# 背景

# 方案

## DDL

### 创建带全局唯一索引的表

**基本流程（deal\_create\_table）：**

1. 获取共享内存中的信息，包括分片信息等；
2. 与zk交互检查表是否存在；
3. 如果存在全局唯一索引（UNIQUE GLOBAL INDEX， deal\_create\_table\_uniq\_key）：

构造主表->CREATE TABLE构造索引表

1. CREATE TABLE ... LIKE ... `存在全局索引的表`（deal\_create\_table\_like\_uniq\_key）：

构造主表->CREATE TABLE LIKE构造索引表

1. 处理TDSQL\_DISTRIBUTED BY ..定义全局分区表（GlobalPart\_deal\_create\_table）
2. 如果是广播表和普通shard表，清除自定义的分片信息，下发DB
3. 所有set返回OK后，会调用add\_del\_shardkey将对应的元数据信息同步到zk中。

**举例：**

CRETE TABLE t(a INT KEY, b INT, c INT, UNIQUE GLOBAL INDEX idx\_b(b)) shardkey=a;

**网关将按如下格式构建主表和索引表：**

# 构建主表

CREATE TABLE t1(a INT KEY, b INT, c INT, INDEX idx\_b(b)) SHARDKEY=a;

# 构建全局唯一索引表，以唯一索引的第一个列作为分区键（manager用户授权时，需要同时对主表和全局索引表进行授权）

CREATE TABLE `t1##idx\_b`(b INT, a INT, UNIQUE INDEX idx\_b(b), INDEX pk(a)) SHARDKEY=b;

**约束条件：**

1. 只支持在shard表（即不支持二级分区、 TDSQL\_DISTRIBUTED表、全局表和单表）中定义全局唯一索引
2. 支持在建表时指定全局唯一索引
3. 必须要有主键，且主键和全局唯一索引的总数不能超过64（便于代码实现时使用bitmap进行判断和处理）
4. 索引表名不能超过DB上最长表名的限制（64字节）
5. 暂不支持唯一索引列有默认值
6. 索引列不允许存在timestamp存在 ON UPDATE CURRENT\_TIMESTAMP

**ZK/共享内存元数据：**

1、zk中记录主键和全局索引的信息



共享内存以XML格式展示的全局索引信息：



### DROP TABLE

基本流程（deal\_table\_query）：

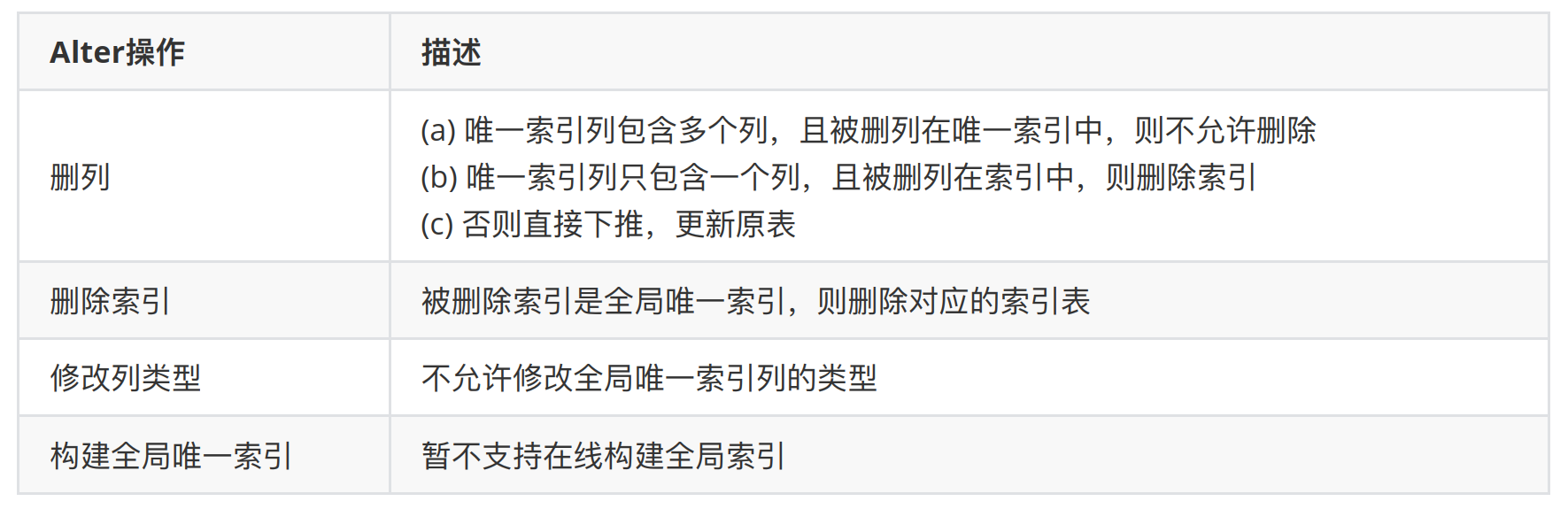
1. 如果删除含有全局唯一索引的表（deal\_drop\_table\_uniq\_key）
2. 如果对含有全局唯一索引的表执行ALTER操作（deal\_alter\_table\_uniq\_key）：
3. 如果对含有全局唯一索引的表执行rename操作（deal\_rename\_table\_uniq\_key）：
4. 如果对含有全局唯一索引的表执行truncate操作（deal\_truncate\_table\_uniq\_key）：
5. 处理全局分区表（GlobalPart\_deal\_table\_query）
6. 处理二级分区表（change\_sets\_sql\_table）

