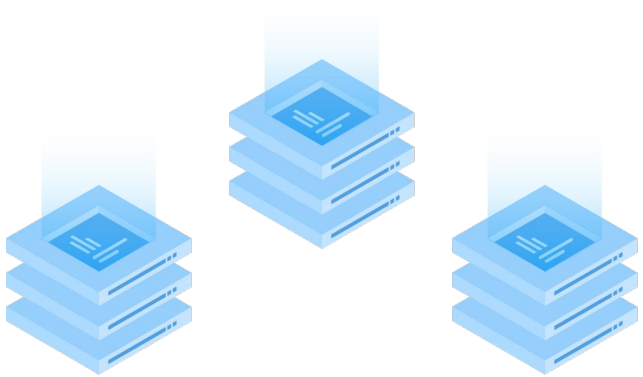
课程编号: PCTA-107-1

TiDB 应用开发最佳实践

高振娇

PingCAP TiDB



PingCAP PingCAP University

讲师介绍

高振娇
PingCAP 用户生态团队 DBA

课程概要

- 事务
  - TiDB 隔离级别详解
  - 显式事务中 DML 语句返回的 affected rows 不可信
  - 避开丢失更新影响的应用开发方法
  - “嵌套事务”
  - 大事务
- 自增 ID
  - TiDB 中的自增 ID 分配原理
  - 自增 ID 设计最佳实践
  - TiDB 中的自增 ID 使用方法

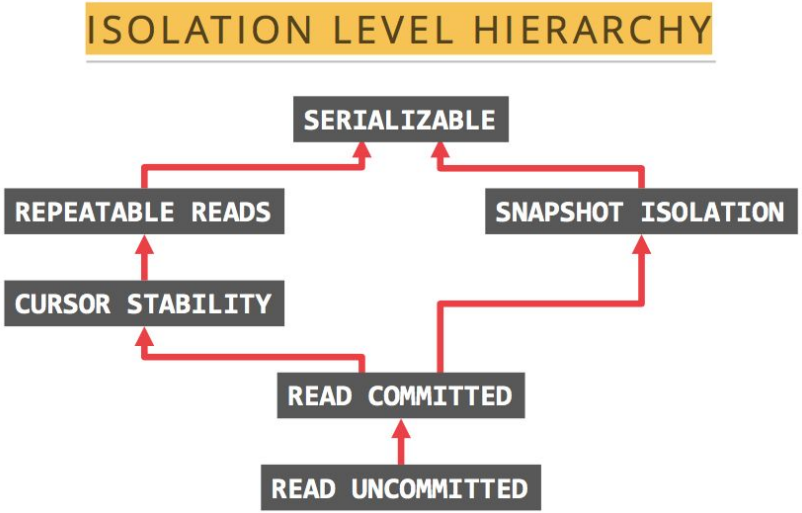
课程概要

- 约束
  - 主键与唯一索引
  - 外键
  - INSERT 语句默认只在提交时进行唯一性约束校验
- 批量计算场景的写入优化
  - 配置 SHARD_ROW_ID_BITS 拆散写入热点
  - 分区表

