```
──────────────────────────── MODULE basicakgi ────────────────────────────
a simplified specification for uniqorn. (1) a time oracle is used for versioning
(2) we omit update operations, which have the same effect as create operations,
(3) each record has only and exactly one alternate key. (4) a background garbage cleanup process
is used to delete garbage index reocrds. (5) when a create needs to reuse a garbage index record,
it will not delete it , i.e. no mandatory garbage cleanup is used, only the background garbage
cleanup process will delete garbage index records.
EXTENDS TLC, Integers, Sequences, FiniteSets, Bags

 phases for create (update operations have the same effect, so are omitted)
CONSTANTS CREATE_INIT_DATA_RECORD
CONSTANTS CREATE_PERSIST_INDEX_RECORD
CONSTANTS CREATE_PERSIST_DATA_RECORD

 phases for cleanup
CONSTANTS CLEANUP_VALIDATE
CONSTANTS CLEANUP_CHANGE_LOCK
CONSTANTS CLEANUP_DELETE_GARBAGE

 delete has only one phase, so we ignore it

 set of integer keys for primary keys and alternate keys
CONSTANTS PKS, AKS

 a non-zero integer
CONSTANTS VAL

 data or index records in data store partitions or index store partitions
VARIABLES persistedDataRecords, persistedIndexRecords

 seperate queues for all create/clenaup operations, delete has only one phase,
 no need a queue for it.
 an operation can equeue and equeue as it progress through its various phases.
 no operations, once
 enqueued, will be dequeued, in order to emulate duplicated operations
VARIABLES inprogressCreates, inprogressCleanups

 we use global monotonic timestamp for the basic case
VARIABLES timestamp

 ******************** data                  store accesses start here********************************
IsDummy(pk) ≜ IF  ∧ pk ∈ DOMAIN persistedDataRecords
                  ∧ persistedDataRecords[pk].ak = 0
                  ∧ persistedDataRecords[pk].val = 0
                THEN TRUE
                ELSE  FALSE

IsStale(pk, ts) ≜ IF  ∧ pk ∈ DOMAIN persistedDataRecords
                      ∧ persistedDataRecords[pk].ts > ts
```

```
                          THEN TRUE
                          ELSE  FALSE

$isLockHeld(pk, ts) \triangleq$ IF $\land\ pk \in$ DOMAIN $persistedDataRecords$
                               $\land\ persistedDataRecords[pk].ts = ts$
                             THEN TRUE
                             ELSE  FALSE

$DataStoreDelete(pk) \triangleq$
    $\land\ pk \in$ DOMAIN $persistedDataRecords$
    $\land\ persistedDataRecords' = [key \in ($DOMAIN $persistedDataRecords \setminus$
                                                          $\{pk\}) \mapsto persistedDataRecords[key]]$
    $\land$ UNCHANGED $\langle persistedIndexRecords, inprogressCreates, inprogressCleanups\rangle$

$DataStoreInitLock(pk, ak, ts) \triangleq$
    $\lor\ \land\ pk \notin$ DOMAIN $persistedDataRecords$
       $\land\ persistedDataRecords' = persistedDataRecords\ @@\ (pk :> [pk \mapsto pk, ts \mapsto ts, ak \mapsto 0, val \mapsto 0])$
       $\land\ inprogressCreates' = inprogressCreates \cup \{[phase \mapsto CREATE\_PERSIST\_INDEX\_RECORD,$
                                                        $pk \ \mapsto pk, ak \mapsto ak, ts \mapsto ts]\}$
    $\lor\ \land\ IsDummy(pk)$
       $\land\ \neg IsStale(pk, ts)$
       $\land\ persistedDataRecords' = [persistedDataRecords$ EXCEPT $![pk].ts = ts]$
