NoSQLz

User's Guide

version 2.1.0

NoSQLz, Version 2 Release 1 Level 0

First Edition (March 2015)

This document applies to *NoSQLz* Version 2 Release 1 Level 0.

Contents

1. Introduction *6*

1.1 What is NoSQLz **6**

1.2 Limitations **6**

2. Installation *9*

2.1 System requirements **9**

2.2 Sysplex requirements **9**

2.3 Installation steps **9**

2.3.1 Uploading product libraries 9
2.3.2 APF authorization 10
2.3.3 Link-list (optional) 11
2.3.4 Prepare the start procedure 11
2.3.5 Prepare the COMMIT dataset 11
2.3.6 Set NoSQLz parameters 11
2.3.7 Set the license code 12
2.3.8 Start the procedure 12
2.3.9 Verification procedure 12
2.3.10 Database maintenance 13

3. Database management *14*

3.1 Creating a database **14**

3.2 Declaring a database **14**

3.3 Database clean-up **15**

3.4 Database initial load **15**

3.5 Database security **15**

4. Function calls *16*

4.1 Description **16**

4.2 The READ function **16**

4.3 The READSEQ function **18**

4.4 The READTEST function **19**

4.5 The CREATE function **19**

4.6 The UPDATE function **20**

4.7 The DELETE function **21**

4.8 The COMMIT function **22**

4.9 The ROLLBACK function **23**

4.10 The FREEALL function **24**

4.11 NSQAREXX: the REXX interface **24**

5. Managing the NoSQLz started task *26*

5.1 NSQPARM parameters ... 26
5.1.1 DBNAME ... 26
5.1.2 COMTIME ... 26

5.2 Starting and stopping the started task ... 26

5.3 Commands to the started task ... 27
5.3.1 Display databases ... 27
5.3.2 Display current commit ... 27
5.3.3 Display tasks ... 28
5.3.4 Display Sysplex information ... 28
5.3.5 Display log entries ... 28
5.3.6 Display communication area ... 29
5.3.7 Close and free a database ... 29
5.3.8 Allocate and open a database ... 29

6. Batch utilities ... 31

6.1 NSQDUT01: database formatting ... 31
6.1.1 DD statements ... 31
6.1.2 Control statements ... 31
6.1.3 Sample JCL ... 32

6.2 NSQDUT02: database initial load ... 32
6.2.1 DD statements ... 32
6.2.2 Control statements ... 32
6.2.3 Sample JCL ... 33

6.3 NSQDUT03: printing database statistics ... 33
6.3.1 DD statements ... 33
6.3.2 Sample JCL ... 34

6.4 NSQDUT04: database clean-up ... 34
6.4.1 DD statements ... 34
6.4.2 Control statements (optional) ... 34
6.4.3 Sample JCL ... 35

6.5 NSQDUT10: database record display (from key) ... 35
6.5.1 DD statements ... 35
6.5.2 Control statements ... 35
6.5.3 Sample JCL ... 36

6.6 NSQDUT11: database record display (from record-id) ... 36
6.6.1 DD statements ... 36
6.6.2 Control statements ... 36
6.6.3 Sample JCL ... 36

6.7 NSQDUT20: database tailoring ... 37
6.7.1 DD statements ... 37
6.7.2 Control statements ... 37
6.7.3 Sample JCL ... 37
6.7.4 Example ... 38

7. Wikitext: a sample NoSQLz application ... 39

7.1 License ... 39

7.2 Data denormalization ... 39

7.3 Database design and application elements ... 40

7.4 Installing the application ... 42

7.5 Loading Wikipedia text ... 43

7.6 Using the application....43

7.7 Some programming details about the application....44

8. Appendix....46

8.1 The parameter block....46

8.1.1 Cobol description....46

8.1.2 Assembler description....46

8.2 Return codes....47

9. Readers' Comments — We'd Like to Hear from You....49

1. Introduction

1.1 What is NoSQLz

NoSQLz is a key-value Database Management System running under z/OS. It provides direct or sequential access to schema-less tables.

A table dataset (or database) is a VSAM dataset that resides on one (or several) disks.

Table data consists of 'format-agnostic' variable-length records. Records are handled through create, read, update and delete ('CRUD') functions.