第一部分 :事务

课程编号: PCTA-1001

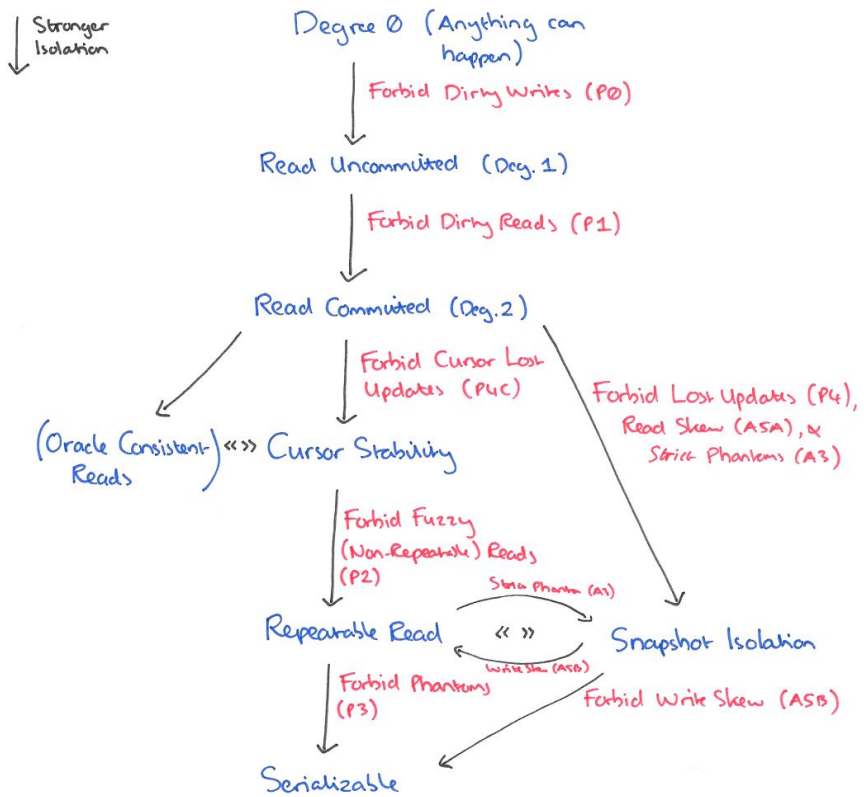
事务 - TiDB 隔离级别详解



几种 DBMS 产品的默认隔离级别：

DBMS 产品	默认隔离级别	实现方式
Oracle	Read Committed	悲观锁 + MVCC
DB2	Cursor Stability	悲观锁
MS-SQL Server	Read Committed	悲观锁 + MVCC
PostgreSQL	Read Committed	悲观锁 + MVCC
MySQL	Reapeatable Read	悲观锁 + MVCC
TiDB	Snapshot Isolation	乐观锁 + MVCC

事务 - TiDB 隔离级别详解



TiDB 支持的隔离级别为 Snapshot Isolation（SI），

与Repeatable Read（RR）隔离级别基本等价。

事务 - TiDB 隔离级别详解

克服幻读异常（Phantom Reads）

- 幻读是指事务 A 首先根据条件查询得到 n 条记录，然后事务 B 增添了 m 条符合事务 A 查询条件的记录，导致事务 A 再次发起请求时发现有 n+m 条符合条件记录，就产生了幻读。

不能克服写偏斜异常（Write Skew）

- 写偏斜是指两个并发的事务读取了两行不同但相关的记录，接着这两个事务各自更新了自己读到的那行数据，并最终都提交了事务，如果这两行相关的记录之间存在着某种约束，那么最终结果可能是违反约束的。这被称为写偏斜异常。

事务重试不能克服丢失更新异常

- 丢失更新是指两个事务 A，B 读取相同记录并更新同一列的值，若 A 先于 B 提交事务，当 B 事务提交后 A 再次查询时发现自己的更新丢失了。

事务 - 写偏斜异常（Write Skew）

- 写偏斜（Write Skew）

值班表有两列，姓名以及 值班状态，0 代表不值班，1 代表值班

姓名	值班状态
张三	0
李四	0
王五	0

有这样一个事务，它的逻辑是判断当前无人 值班，则分配一个 值班人。

当该程序顺序执行时，只会分配一个 值班人。但当它并行 执行时，就可能出 现多人同时为值班状态的逻辑错误。

事务 - 重试机制 — 丢失更新异常

TiDB 使用了乐观锁机制，乐观锁仅在提交时才会进行冲突检测和数据上锁，且具有事务重试特性，由参数 tidb_disable_txn_auto_retry 和 tidb_retry_limit 开启和关闭该特性。

- **显式事务**

在 TiDB 中开启事务重试机制，并且使用 begin 或关闭 auto_commit 属性。此时，对提交时遇到冲突而发生退避的事务会进行自动重试，当事务达到退避次数限制（ tidb_retry_limit ），依然不能成功提交 时，事务才会被回滚，并返回给用户执行失败的信息。

否则，发生退避的事务会重新获取时间戳，将事务第一次执行的所有语句重新执行一遍，当一个事 务里的后续语句是否执行取决于前面 语句执行结果的时候，自动重试会违反快照隔离，导致丢失更新。这种情况下，需要在 应用层重试整个事务。