       $\land\ inprogressCreates' = inprogressCreates \cup \{[phase \mapsto CREATE\_PERSIST\_INDEX\_RECORD,$
                                                        $pk \ \mapsto pk, ak \mapsto ak, ts \mapsto ts]\}$
    $\lor$ UNCHANGED $\langle persistedDataRecords, inprogressCreates\rangle$

$DataStoreUpdateOptimistically(pk, ak, ts) \triangleq$
    $\lor\ \land\ isLockHeld(pk, ts)$
       $\land\ persistedDataRecords' = [persistedDataRecords$ EXCEPT $![pk].ts = @ + 1,$
                                                        $![pk].ak = ak, ![pk].val = VAL]$
    $\lor$ UNCHANGED $persistedDataRecords$

$DataStoreValidate(pk, ak, ts) \triangleq$
    IF $pk \in$ DOMAIN $persistedDataRecords \ \land\ persistedDataRecords[pk].ak = ak$ THEN
      UNCHANGED $inprogressCleanups$
     ELSE
      $inprogressCleanups' = inprogressCleanups \cup \{[phase \mapsto CLEANUP\_CHANGE\_LOCK, pk \mapsto pk,$
                                                                  $ak \mapsto ak, ts \mapsto ts]\}$

$DataStoreChangeLock(pk, ak, ts) \triangleq$
    IF $pk \in$ DOMAIN $persistedDataRecords$ THEN
        IF $ak \neq persistedDataRecords[pk].ak$ THEN
            $\land$ IF $persistedDataRecords[pk].val = 0$ THEN
                   $persistedDataRecords' = \ [key \in ($DOMAIN $persistedDataRecords \setminus$
                                                          $\{pk\}) \mapsto persistedDataRecords[key]]$
                 ELSE
                  $persistedDataRecords' = [persistedDataRecords$ EXCEPT $![pk].ts = @ + 1]$
```

```
          ∧ inprogressCleanups' = inprogressCleanups ∪ {[phase ↦ CLEANUP_CHANGE_LOCK,
                                                        pk ↦ pk, ak ↦ ak, ts ↦ ts]}
        ELSE  UNCHANGED ⟨persistedDataRecords, inprogressCleanups⟩
     ELSE
        ∧ inprogressCleanups' = {inprogressCleanups} ∪ {[phase ↦ CLEANUP_DELETE_GARBAGE,
                                                           pk ↦ pk, ak ↦ ak, ts ↦ ts]}
        ∧ UNCHANGED persistedDataRecords

 data store partitioning/routing policies do not affect the correctness, so we ignore them
 ******************* data                  store accesses start here**********************************

 ******************* index                  store accesses start here*********************************
 index store has only two accesses methods: insert and delete. Update and replace accesses can
 be derived from these two acesses.
IndexStoreDirectlyInsert(ak, pk, ts) ≜
   ∨ ∧ ak ∉ DOMAIN persistedIndexRecords
      ∧ persistedIndexRecords' = persistedIndexRecords  @@ (ak :> [ak ↦ ak, pk ↦ pk,
                                                                   ts ↦ ts])
      ∧ inprogressCreates' = inprogressCreates ∪ {[phase ↦ CREATE_PERSIST_DATA_RECORD,
                                                   pk  ↦ pk, ak ↦ ak, ts ↦ ts]}
   ∨ UNCHANGED ⟨persistedIndexRecords, inprogressCreates⟩

IndexStoreDeleteOptimistically(ak, pk, ts) ≜
  IF  ∧ ak ∈ DOMAIN persistedIndexRecords
      ∧ persistedIndexRecords[ak].pk = pk
      ∧ persistedIndexRecords[ak].ts  = ts
   THEN
     persistedIndexRecords' = [key ∈ (DOMAIN persistedIndexRecords \
                                                {ak}) ↦ persistedIndexRecords[key]]
   ELSE  UNCHANGED persistedIndexRecords
 ******************* index                  store accesses end here********************************


 make a create operation go through its phases