Record size goes from 500 bytes (minimum) to 75 MB (large records are supported). There may be up to approximately 1,000,000,000 records in a single table.

A record key is always a variable-length string, from 1 to 250 bytes. In the same database, records may have different key-lengths. For this reason, the key length (along with the key value) must always be supplied to access records. A record is uniquely referred to by the combination (key length, key value).

ACID properties are provided for application programs that implement database transactions. A transaction is a set of information processing operations that leave the database in a consistent state: the operations that a transaction consists of are either all completed successfully or all cancelled successfully. The COMMIT processing takes care of consistency to maintain ACID properties throughout the life of the databases.

With NoSQLz, data retrieval is always favored, at the possible expense of updates.

NoSQLz version 1 is a free mono-system version destined to a monoplex or a non-Sysplex system. It can also be used on a development test system.

NoSQLz version 2 and above is a chargeable version that provides multiple concurrent accesses natively throughout the Sysplex.

NoSQLz may be used to great benefit in highly-available applications, data warehousing, document store, Internet applications, Big data applications, OLAP, etc. It can also be used to develop quick applications without involving sophisticated database management systems or transaction managers.

1.2 Limitations

Because of its simple design, NoSQLz is subject to some limitations, compared to a "full-fledged" DBMS:

- secondary indexes are not supported

- there is no "rollforward" journal that would enable the installation to apply committed transactions again in order to bring a database up to date after it is restored from a recent back-up version

- no specific database back-up utility is offered

- because NoSQLz implements optimistic concurrency control, timestamp-based concurrency control and multiversion concurrency control (MVCC), record updates may fail in the COMMIT process (as signaled by the return code)

- the NoSQLz clean-up utility should be frequently used for each database to have enough spare storage capacity (since update is never in-place, any record update creates obsolete versions of the record)

- there is no “two-phase commit” protocol available to run transactions that access both NoSQLz and non-NoSQLz databases

- no database buffer cache is used, I/O operations are always physical

- NoSQLz databases may occupy a much larger space than expected, because space allocation for them is always rounded up to a higher value (see also NSQDUT20: database tailoring, page 37)

- NoSQLz databases cannot be extended once allocated

- by nature, NoSQLz performs best when the read/write ratio is high; too many writes at the same time may impact the performance

2. Installation

2.1 System requirements

You must run a supported version of z/OS.

CPACF (Central Processor Assist for Cryptographic Functions) hash functions must be active (at least SHA-1 must be supported by the hardware).

This version supports a monoplex or full Sysplex system.

2.2 Sysplex requirements

At least a monoplex should be active.

Clock synchronization is essential since NoSQLz is based on timestamp-based concurrency control.

NoSQLz uses the following ENQ names to serialize on resources. SYSTEMS type names must be propagated throughout the sysplex. SYSTEM type names must not be propagated beyond the local system.

NoSQLz ENQ resources:

Major	Minor	Type
'NSQCOMIT'	NoSQLz 4-character system name	SYSTEMS
'NSQL'+ NoSQLz 4-character system name	'ACTIVE'	SYSTEM
'NSQCSLOT'	slot number + database dsname	SYSTEMS

A DISPLAY XCF command can display the status of a NoSQLz Sysplex group after the NoSQLz started tasks are active.

Example: D XCF,GROUP,NSQGPRD1,ALL (to display the NoSQLz 'PRD1' group).

2.3 Installation steps

2.3.1 Uploading product libraries

The binary file (for example: NSQ.NSQINST.XMI) must be transferred to MVS into a sequential LRECL 80 file.

Using the binary transfer method, upload it to your mainframe host into a pre-allocated, LRECL 80, sequential dataset.

You must now enter the following TSO command against the file:

```
 RECEIVE INDA('NSQ.NSQINST.XMI')
```

Respond as follows to the "Enter restore parameters…." message:

```
DA('desired.installation.dataset.name')
```

or :

```
DA('desired.installation.dataset.name') VOL(desired.allocation.volume)
```

Adapt and submit member **RECEIVE** of the newly created partitioned dataset. JCL variable INSTDS refers to the PDS you have just created using the RECEIVE command. JCL variable PFXC contain the high level qualifiers to assign to the product libraries.