- **隐式事务**

在 TiDB 中改变 tidb_disable_txn_auto_retry 变量不会影响 auto_commit = 1 的单语句的隐式事务，因为该语句的自动重试，不会造成 丢失更新等异常，即不会破坏事 务的隔离性。

事务 - 重试机制 — affected rows 不可信

- **显式事务**

在显式事务中，DML 操作所返回的 affected rows 并不保证与最终提交时所影响的数据行数一致。

这是由于在显式执行的事务时，DML 操作与提交操作分开被执行。在事务提交过程中，如果发生事务冲突，如找不到 TiKV，网络不稳定等原因而发生了重试。TiDB 将获取新的时间戳重新执行本事务中的操作，原本的 SI 隔离级别在重试后会产生类似 RC 隔离级别的不可重复读与幻读异常现象。

重试机制在 TiDB 内部完成，如果最终本事务提交成功，用户一般是无法感知到是否发生了重试的。因此，**不能通过 affected rows 来作为程序执行逻辑的判断条件**。

- **隐式事务**

隐式事务中（以单条 SQL 为单位进行提交），语句的返回是提交之后的结果。因此，**隐式事务中的 affected rows 是可信的。**

事务 - 避开丢失更新影响的应用开发方法

场景一，在转账交易的场景中，一般通过账号筛选出需要修改余额的记录。想象一个场景：账户 1，2，3 都有 1000 元余额。此时，账户 1 和 账户 2 分别向账户 3 转账 100 元。如果在应用中，计算出转账后两账户的余额，使用常值写入余额字段，这样的实现方式在事务并发执行时将会导致总账错误。

时刻	SESSION A	SESSION B
T1	begin; select realtimeremain from bank_acount_detail_info where userid = (1,3); update bank_acount_detail_info set realtimeremain = 900 where user_id = 1 ; update bank_acount_detail_info set realtimeremain = 1100 where user_id = 3 ;	
T2		begin; select realtimeremain from bank_acount_detail_info where userid = (2,3); update bank_acount_detail_info set realtimeremain = 900 where user_id = 2 ; update bank_acount_detail_info set realtimeremain = 1100 where user_id = 3 ;
T3		commit
T4	commit;	
T5	select user_id, realtimeremain from bank_acount_detail_info where user_id in(1,3);	select user_id, realtimeremain from bank_acount_detail_info where user_id in(2,3);

事务 - 避开丢失更新影响的应用开发方法

场景二，某公司决定给所有部门为 IT 的员工加薪 100，如下的实现的事务因某种原因而并发执行时，假设操作员 A 执行了加薪操作，事务还未提交时操作员 B 检查发现还没有人执行加薪操作，B 重复执行了程序，在 A 提交事务之后 B 也提交了事务，A 查看 IT 部门的最新薪资发现员工被加薪了 200，这样就会导致重复加薪的错误：

时刻	SESSION A	SESSION B
T1	begin; update compensation set salary=salary+100 where dept='IT';	
T2		begin; update compensation set salary=salary+100 where dept='IT';
T3		commit;
T4		select dept , salary from compensation where dept='IT';
T5	commit;	
T6	select dept , salary from compensation where dept='IT';	