**举例：**

DROP TABLE t;

删除主表和全局唯一索引表，删除zk元数据

### ALTER/DROP INDEX



### 非在线创建语法

CREATE UNIQUE GLOBAL INDEX `idx` ON `t1`(`b`) OFFLINE;

// 创建一个未填充数据的全局索引表

INSERT INTO `t1##idx` SELECT b,pk FROM t1;

// 将T1的数据静态搬迁至全局索引表中，或者通过多源同步的方式同步数据

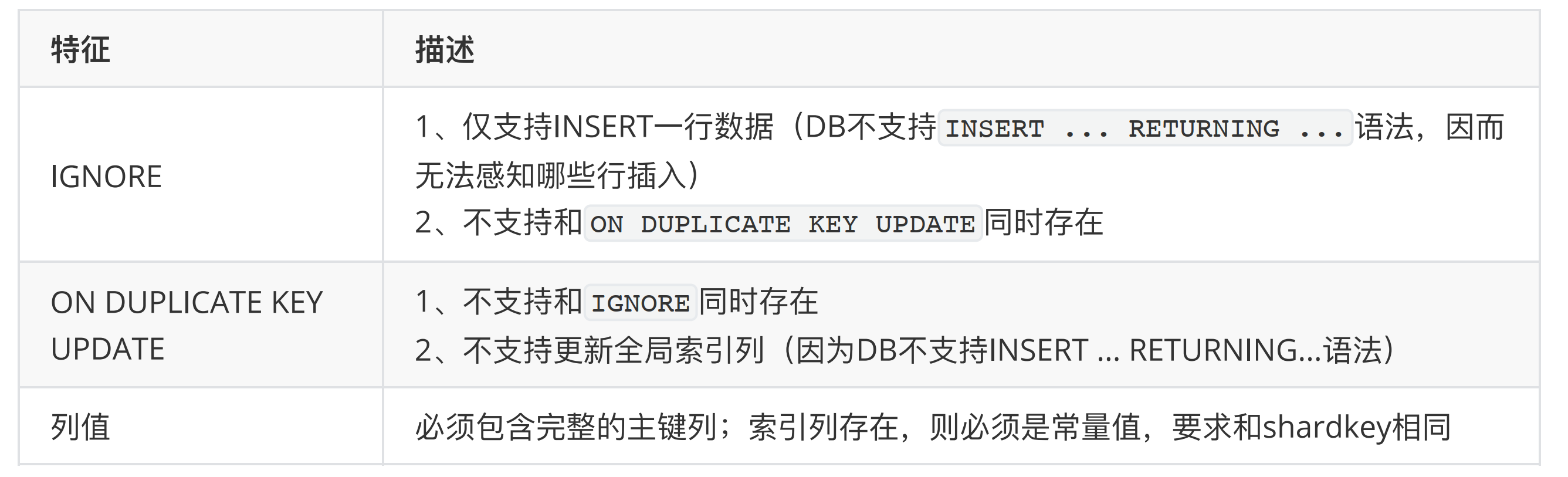
ALTER TABLE t1 MODIFY GLOBAL UNIQUE INDEX idx PUBLIC;

// 将全局索引表与原表绑定

## DML

### INSERT

**要点：**



**主要接口：**

joiner\_t::make\_insert\_select\_action

**执行流程：**

1. 当前会话不在事务中，则开启新的分布事务；
2. 对主表执行INSETRT操作；
3. 如果步骤(2)成功且影响行数大于0，则对所有索引表构建并执行INSERT操作（如果分区键为空，则默认为null）
4. 如果以上步骤皆执行成功：
5. 如果当前事务为新创建的分布式事务，则提交当前事务；
6. 返回步骤2的影响行数。

