的使用，你可以适当加上增量或者差异备份。

太大的库可能全备时间太长，优化过后还是不能接受，可以选择一个相对不繁忙的时间做全量备份，然后加上增量或者差异备份，当然这个全量的频率还是需要根据你的业务来结合。总不能说，我需要恢复昨天的数据，你告诉我只有上上上个月的全备加上增量或者差异，这个恢复的时间一般会比较长，这个肯定不能满足的需求，所以备份计划要完全贴合你的业务需求，以及需要详细指定最大容忍的时间性要求。

备份执行过程应有详细的规划和记录，包括备份重要等级、备份主体、备份时间、备份策略、备份路径、记录介质（类型）等。

同时备份文件加密权限控制也是不可或缺的，所有操作需要保留相应记录，方便审计跟踪，安全需要时刻关注。

恢复规划，这个不能说是规划，我觉得这个应该更像一个日常的计划任务，如果你有充足的资源，特别的恢复服务器组，我希望你在每天完成备份并且上传统一的备份介质后，每天都可以把所有的系统都恢复一次，不要觉得太麻烦，有时候这些麻烦会帮你很多，当你把所有的一切做成定时任务之后，你会发现，生活太美好。

实际的情况往往比这个更加复杂，我们把容灾划分成很多的等级，计算机软件故障、人为原因、计划性停机，然后从等级上是时间上有几个九，同时还有灾备的环境一些措施，确保可以在规定的时间内完成有效性恢复。

但是，当灾难发生的时候，很多东西都会很巧合地并发发生，会很乱，假使灾备环境也不是正常的或者压根就没有所谓的灾备环境，那个时候你依赖的可能也仅仅是备份的一个有效恢复，那个时候，压力会稍微有点大，如果你对恢复情况不是很了解，或者不是很熟练，我们也只能就这么看着能恢复多少，心里没有底，这个时候才会觉得恢复测试真的非常重要。

平时我们关注的大都在日常的备份上，并不会实际去验证，不过其实日常恢复演练更加重要，即使不能做到所有系统的恢复计划，起码保证核心系统的备份的有效性，同时制定恢复测试计划。

总结

总之，对于数据库而言，首先要求我们拥有一个完整的备份，有了备份你才能做很多事，同时也会省了很多事。其次是安全，无论是数据备份的安全还是数据本身或者研发使用中的安全，都值得关注。然后是数据库本身的使用，合情合理的使用。最后对于你所维护的数据库你需要拥有详细的规划，无论是管理角度还是使用角度都需要。

最后给大家提几点建议：

1. 未雨绸缪，不要停留在制度上，而是实际做出来。
2. 亡羊补牢，举一反三，切记，不能好了伤疤忘了疼。
3. 完备的架构设计及备份，恢复策略及备份验证策略。
4. 定期思考，并实战模拟以上策略演练。
5. 实践是检验真理的唯一标准。

文章来自微信公众号：DBAplus社群