事务 - 避开丢失更新影响的应用开发方法

在开启事务重试功能的前提下，针对上面出现的丢失更新的异常问题，建议应用端通过如下方式进行规避：

- **SQL 语句**

使用 `select for update` 时，TiDB 不会重试提交时遇到冲突的事务，在提交时检查到冲突（数据已经被修改），事务会被回滚。

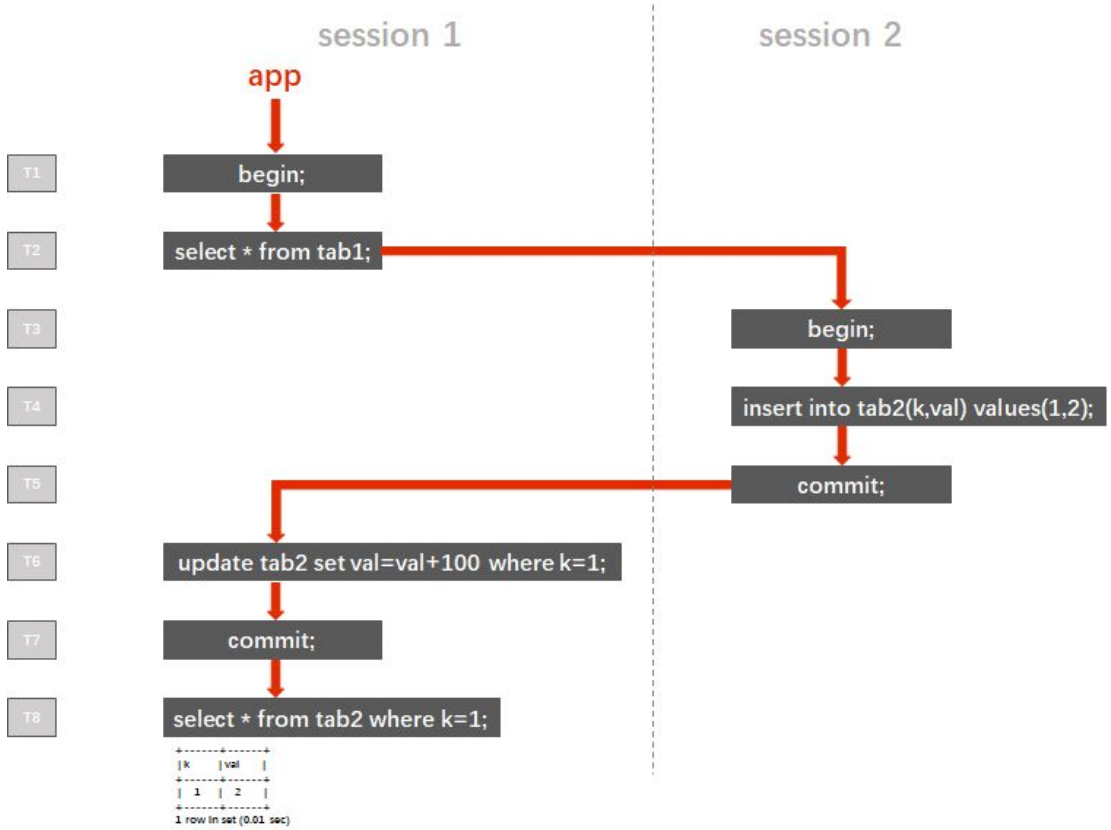
- **应用端设计**

在开发过程中，避免使用引起事务逻辑错误的编码方式。

事务 - “嵌套事务”

事务间不应该产生“嵌套”

遵照 ACID 理论，并发事务间应彼此相互隔离，避免互相干 扰，即事务的隔离性。



事务 - 大事务

TiDB 对事务大小的限制：

- 每个键值对不超过 6MB
- 键值对的总大小不超过 100MB
- 键值对的总数不超过 300,000

```
CREATE TABLE `user_info` (
  `id` bigint(11) NOT NULL AUTO_INCREMENT,
  `name` char(10) CHARSET utf8mb4 COLLATE utf8mb4_bin
DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `idx_name` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_bin ;

insert into user_info ( name ) values ( ‘gzj’ );
```

在 TiDB 中执行 insert 操作后，生成了 2 个键值对：

- 插入数据本身
- 插入普通索引

事务 - 大事务

TiDB 官方虽然提供了内部 batch 的方法，来绕过大事务的限制：

- tidb_batch_insert
- tidb_batch_delete
- tidb_dml_batch_size

但需要注意的是，开启了 batch 功能之后，大事务的完整性就没法保证了，只能保证每个批次的事务完整性。

故强烈建议在应用端进行缩减事务量的改造，最佳实践是将大事务分拆为小事务，分段提交。

第二部分：自增 ID

课程编号: PCTA-1001

自增 ID - TiDB 中的自增 ID 分配原理

- **分配原理**

TiDB 目前采用批量分配的方式，每个 tidb-server 实例缓存一段 ID 值用于分配（目前会缓存 30000 个 ID），用完这段值再去取下一段。

- **异常问题**

- 当客户端多个线程并发往不同的 tidb-server 插入数据的时候，有可能会出现后插入的数据自增 ID 小的情况。
- 多客户端插入数据时，可能会出现主键冲突的问题

假设集群中有两个 tidb-server 实例 A 和 B（A 缓存 [1,30000] 的自增 ID，B 缓存 [30001,60000] 的自增 ID），依次执行如下操作：