This will create and upload all libraries from the installation members. You should not change the last dsn qualifier of the libraries (for example: NSQCNTL).

List of libraries:

Last dsn qualifier	**Usage**
NSQLOAD	Load library
NSQCNTL	JCLs and parameters
NSQTEST	Samples
NSQWIKI	Wikitext application material

2.3.2 APF authorization

You have to specify in the PROGxx member of the current PARMLIB the dsname of the NoSQLz load-module library, and the name of the disk volume it resides on, for example:

```
          ...
     APF ADD  DSNAME(PROD.NSQ.V210.NSQLOAD)  VOLUME(VOL001)
          ...
```

Make sure the new entry is taken into account by the system: this requires an IPL, or a SET PROG=xx command, or the SETPROG command:

```
     SETPROG APF,ADD,DSN=PROD.NSQ.V210.NSQLOAD,VOL=VOL001
```

An alternate option consists in copying NoSQLz authorized modules to a system library or to an existing APF library. The modules that must be copied are the ones that are marked AC=1.

2.3.3 Link-list (optional)

If the NoSQLz product becomes heavily used, it may be preferable to put the NoSQLz load library in link-list. The LNKLSTxx member of the system PARMLIB should be updated accordingly.

The alternative is to use the NoSQLz load library as STEPLIB or JOBLIB.

2.3.4 Prepare the start procedure

NoSQLz necessitates that a started task named '**NSQTASK'** be activated.

Copy the member NSQCNTL(NSQTASK) in one of your system PROCLIBs. Set variable NSQLOAD to the name of the product load library. DD NSQPARM may point to any PDS you wish. NSQPARM is discussed below in paragraph **Set NoSQLz parameters**.

Have the started task NSQTASK launched just after IPL.

The P= parameter of the NSQTASK procedure stands for the Sysplex-wide name of your NoSQLz system. It should be a 4-byte name, for example PROD, or TST1.

The START command should be in the following form:

```
S NSQTASK,REUSASID=YES[,P=<NoSQLz system name]
```

The STEPLIB must be APF-authorized (see paragraph « APF authorization »).

The RACF userid associated with the NSQTASK STC must be authorized in UPDATE to all databases that may be handled by the NoSQLz system, and in READ to NoSQLz parameter files.

2.3.5 Prepare the COMMIT dataset.

The COMMIT dataset (or COMDS) is required for COMMIT processing. This dataset can be allocated and formatted using the sample JCL in NSQCNTL(COMDSINI). The IEC161I message can be ignored.

2.3.6 Set NoSQLz parameters

The NSQPARM DD statement of the NSQTASK procedure refers to a PDS member that will contain all NoSQLz parameters ('NSQPARM parameters'). It is recommended to set the 4-byte NoSQLz system name as suffix of the member name, for example NSQP**TST1** if the NoSQLz system name is **TST1**.

The member should at least contain a statement to describe the required COMMIT dataset:

```
*---+----1----+----2----+----3----+----4----+----5----+----6----+
DBNAME   NSQCOMDS PROD1.COMDS                                  W
```

A sample may be found in member NSQCNTL(NSQPARM).

2.3.7 Set the license code

The NSQLIC DD statement of the NSQTASK procedure refers to a PDS member that will contain all license code parameters ('NSQLIC parameters'). It is recommended to set the 4-byte NoSQLz system name as suffix of the member name, for example NSQL**TST1** if the NoSQLz system name is **TST1**.

A sample may be found in member NSQCNTL(NSQLIC).

2.3.8 Start the procedure

The **NSQTASK** started task (or whatever name you gave it) may now be activated.

You may start it on any system that is part of your Sysplex. The JCL procedure and the parameters must be the same on all members of the Sysplex.

2.3.9 Verification procedure

You may want to use the provided samples to test that the installation is OK. The NSQTASK started task should not be active at this point, you must stop it otherwise.

1. JCL DBSTATES of NSQ.NSQTEST must be adapted and submitted to initialize and load a DBSTATES NoSQLz database. This JCL will create a DBSTATES table containing all states of the USA.

