```
──────────────────────── 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, CREATE_PERSIST_INDEX_RECORD, CREATE_PERSIS

 phases for cleanup
CONSTANTS CLEANUP_VALIDATE, CLEANUP_CHANGE_LOCK, 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) ≜ IF  ∧ pk ∈ DOMAIN persistedDataRecords
                         ∧ persistedDataRecords[pk].ts = ts
                       THEN TRUE
```

```
                                ELSE  FALSE

DataStoreDelete(pk) ≜  ∧ pk ∈ DOMAIN persistedDataRecords
                       ∧ persistedDataRecords′ = [key ∈ (DOMAIN persistedDataRecords \ {pk}) ↦ persisted
                       ∧ UNCHANGED ⟨persistedIndexRecords, inprogressCreates, inprogressCleanups⟩

DataStoreInitLock(pk, ak, ts) ≜
        ∨ ∧ pk ∉ DOMAIN persistedDataRecords
          ∧ persistedDataRecords′ = persistedDataRecords @@ (pk :> [pk ↦ pk, ts ↦ ts, ak ↦ 0, val ↦ 0])
          ∧ inprogressCreates′ = inprogressCreates ∪ {[phase ↦ CREATE_PERSIST_INDEX_RECORD, pk
        ∨ ∧ IsDummy(pk)
          ∧ ¬IsStale(pk, ts)
          ∧ persistedDataRecords′ = [persistedDataRecords EXCEPT ![pk].ts = ts]
          ∧ inprogressCreates′ = inprogressCreates ∪ {[phase ↦ CREATE_PERSIST_INDEX_RECORD, pk
        ∨ UNCHANGED ⟨persistedDataRecords, inprogressCreates⟩

DataStoreUpdateOptimistically(pk, ak, ts) ≜
      ∨ ∧ isLockHeld(pk, ts)
        ∧ persistedDataRecords′ = [persistedDataRecords EXCEPT ![pk].ts = @ + 1, ![pk].ak = ak, ![pk].val =
      ∨ UNCHANGED persistedDataRecords

DataStoreValidate(pk, ak, ts) ≜
                                IF pk ∈ DOMAIN persistedDataRecords  ∧ persistedDataRecords[pk].ak = ak THEN
                                   UNCHANGED inprogressCleanups
                                 ELSE
                                   inprogressCleanups′ = inprogressCleanups ∪ {[phase ↦ CLEANUP_CHANGE_LO

DataStoreChangeLock(pk, ak, ts) ≜
                      IF pk ∈ DOMAIN persistedDataRecords THEN
                         IF ak ≠ persistedDataRecords[pk].ak THEN
                            ∧ IF persistedDataRecords[pk].val = 0 THEN
                                 persistedDataRecords′ =  [key ∈ (DOMAIN persistedDataRecords \ {pk}) ↦ persiste
                               ELSE
                                 persistedDataRecords′ = [persistedDataRecords EXCEPT ![pk].ts = @ + 1]
                            ∧ inprogressCleanups′ = inprogressCleanups ∪ {[phase ↦ CLEANUP_CHANGE_LOCK
                          ELSE  UNCHANGED ⟨persistedDataRecords, inprogressCleanups⟩
                       ELSE
                         ∧ inprogressCleanups′ = {inprogressCleanups} ∪ {[phase ↦ CLEANUP_DELETE_GARBA
                         ∧ 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 acesse
IndexStoreDirectlyInsert(ak, pk, ts) ≜  ∨ ∧ ak ∉ DOMAIN persistedIndexRecords
                                          ∧ persistedIndexRecords′ = persistedIndexRecords  @@ (ak :> [ak ↦
                                   ∧ inprogressCreates′ = inprogressCreates ∪ {[phase ↦ CREATE_P
                                 ∨ 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}) ↦ persiste
                      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, a
       ∧ UNCHANGED ⟨persistedDataRecords, persistedIndexRecords, inprogressCleanups⟩

 issue a cleanup operation
Cleanup(ak1, pk1, ts1) ≜  ∧ inprogressCleanups′ = inprogressCleanups ∪ {[phase ↦ CLEANUP_VALIDATE
                          ∧  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, persistedInde
         ∨ ∃ createOp ∈ inprogressCreates :  RunCreate(createOp)
         ∨ ∃ cleanupOp ∈ inprogressCleanups :  RunCleanup(cleanupOp)
       ∧ timestamp′ = timestamp + 1

Spec ≜ Init ∧ □[Next]_⟨persistedDataRecords, persistedIndexRecords, inprogressCreates, inprogressCleanups, timestamp⟩

 no missing index record invariant
NoMissing ≜ ∀ pk ∈ DOMAIN persistedDataRecords :
                IF persistedDataRecords[pk].ak ≠ 0 THEN
                    ∃ ak ∈ DOMAIN persistedIndexRecords :
                      ∧ persistedIndexRecords[ak].pk = pk
                      ∧ persistedDataRecords[pk].ak = ak
                 ELSE  TRUE

THEOREM Spec ⇒ NoMissing
└─────────────────────────────────────────────────────────────────────┘
 \* Modification History
 \* Last modified Mon Mar 05 09:24:04 PST 2018 by jyi
 \* Created Thu Feb 01 13:21:10 PST 2018 by jyi
```