RunCreate(createOp) ≜
   LET phase ≜ createOp.phase
       pk ≜ createOp.pk
       ak ≜ createOp.ak
       ts ≜ createOp.ts
   IN   ∨ ∧ phase = CREATE_INIT_DATA_RECORD
          ∧ DataStoreInitLock(pk, ak, ts)
          ∧ UNCHANGED ⟨persistedIndexRecords, inprogressCleanups⟩
        ∨ ∧ phase = CREATE_PERSIST_INDEX_RECORD
          ∧ IndexStoreDirectlyInsert(ak, pk, ts)
          ∧ UNCHANGED ⟨persistedDataRecords, inprogressCleanups⟩
        ∨ ∧ phase = CREATE_PERSIST_DATA_RECORD
```

```
           ∧ DataStoreUpdateOptimistically(pk, ak, ts)
           ∧ UNCHANGED ⟨persistedIndexRecords, inprogressCleanups, inprogressCreates⟩

 issue a create operation
Create(pk, ak) ≜
   ∧  inprogressCreates' = inprogressCreates ∪ {[phase ↦ CREATE_INIT_DATA_RECORD,
                                                  pk ↦ pk, ak ↦ ak, ts ↦ timestamp]}
   ∧  UNCHANGED ⟨persistedDataRecords, persistedIndexRecords, inprogressCleanups⟩

 issue a cleanup operation
Cleanup(ak, pk, ts) ≜
   ∧ inprogressCleanups' = inprogressCleanups ∪ {[phase ↦ CLEANUP_VALIDATE, ak ↦ ak,
                                                               pk ↦ pk, ts ↦ ts]}
   ∧  UNCHANGED ⟨persistedDataRecords, persistedIndexRecords, inprogressCreates⟩

 make a garbage cleanup operation go through its phases
RunCleanup(cleanupOp) ≜
   LET phase ≜ cleanupOp.phase
       pk  ≜ cleanupOp.pk
       ak  ≜ cleanupOp.ak
        ts ≜ cleanupOp.ts
   IN   ∨ ∧ phase = CLEANUP_VALIDATE
          ∧ DataStoreValidate(pk, ak, ts)
          ∧ UNCHANGED ⟨persistedDataRecords, persistedIndexRecords, inprogressCreates⟩
        ∨ ∧ phase = CLEANUP_CHANGE_LOCK
          ∧ DataStoreChangeLock(pk, ak, ts)
          ∧ UNCHANGED ⟨persistedIndexRecords, inprogressCreates⟩
        ∨ ∧ phase = CLEANUP_DELETE_GARBAGE
          ∧ IndexStoreDeleteOptimistically(ak, pk, ts)
          ∧ UNCHANGED ⟨persistedDataRecords, inprogressCreates, inprogressCleanups⟩

 all aks and pks are initialized in each partition
Init ≜  ∧ persistedDataRecords  = [pk ∈ {} ↦ {}]
        ∧ persistedIndexRecords = [ak ∈ {} ↦ {}]
        ∧ inprogressCreates = {}
        ∧ inprogressCleanups = {}
        ∧ timestamp = 0

Next ≜  ∧ ∨ ∃ pk ∈ PKS : DataStoreDelete(pk)
          ∨ ∃ pk ∈ PKS, ak ∈ AKS : Create(pk, ak)
          ∨ ∃ ak ∈ DOMAIN persistedIndexRecords : Cleanup(ak,
                    persistedIndexRecords[ak].pk, persistedIndexRecords[ak].ts)
          ∨ ∃ createOp ∈ inprogressCreates :  RunCreate(createOp)
          ∨ ∃ cleanupOp ∈ inprogressCleanups :  RunCleanup(cleanupOp)
        ∧ timestamp' = timestamp + 1

Spec ≜ Init ∧ □[Next]⟨persistedDataRecords, persistedIndexRecords, inprogressCreates,
                                   $inprogressCleanups,\ timestamp\rangle$

 no missing index record invariant
$NoMissing \triangleq \forall\, pk \in$ DOMAIN $persistedDataRecords :$
                  IF $persistedDataRecords[pk].ak \neq 0$ THEN
                      $\exists\, ak \in$ DOMAIN $persistedIndexRecords :$
                        $\land\ persistedIndexRecords[ak].pk = pk$
                        $\land\ persistedDataRecords[pk].ak = ak$
                   ELSE  TRUE

THEOREM $Spec \Rightarrow NoMissing$
```

* Modification History
* Last modified *Wed Mar* 07 17:22:58 *PST* 2018 by *jyi*
* Created *Thu Feb* 01 13:21:10 *PST* 2018 by *jyi*