2. JCL DBCITIES of NSQ.NSQTEST must be adapted and submitted to initialize and load a DBCITIES NoSQLz database. You must manually replace NSQ.NSQTEST by the name of your NSQTEST library (in step 'PREPARE'). This JCL will create a DBCITIES table containing all cities and towns of the USA.

The output message IEC161I is normal in steps 1 and 2.

3. You must check that the NSQPARM parameters contain a statement for both DBSTATES and DBCITIES (in addition to NSQCOMDS), for example :

```
*---+----1----+----2----+----3----+----4----+----5----+----6----+

DBNAME   DBSTATES TST1.DBCITIES                                W

DBNAME   DBSTATES TST1.DBSTATES                                W
```

4. The NSQTASK procedure may now be started. The NoSQLz system name may be set for example to TST1 and NSQPARM must point to the parameters described in step 3.

5. You may run the REXXCITY REXX exec under TSO. This EXEC can be found in the NSQTEST library.

Note that this exec uses the NSQAREXX external function. Prior to using the exec, either NSQLOAD should be in linklist, or the following command should be issued from a TSO READY environment: TSOLIB ACTIVATE DATASET('NSQ.NSQLOAD')

The exec prompts you for a US city and, by requests to the TST1 NoSQLz system, displays all towns or cities of that name that can be found in the DBCITIES database.

For example:

```
TSO EX 'NSQ.NSQTEST(REXXCITY)'

51 states read

 Enter the name of a state / city / town / village in the USA:

new york

Searching for:  New York

 New York township              Missouri            00000  Population=269

 New York                       New York            00000  Population=19465197

 New York city                  New York            51000  Population=8244910

 New York County                New York            00000  Population=1601948

 ***
```

6. The verification procedure is terminated. You may try to test a more elaborate sample application, Wikitext (see page 39).

2.3.10 Database maintenance

In addition to routine backups, databases may require frequent clean-up, depending on the update activity they undergo. You should schedule batch jobs in this purpose. See also Database clean-up page 15.

3. Database management

3.1 Creating a database

To create a database, you need to fix the following parameters:

- the name of the database (8-byte name): this name will serve to uniquely identify the database in the NoSQLz system;
- the average size of a record (key + data);
- the maximum number of records to be stored in the database.

The database is tailored (using the NSQDUT20 utility), allocated on disk (using IDCAMS), and formatted (using the NSQDUT01 utility). Once formatted, it cannot take new extents: it is built for the maximum number of records that has been set (this number is actually rounded up to the next power of 2, so it can be larger than you may think). The average record size is always rounded up to a VSAM CI size (control interval size), because only one record is stored in a CI. You must keep that in mind when you plan to allocate a new database and understand the implications for your site as far as DASD storage is concerned.

A sample JCL may be found in NSQCNTL(DBINIT).

3.2 Declaring a database

Each database must be declared in the NSQPARM parameters by a DBNAME statement:

```
*---+----1----+----2----+----3----+----4----+----5----+----6----+

DBNAME   NSQCOMDS PROD1.DBCITIES                              W
```

The DBNAME statement consists of:

- the NoSQLz name of the database: up to 8 bytes, from column 10;
- the dsname of the associated VSAM file: up to 44 bytes, from column 19;
- the access option, that sets whether the database is accessed in read/write ('W') or in read-only ('R'): one byte at column 64.

See also sample statements in member NSQCNTL(NSQPARM).

3.3 Database clean-up

Databases never need reorganization. Because NoSQLz is based on timestamp-based concurrency control and multiversion concurrency control, databases need frequent clean-ups, using the **NSQDUT04** utility. The clean-up process will suppress the following entries in the databases:

- uncommitted records (records created but never committed)
- outdated records (old versions of records, no longer referenced in the database index)
- logically deleted records (DELETE function call)
- malformed records or records in error

A sample JCL may be found in NSQCNTL(DBCLEAN).

3.4 Database initial load

A database may be initially loaded by an application program with a series of **CREATE** function calls, or by using the **NSQDUT02** batch utility. It may also remain empty until online CREATE function calls populate it.

3.5 Database security

As for any dataset, access to databases is controlled based on the RACF userid that the access originates from. An open error will happen if the user is not authorized to access the dataset.

