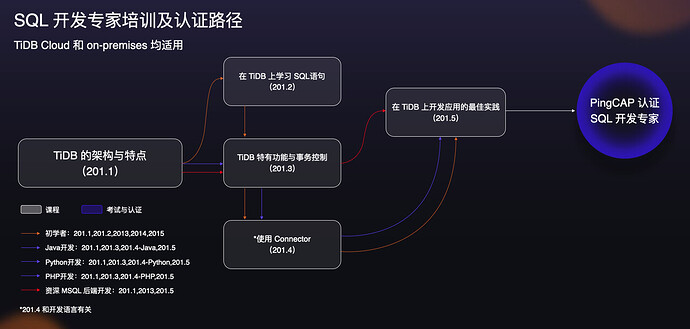
<https://asktug.com/t/topic/994111>

0.内容：

PingCAP 认证 SQL 应用开发专家考试旨在验证应用开发者掌握 TiDB 独特的功能、在 TiDB 上运用最佳实践开发高可用、高弹性的应用的开发能力。

[](https://asktug.com/uploads/default/original/4X/f/0/8/f08196ee0f39dce1e113e438feb51054a10a8697.jpeg" \o "image)

1.考题类型

问题类型分为三类: 单选题、多选题、是非题。

2.计分规则

共 60 题，每题 1 分，考试通过分数为 36 分。考试时间为 75 分钟。

3.考试范围

| 知识领域 | 百分比 |
| --- | --- |
| TiDB 架构 | 15% |
| TiDB SQL 运用 | 30% |
| TiDB 特有功能(相较于 MySQL) | 30% |
| 应用开发最佳实践 | 25% |

## 笔记

* TiDB 架构
  + 理解 TiDB 的构成组件
  + 理解 TiKV Server 的功能
  + 理解 TiDB Server 的功能
  + 理解 PD Server 的功能
  + 理解 TiFlash 的作用
  + 理解 分布式 SQL 与 TiKV 表达式下推
  + 理解 HTAP 的定义与优势
* TiDB SQL 运用
  + 运用 JOIN
  + 运用 子查询
  + 处理 NULL 值
  + 理解 字符、数字、日期时间类数据类型的特征
  + 理解字符集与排序
  + 运用不同的 SQL\_MODE
* TiDB 特有功能（相较于 MySQL）
  + 运用 AUTO\_RANDOM
  + 运用 AUTO\_INCREMENT
  + 运用 Placement Policy
  + 运用 Temporary Table

为了避免内存溢出，用户可通过系统变量 [tidb\_tmp\_table\_max\_size](https://docs.pingcap.com/zh/tidb/stable/system-variables" \l "tidb_tmp_table_max_size-%E4%BB%8E-v53-%E7%89%88%E6%9C%AC%E5%BC%80%E5%A7%8B%E5%BC%95%E5%85%A5) 限制每张临时表的大小。当临时表大小超过限制后 TiDB 会报错。tidb\_tmp\_table\_max\_size 的默认值是 64MB

临时表类型

TiDB 的临时表分为本地临时表和全局临时表：

- 本地临时表：表定义和表内数据只对当前会话可见，适用于暂存会话内的中间数据。

- 本地临时表会在会话结束后连同数据和表结构都进行自动清理

- 本地临时表的语义与 MySQL 临时表一致，它有以下特性：

- 本地临时表的表定义不持久化，只在创建该表的会话内可见，其他会话无法访问该本地临时表

- 不同会话可以创建同名的本地临时表，各会话只会读写该会话内创建的本地临时表

- 本地临时表的数据对会话内的所有事务可见

- 本地临时表可以与普通表同名，此时在 DDL 和 DML 语句中，普通表被隐藏，直到本地临时表被删除

- 全局临时表：表定义对整个 TiDB 集群可见，表内数据只对当前事务可见，适用于暂存事务内的中间数据。

- 通过加上 GLOBAL 关键字来声明所创建的是全局临时表，必须在末尾 ON COMMIT DELETE ROWS 修饰，全局数据表的所有数据行将在事务结束后被删除