1. 客户端向 B 插入一条将 id 设置为 1 的语句 insert into t values (1, 1)，并执行成功。
2. 客户端向 A 发送 Insert 语句 insert into t (c) (1)，这条语句中没有指定 id 的值，所以会由 A 分配，当前 A 缓存了 [1, 30000] 这段 ID，所以会分配 1 为自增 ID 的值，并把本地计数器加 1。而此时数据库中已经存在 id 为 1 的数据，最终返回 Duplicated Error 错误。

- 使用约束

TiDB 允许给整型类型的字段指定 auto_increment，且一个表只允许一个属性为 auto_increment 的字段。

自增 ID - 自增 ID 设计最佳实践

- **表结构设计**

- 自增 ID 的字段类型必须为整型
- 建议使用 bigint，避免分布式数据库 ID 耗光的情况
- 自增 ID 一般不需要存储负值，为字段增加 unsigned 属性

示例如下：

`auto_inc_id` bigint unsigned not null unique index auto_increment comment '自增 ID'

自增 ID - TiDB 中的自增 ID 使用方法

两种方式让自增 ID 自动赋值

- 不指定自增 ID 字段，TiDB自动赋值
- 指定自增 ID 字段值为 NULL，TiDB 会自动进行赋值

```
MySQL [gin]> create table autoid(`auto_inc_id` bigint unsigned not null primary key auto_increment comment '自增 ID',b int);
Query OK, 0 rows affected (0.14 sec)

MySQL [gin]> insert into autoid(b) values(100);
Query OK, 1 row affected (0.17 sec)

MySQL [gin]> insert into autoid(b) values(100);
Query OK, 1 row affected (0.02 sec)

MySQL [gin]> insert into autoid(b) values(100);
Query OK, 1 row affected (0.02 sec)

MySQL [gin]> insert into autoid values(null,200);
Query OK, 1 row affected (0.02 sec)

MySQL [gin]> insert into autoid values(null,200);
Query OK, 1 row affected (0.02 sec)

MySQL [gin]> insert into autoid values(null,200);
Query OK, 1 row affected (0.02 sec)

MySQL [gin]> select * from autoid;
+-------------+------+
| auto_inc_id | b    |
+-------------+------+
|           1 |  100 |
|           2 |  100 |
|           3 |  100 |
|           4 |  200 |
|           5 |  200 |
|           6 |  200 |
+-------------+------+
6 rows in set (0.00 sec)
```

第三部分 :约束

课程编号: PCTA-1001

约束 - 主键、唯一键和外键

DBMS 中主键和唯一键都是表中数据的唯一性 约束，外键约束两个关系的相关性。但在 TiDB 中，开发时需要注意：

- 主键

TiDB 中的主键必须在建表时声明，目前版本 还不能为已有的表添加，修改或 删除主键。

- 唯一键

TiDB 中的唯一键可在新建表 时添加，也可以在已有表上 进行添加和删除操作。

- 外键

TiDB 不支持外键，要去掉所有表 结构中创建外键的相关语句。外键的级联操作多表数据的功能需要在应用中实现。

第四部分：批量计算场景的写入优化

课程编号: PCTA-1001

批量计算场景的写入优化

- **热点产生原因**

批量计算的回写操作往往产生写入热点。TiKV 是一个按 range 切分的 KV 系统，KV 的 Key 决定了写入位置在哪个 region，Key 的值取决于两种情况：

1. 当主键为整型（int, bigint, ...）时，Key 就是主键。
2. 其他情况 TiDB 为表创建隐藏列（_tidb_rowid），Key 是该隐藏列。

- **打散热点的方法**

- 用 SHARD_ROW_ID_BITS 来设置隐藏列 _tidb_rowid 分片数量的 bit 位数，默认值为 0，即 2^0 = 1 个分片。

  CREATE TABLE 语句示例：CREATE TABLE t (c int) SHARD_ROW_ID_BITS = 4;

  ALTER TABLE 语句示例：ALTER TABLE t SHARD_ROW_ID_BITS = 4;

- partitioned table

答疑安排

- 请登录 PingCAP 官方网校：university.pingcap.com

PingCAP

Thank You !