4. Function calls

4.1 Description

All function calls always require a "parameter block" (NSQ_BLOCK) as first parameter to specify the table involved, the key value, etc. This block is described in the **Appendix**. You must test the return code (NSQ_RETCD): a value of 0 (or 4) means the operation completed successfully (or partly).

A second parameter may also be provided, to obtain or write record data.

To invoke NoSQLz services, you must call the "transaction call API routine" **NSQTCAPI** that is provided in the NoSQLz LOAD library.

Example in Cobol (with 2 parameters):

```
CALL 'NSQTCAPI' USING NSQ-BLOCK, OUTZONE.
```

Example in assembler (with 2 parameters):

```
CALL NSQTCAPI,(NSQ_BLOCK,OUTZONE),VL
```

Example in REXX (with 2 parameters):

```
ADDRESS LINKPGM "NSQTCAPI NSQ_BLOCK OUTZONE"
```

Several NoSQLz systems may be active in your system. You have to specify which system your application program requires access to. This can be done by specifying in your JCL a DD statement in the following form:

```
//NSQLxxxx  DD   DUMMY
```

where "xxxx" stands for the name of the NoSQLz system (for example NSQLTST1 for the TST1 NoSQLz system). If you use REXX, you may also code a REXX statement as follows:

```
ADDRESS TSO  "ALLOC F(NSQLxxxx) DUMMY"
```

The basic element of processing is the task: a task can run only one transaction at a time. The transaction begins by any NoSQLz function call, and terminates either by a COMMIT function call or a ROLLBACK function call. After that, the task may start a new transaction or stop.

4.2 The READ function

The READ function is used to directly access a record by key.

The returned record is always 'older' than the transaction. You cannot read records created after the time your transaction began, or records currently modified by another transaction.

Parameters: NSQ_BLOCK and output zone.

Input data (NSQ_BLOCK):

NSQ_VRSN	'1' (1-byte literal)
NSQ_ACT	'READ ' (8-byte literal, left-justified)
NSQ_OPT	'N' (1-byte literal) to get the lastest version of the record
NSQ_TBLNM	Name of database (8 bytes, left-justified)
NSQ_KYLEN	Length of key (4-byte integer, value is 1 to 250)
NSQ_KYVAL	Value of key for record to search (length in bytes from 1 to NSQ_KYLEN)
NSQ_INLEN	Length of the provided output zone (4-byte integer)

'O' in the NSQ_OPT field can be used to retrieve the 'oldest' version of the record, as defined below:

- it is the same version as the current one, if no other version has been created after the initial CREATE operation for this key;

- it is the version that chronologically precedes the current one (because of an UPDATE operation for this key) if it has not been deleted by the clean-up utility (if it has been deleted, an error is returned: return code 8).

Output data (NSQ_BLOCK):

NSQ_RETCD	Return code (4-byte integer)
NSQ_MESSG	Error message (100 bytes)
NSQ_OUTLN	Number of bytes of data returned in the provided output zone (4-byte integer)

NSQ_RETCD will contain 0 if a whole record has been returned. NSQ_RETCD will contain 4 if the record has been truncated (output zone was too small, or the whole record could not be read).

NSQ_RETCD will contain 10 in the unlikely case that the record has been modified at least 2 times since your transaction started. You should restart the transaction.

Other values of NSQ_RETCD mean that no record matches the provided key, or that an error occurred.

Output (output zone): will contain the data part of the record, if the NSQ_RETCD returned is 0 or 4.

Samples: NSQTEST(ASMREAD), NSQTEST(COBREAD), NSQTEST(REXREAD).

4.3 The READSEQ function

The READSEQ function is used to sequentially read database records. Each READSEQ call returns one record, in no specific key order. The record key is returned in the NSQ_BLOCK while the data is returned in the output zone.

READSEQ does not necessarily provide a consistent view of the data, especially when the update rate is high. Records that are being updated and new records are not returned. You can never sure that you got all records of the database through one READSEQ loop.

Parameters: NSQ_BLOCK and output zone.