- 全局临时表在**事务结束**后会自动清除数据，但是表结构依然保留，需要手动删除。

- 在对全局临时表导入数据时，要特别注意，必须通过 BEGIN 显式声明事务的开始。否则导入的数据在 INSERT INTO 语句执行后就清除掉，因为 Auto Commit 模式下，INSERT INTO 语句的执行结束，事务会自动被提交，事务结束，全局临时表的数据便被清空了

- 通过 SHOW [FULL] TABLES 语句可以查看到已经创建的全局临时表，但是无法看到本地临时表的信息

- 全局临时表的表定义会持久化，对所有会话可见

- 全局临时表的数据只对当前的事务内可见，事务结束后数据自动清空

- 全局临时表不能与普通表同名

临时表与视图比较：

- 在对[视图](https://docs.pingcap.com/zh/tidb/stable/dev-guide-use-views)进行查询时，会再执行导入数据时所使用的原始查询

- 在对临时表进行查询时，不会再执行导入数据时所使用的原始查询，而是直接从临时表中获取数据

* + 运用 Cached Table

- 该功能适用于频繁被访问且很少被修改的热点小表，即把整张表的数据加载到 TiDB 服务器的内存中，直接从内存中获取表数据，避免从 TiKV 获取表数据，从而提升读性能。

- 读取缓存表会使用 UnionScan 算子，所以通过 explain 查看缓存表的执行计划时，可能会在结果中看到 UnionScan

- 往缓存表写入数据时，有可能出现秒级别的写入延迟。延迟的时长由全局环境变量 [tidb\_table\_cache\_lease](https://docs.pingcap.com/zh/tidb/stable/system-variables" \l "tidb_table_cache_lease-%E4%BB%8E-v600-%E7%89%88%E6%9C%AC%E5%BC%80%E5%A7%8B%E5%BC%95%E5%85%A5) 控制。执行修改操作后缓存会失效，需要重新加载

- 对缓存表执行 DDL 语句会失败。若要对缓存表执行 DDL 语句，需要先去掉缓存属性，将缓存表设回普通表后，才能对其执行 DDL 语句

- 对于每张缓存表的大小限制为 64 MB。如果表的数据超过了 64 MB，执行 ALTER TABLE t CACHE 会失败

- 不支持临时表、分区表、视图，不支持stale read，设置 tidb\_snapshot 读取历史数据时无法启用缓存表

- 所有的 TiDB 数据迁移工具均不支持缓存表功能，包括 Backup & Restore (BR)、TiCDC、Dumpling 等组件，它们会将缓存表当作普通表处理

* + 运用事务隔离级别

- TiDB 的 SI 隔离级别可以克服幻读异常 (同一个事务内先后读取记录数变多)，但 ANSI/ISO SQL 标准中的 RR 不能

- TiDB 的 SI 隔离级别不能克服写偏斜异常（Write Skew），需要使用 Select for update 语法来克服写偏斜异常

- 事务大小限制：最大单行记录为 120MB，最大单个事务为 10GB，建议每 100～500 行写入一个事务。自动提交下的 SELECT FOR UPDATE 目前不会加锁，可以通过使用显式的 BEGIN;COMMIT; 解决该问题

* + 理解 Clustered Primary Key

- 出于为性能考虑，尽量避免存储超宽表，表字段数不建议超过 60 个，建议单行的总数据大小不要超过 64K

- 聚簇索引优势：插入或涉及主键的读，会减少网络访问从而提高性能，将带来性能和吞吐量的优势

- 聚簇索引劣势：当使用大于 64 位的数据类型作为主键时，有多个二级索引时会用更多空间。批量插入大量取值相邻的主键时，可能会产生较大的写热点问题

* 应用开发最佳实践
  + 运用 Connector 处理异常的常识

- 在 TiDB 中，如果采用乐观事务模型，想要避免提交失败，需要在自己的应用程序的业务逻辑中添加机制来处理相关的异常。特别是，您的重试逻辑必须：

- 如果失败重试的次数达到 max\_retries 限制，则抛出错误

- 使用 try ... catch ... 语句捕获 SQL 执行异常，当遇到下面这些错误时进行失败重试，遇到其它错误则进行回滚。

- Error 8002: can not retry select for update statement：SELECT FOR UPDATE 写入冲突报错。