如果以上任意步骤执行失败：

1. 如果当前事务不是新创建的分布式事务，则将事务状态设置为rollback only状态，要求用户必须回滚该事务；
2. 否则，回滚该事务，并返回报错信息

### DELETE

**要点：**



**主要接口：**

joiner\_t::make\_delete\_action

**执行步骤：**

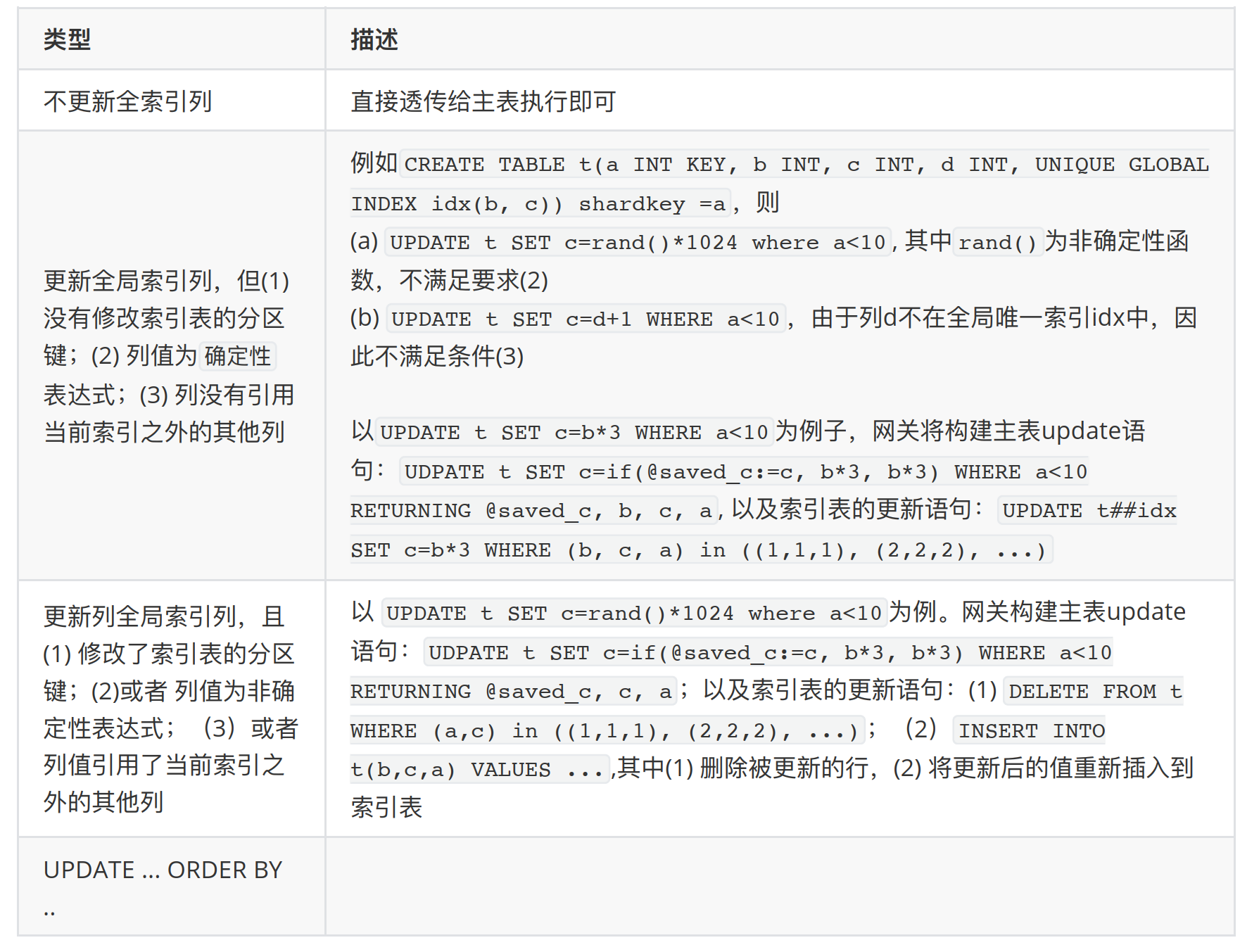
1. 当前会话不在事务中，则开启新的分布事务；
2. 对主表执行DELETE操作，并将returning返回的结果缓存在内存中（内存大小有参数security.key.max\_modified\_rows控制）；
3. 如果步骤(2)成功且影响行数大于0，则根据(2)得到的结果，为所有索引表构建并执行DELETE操作（如果分区键为空，则默认为null；每4k行构建一条DELETE）
4. 如果以上步骤皆执行成功：
5. 如果当前事务为新创建的分布式事务，则提交当前事务；
6. 返回步骤2的影响行数。