Input data (NSQ_BLOCK):

NSQ_VRSN	'1' (1-byte literal)
NSQ_ACT	'READSEQ ' (8-byte literal, left-justified)
NSQ_OPT	'N' (1-byte literal) to get the lastest version of the record
NSQ_TBLNM	Name of database (8 bytes, left-justified)
NSQ_INLEN	Length of the provided output zone (4-byte integer)

Output data (NSQ_BLOCK):

NSQ_RETCD	Return code (4-byte integer)
NSQ_MESSG	Error message (100 bytes)
NSQ_KYLEN	Length of key of returned record (4-byte integer, value is 1 to 250)
NSQ_KYVAL	Value of key of returned record (length is NSQ_KYLEN bytes)
NSQ_OUTLN	Number of bytes of data returned in the provided output zone (4-byte integer)

NSQ_RETCD will contain 0 if a whole record has been returned. NSQ_RETCD will contain 4 if the record has been truncated (output zone was too small, or the whole record could not be read). NSQ_RETCD will contain 14 when no more record can be returned because the end of the database has been reached.

Other values of NSQ_RETCD mean that an error occurred.

Output (output zone): will contain the data part of the record, if the NSQ_RETCD returned is 0 or 4.

Samples: NSQTEST(COBREADS), NSQTEST(REXREADS).

4.4 The READTEST function

The READTEST function is used to test whether a specify key exists in the database (and matches an existing record).

No record data is returned by this function.

Parameters: NSQ_BLOCK only.

Input data (NSQ_BLOCK):

NSQ_VRSN	'1' (1-byte literal)
NSQ_ACT	'READTEST' (8-byte literal, left-justified)
NSQ_TBLNM	Name of database (8 bytes, left-justified)
NSQ_KYLEN	Length of key (4-byte integer, value is 1 to 250)
NSQ_KYVAL	Value of key for record to search (length in bytes from 1 to NSQ_KYLEN)

Output data (NSQ_BLOCK):

NSQ_RETCD	Return code (4-byte integer)
NSQ_MESSG	Error message (100 bytes)

NSQ_RETCD will contain 0 if the specified key matches an existing record.

The NSQ_MESSG zone will display some technical information about the index entry of the key.

Other values of NSQ_RETCD mean that no record matches the provided key, or that an error occurred.

Samples: NSQTEST(COBREADT), NSQTEST(REXREADT).

4.5 The CREATE function

The CREATE function is used to create a new record in the database under a new key (the key does not match any existing record). The created record will not be visible to any user (except your application program) until the COMMIT function is called.

Parameters: NSQ_BLOCK and input zone.

Input data (NSQ_BLOCK):

NSQ_VRSN	'1' (1-byte literal)
NSQ_ACT	'CREATE ' (8-byte literal, left-justified)
NSQ_TBLNM	Name of database (8 bytes, left-justified)
NSQ_KYLEN	Length of key (4-byte integer, value is 1 to 250)
NSQ_KYVAL	Value of key for record to create (length in bytes from 1 to NSQ_KYLEN)
NSQ_INLEN	Length of the provided input zone (4-byte integer)

Input (input zone): contains the data part of the record to be created.

Output data (NSQ_BLOCK):

NSQ_RETCD	Return code (4-byte integer)
NSQ_MESSG	Error message (100 bytes)

NSQ_RETCD will contain 0 if the record has been created. Note that a subsequent COMMIT function call must be issued, otherwise the index of the database will not be updated and the record will be not retrieved again.

NSQ_RETCD will contain 9 if an existing record with the same key already exists.

NSQ_RETCD will contain 24 if no room has been found to insert the new record. Either the database is full, or too many uncommitted records with the same key have been created. In either case, the database should be cleaned up using the NSQDUT04 utility.

Other values of NSQ_RETCD mean that an error occurred.

Samples: NSQTEST(COBCREAT), NSQTEST(REXCREAT).

4.6 The UPDATE function

The UPDATE function is used to update an existing record. The updated record will not be visible to any user (except your application program) until the COMMIT function is called.

It need not be preceded by a READ function call, although it may generally be advisable.

Parameters: NSQ_BLOCK and input zone.