- Error 8022: Error: KV error safe to retry：事务提交失败报错。

- Error 8028: Information schema is changed during the execution of the statement：表的 Schema 结构因为完成了 DDL 变更，导致事务提交时报错。

- Error 9007: Write conflict：写冲突报错，一般是采用乐观事务模式时，多个事务都对同一行数据进行修改时遇到的写冲突报错。

- 在 try 块结束时使用 COMMIT 提交事务：

* + 运用乐观锁和悲观锁

- [乐观事务](https://docs.pingcap.com/zh/tidb/stable/optimistic-transaction)模型就是直接提交，遇到冲突就回滚，适合冲突率不高的场景，回滚的代价会比较大

- [悲观事务](https://docs.pingcap.com/zh/tidb/stable/pessimistic-transaction)模型就是在真正提交事务前，先尝试对需要修改的资源上锁，只有在确保事务一定能够执行成功后，才开始提交。对于冲突率高的场景，提前上锁的代价小于事后回滚的代价，在冲突率不高的场景并没有乐观事务处理高效

- 从应用端实现的复杂度而言，悲观事务更直观，更容易实现。而乐观事务需要复杂的应用端重试机制来保证

* + 运用预编译语句

推荐配置：

- 当需要修改、删除、插入多行数据时，推荐使用单个 SQL 多行数据的语句，不推荐使用多个 SQL 单行数据的语句。使用addBatch()和executeBatch API，同时jdbc-url里配置rewriteBatchedStatements=true实现 Batch 网络发送

JDBC:mysql://{TiDBIP}:{PORT}/{dbname}?characterEncoding=utf8&useSSL&\

useServerPrepStmts=true&\ # 开启预编译，对OLTP友好

preStmtCacheSqlLimit=10000000000&\ #缓存SQL的条数

useConfigs=maxPerformance&\ #jdbc MySQL超时等参数组合的最大性能

rewriteBatchedStatements=true&\ #批量操作处理

Defaultfetchsize=-214783648 #不缓存数据到客户端便于大量数据场景流式读取

- 当需要多次执行某个 SQL 语句时，推荐使用 PREPARE 语句，可以避免重复解析 SQL 语法的开销。注意不要重复执行 PREPARE 语句，否则并不能提高执行效率。 jdbc-url里设置useServerPrepStmts=true

- 如非必要，不要总是用 SELECT \* 返回所以列的数据，应该仅查询需要的列信息

- 读取大量数据时，流式读取

- 设置 [FetchSize 为 Integer.MIN\_VALUE](https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-reference-implementation-notes.html" \l "ResultSet" \t "https://docs.pingcap.com/zh/tidb/stable/_blank) 让客户端不缓存，客户端通过 StreamingResult 的方式从网络连接上流式读取执行结果，这个比较高效，推荐。

- 另一个相比低效但也可以用的是，使用 Cursor Fetch，首先需[设置 FetchSize](http://makejavafaster.blogspot.com/2015/06/jdbc-fetch-size-performance.html" \t "https://docs.pingcap.com/zh/tidb/stable/_blank) 为正整数，且在 JDBC URL 中配置 useCursorFetch = true。

- 当需要删除一个表的所有数据时，推荐使用 TRUNCATE 语句，不推荐使用 DELETE 全表数据

- TiDB 支持在线添加索引操作，添加索引不会阻塞表中的数据读写。可以通过修改[tidb\_ddl\_reorg\_worker\_cnt](https://docs.pingcap.com/zh/tidb/stable/system-variables" \l "tidb_ddl_reorg_worker_cnt)和[tidb\_ddl\_reorg\_batch\_size](https://docs.pingcap.com/zh/tidb/stable/system-variables" \l "tidb_ddl_reorg_batch_size)系统变量来调整 DDL 操作 re-organize 阶段的并行度与回填索引的单批数量大小，线上对加索引列有频繁写操作时可以适当调小它们的值，如果只有读或线上负载不大可以适当调大

* + 运用 KeySet Seeker 提升分页性能