如果以上任意步骤执行失败：

1. 如果当前事务不是新创建的分布式事务，则将事务状态设置为rollback only状态，要求用户必须回滚该事务；
2. 否则，回滚该事务，并返回报错信息

### UPDATE

**要点：**



**主要接口：**

joiner\_t::make\_update\_action

**执行步骤：**

1. 当前会话不在事务中，则开启新的分布事务；
2. 对主表执行UPDATE操作，并将returning返回的结果缓存在内存中（内存大小有参数security.key.max\_modified\_rows控制）；
3. 如果步骤(2)成功且影响行数大于0，则根据(2)的结果，对所有索引表构建并执行UPDATE操作（如果分区键为空，则默认为null；每4k行执行一次UPDATE操作）
4. 如果以上步骤皆执行成功：
5. 如果当前事务为新创建的分布式事务，则提交当前事务；
6. 返回步骤2的影响行数。

如果以上任意步骤执行失败：

1. 如果当前事务不是新创建的分布式事务，则将事务状态设置为 rollback only 状态，要求用户必须回滚该事务；
2. 否则，回滚该事务，并返回报错信息

### REPLACE

// 暂未实现，不支持。实现、优化单条的。

仅支持单行数据的插入。转换为：

DELETE FROM t1 where .... RETURNING ...;

INSERT INTO t1 ...;

### SELECT

暂未借助全局唯一索引进行优化

### INSERT/REPLACE INTO... SELECT...

**要点：**

1. 不支持 REPLACE INTO
2. 不支持 ignore
3. 不支持 ON DUPLICATE KEY UPDATE 对索引列进行更新

**执行步骤：**

1. 当前会话不在事务中，则开启新的分布事务（用户无感知）；
2. 先执行SELECT，计算出要插入的数据；
3. 根据步骤(2)的结果，构建主表的INSERT语句
4. 如果步骤(3)成功，则根据步骤(2)的结果依次为每个索引表构建并执行INSERT操作（如果分区键为空，则默认为null）
5. 如果以上步骤皆执行成功：
6. 如果当前事务为新创建的分布式事务，则提交当前事务；
7. 返回步骤2的影响行数

如果以上任意步骤执行失败：  
1）如果当前事务不是新创建的分布式事务，则将事务状态设置为 rollback only 状态，要求用户必须回滚该事务；

2）否则，回滚该事务，并返回报错信息

### 多表联合update

要点：

1. 不支持更新主键
2. 如果修改了全局索引列，则对全局索引执行UPDATE时会转换为DELETE和INSERT
3. 如下场景下，对主表的使用逐行更新的方式：

例如UPDATE t1, t2 SET t1.b=t2.b WHERE ...，t1.b的新值依赖于其他表的值

例如UPDATE t1 SET b=cast(rand()\*1024 as unsigned) WHERE ... , 全局索引列b依赖与非确定性函数

**逐行更新主表的方式为:**

UPDATE t1 SET b=520 WHERE pk = ..; UPDATE t1 SET b=521 WHERE pk = ..; ...

批量更新主表的方式为：

UPDATE t1 SET b=b+1 WHERE (pk) in ((1), (2), (3), ...)

**执行步骤：**

1. 当前会话在事务中，则开启新的分布事务（用户无感知）；
2. 先执行SELECT，计算出要被更新行的主键，被更新列的新值；
3. 根据步骤(2)的结果，构建主表的UPDATE语句（逐行或者批量更新）
4. 如果步骤(3)成功且修改了全局索引列，则根据步骤(2)的结果依次为每个索引表构建并执行DELETE操作，删除被更新的行；然后构建并执行INSERT操作，将更新后的新值插入到了全局索引表
5. 如果以上步骤皆执行成功：
6. 如果当前事务为新创建的分布式事务，则提交当前事务；
7. 返回步骤2的影响行数。

如果以上任意步骤执行失败：

1. 如果当前事务不是新创建的分布式事务，则将事务状态设置为 rollback only 状态，要求用户必须回滚该事务；
2. 否则，回滚该事务，并返回报错信息

### 多表联合删除

**要点：**

分别为主表和索引表构建DELETE

**执行步骤：**

1. 当前会话不在事务中，则开启新的分布事务（用户无感知）；
2. 先执行SELECT，计算出要被删除行的主键以及各个全局索引列；
3. 根据步骤(2)的结果，构建主表的DELETE语句，进行批量删除： DELETE FROM t WHERE (pk) in ((1), (2),...)
4. 如果步骤(3)，则根据步骤(2)的结果依次为每个索引表构建并执行DELETE操作: DELETE FROM t WHERE (pk)in ((1),(2), ...) ，并根据全局索引列的分区键路由到对应SET执行
5. 如果以上步骤皆执行成功：
6. 如果当前事务为新创建的分布式事务，则提交当前事务；
7. 返回步骤2的影响行数。

如果以上任意步骤执行失败：

1. 如果当前事务不是新创建的分布式事务，则将事务状态设置为 rollback only 状态，要求用户必须回滚该事务；
2. 否则，回滚该事务，并返回报错信息。