Input data (NSQ_BLOCK):

NSQ_VRSN	'1' (1-byte literal)
NSQ_ACT	'UPDATE ' (8-byte literal, left-justified)
NSQ_TBLNM	Name of database (8 bytes, left-justified)
NSQ_KYLEN	Length of key (4-byte integer, value is 1 to 250)
NSQ_KYVAL	Value of key for record to update (length in bytes from 1 to NSQ_KYLEN)
NSQ_INLEN	Length of the provided input zone (4-byte integer)

Input (input zone): contains the data part of the record to be updated.

Output data (NSQ_BLOCK):

NSQ_RETCD	Return code (4-byte integer)
NSQ_MESSG	Error message (100 bytes)

NSQ_RETCD will contain 0 if the record has been updated. Note that a subsequent COMMIT function call must be issued, otherwise the index of the database will not be updated and the previous version of the record will stay active.

NSQ_RETCD will contain 8 if the record has not been found.

NSQ_RETCD will contain 24 if no room has been found to update the record. Either the database is full, or too many uncommitted records with the same key have been created. In either case, the database should be cleaned up using the NSQDUT04 utility.

Other values of NSQ_RETCD mean that an error occurred.

Samples: NSQTEST(COBUPDAT), NSQTEST(REXUPDAT).

4.7 The DELETE function

The DELETE function is used to delete a specific key from the database. The effect of the delete operation will not be visible to any user (except your application program) until the COMMIT function is called.

Note that it is a logical delete, the matching record will be deleted physically by the NSQDUT04 clean-up utility. A READ function call with NSQ_OPT = 'N' (newest) will return an error, whereas a READ with NSQ_OPT = 'O' (oldest) will return the deleted record.

Parameters: NSQ_BLOCK only.

Input data (NSQ_BLOCK):

NSQ_VRSN	'1' (1-byte literal)
NSQ_ACT	'DELETE ' (8-byte literal, left-justified)
NSQ_TBLNM	Name of database (8 bytes, left-justified)
NSQ_KYLEN	Length of key (4-byte integer, value is 1 to 250)
NSQ_KYVAL	Value of key for record to update (length in bytes from 1 to NSQ_KYLEN)

Output data (NSQ_BLOCK):

NSQ_RETCD	Return code (4-byte integer)
NSQ_MESSG	Error message (100 bytes)

NSQ_RETCD will contain 0 if the record has been deleted. Note that a subsequent COMMIT function call must be issued, otherwise the index of the database will not be updated and the current version of the record will stay active.

NSQ_RETCD will contain 8 if the record has not been found.

Other values of NSQ_RETCD mean that an error occurred.

Samples: NSQTEST(COBDELET), NSQTEST(REXDELET).

4.8 The COMMIT function

The COMMIT function is used to end the transaction and to make the database updates (resulting from CREATE, UPDATE or DELETE function calls) visible to all users. It is assumed that your transaction leaves the databases in a consistent state between two commits.

Database positioning is not lost after COMMIT: for example, READSEQ may continue as if no COMMIT took place.

If your transaction does not end with a COMMIT call, any update previously done by a CREATE, UPDATE or DELETE function call will be ignored (actually, it will stay uncommitted and the next database clean-up will erase it).

The COMMIT process can only handle a limited number of updates, so an online transaction should generate only a few updates before committing.

Parameters: NSQ_BLOCK only.

Input data (NSQ_BLOCK):

NSQ_VRSN	'1' (1-byte literal)
NSQ_ACT	'COMMIT ' (8-byte literal, left-justified)

Output data (NSQ_BLOCK):

NSQ_RETCD	Return code (4-byte integer)
NSQ_MESSG	Error message (100 bytes)

NSQ_RETCD will contain 0 if the COMMIT call has been successful.

Other values of NSQ_RETCD mean that an error occurred. For example, there may be a concurrent access to a record key, or a record may have been modified after your transaction started. A COMMIT call is by no means assured of success, because of concurrent access.

4.9 The ROLLBACK function

The ROLLBACK function is used to end the transaction and to discard the database updates that it may have done.

Database positioning is not lost after ROLLBACK: for example, READSEQ may continue as if no ROLLBACK took place.

With the ROLLBACK function, any update previously done by a CREATE, UPDATE or DELETE function call is ignored (actually, it stays uncommitted and the next database clean-up will erase it).

Parameters: NSQ_BLOCK only.

Input data (NSQ_BLOCK):

NSQ_VRSN	'1' (1-byte literal)
NSQ_ACT	'ROLLBACK' (8-byte literal, left-justified)

Output data (NSQ_BLOCK):

NSQ_RETCD	Return code (4-byte integer)
NSQ_MESSG	Information message (100 bytes)

NSQ_RETCD will always contain 0.

4.10 The FREEALL function

The FREEALL function is used to free all databases that your task may have used till now. This function call should be used with care, because the FREEALL function call will close and free all databases that has been used by the current task. FREEALL may prove useful in a TSO/REXX environment, at the end of the REXX exec: the databases will not remain allocated to your TSO user. This function only affects the current address-space.

Parameters: NSQ_BLOCK only.

Input data (NSQ_BLOCK):

NSQ_VRSN	'1' (1-byte literal)
NSQ_ACT	'FREEALL' (8-byte literal, left-justified)

Output data (NSQ_BLOCK):

NSQ_RETCD	Return code (4-byte integer)
NSQ_MESSG	Information message (100 bytes)

NSQ_RETCD will always contain 0.

4.11 NSQAREXX: the REXX interface

Although the **NSQTCAPI** API routine may be directly called in REXX (using the LINKPGM host command environment), it may be preferable to use the **NSQAREXX** interface.

Syntax:

description of function call:	used for actions:
output = NSQAREXX()	returns the name of the current NoSQLz system (4 bytes) or x'00000000' if none established
output = NSQAREXX(action)	'COMMIT', 'ROLLBACK', 'FREEALL'
output = NSQAREXX(action, dbname, key)	'READ', 'READSEQ', 'READTEST', 'DELETE'
output = NSQAREXX(action, dbname, key, record_zone)	'CREATE', 'UPDATE'

Parameters:

parm#	definition	length	description or value
1	action	8	'READ', 'READSEQ', 'CREATE', 'DELETE', 'UPDATE', 'READTEST', 'COMMIT', 'ROLLBACK', 'FREEALL'.
2	database name	8	name of database as declared in NSQPARM
3	key	1-250	value of record key (required but ignored for 'READSEQ')
4	record zone	variable	mandatory for 'CREATE' and 'UPDATE' actions

Output:

action	returned data
'READ'	record (data part)
'READSEQ'	record (only the key part) or 108-byte error zone at end of data (RC=14)
'CREATE', 'DELETE', 'UPDATE'	x'00000000' (4 bytes) if operation successful
'READTEST', 'COMMIT', 'ROLLBACK', 'FREEALL'	100-byte message zone if operation successful

Error zone:

If the operation is unsuccessful, a 108-byte error zone is returned, the structure of which is:

- literal '**RC' (4 bytes)
- return code (binary value, 4 bytes)
- error message (text, 100 bytes)

Example:

```
SAY NSQAREXX('READTEST','DBWIKIPD','4D5653'X)
>RIG - SLOT=0000000224 ID=FBB9F5DEAB6F00EB9D7E5BA5F5 CELL=00CE59EC96FE8020010000
```

5. Managing the NoSQLz started task

5.1 NSQPARM parameters

The NSQPARM DD statement of the NSQTASK procedure points to a PDS member that contains all NoSQLz parameters ('NSQPARM parameters').

It is recommended to set the 4-byte NoSQLz system name as suffix of the member name, for example NSQP**PRD1** if the NoSQLz system name is **PRD1**.

All Sysplex members of the same NoSQLz complex must use the same set of NSQPARM parameters.

5.1.1 DBNAME

The DBNAME parameter is used to declare a NoSQLz database. See paragraph "**Declaring a database**" above.

5.1.2 COMTIME

The COMTIME parameter is used to set the maximum time application programs will wait for one COMMIT process to complete. When this time is exceeded, the COMMIT ends with a NSQ_RETCD return code 13 (NSQ_MESSG="NO ANSWER FROM COMMIT SUBTASK").

The COMTIME statement consists of:

- the maximum duration set for the COMMIT process, expressed in hundredths of a second: up to 10 numeric bytes, from column 10.

Example:

```
*---+----1----+----2----+----3----+----4----+----5----+----6----+----

COMTIME  1000                    MAX DURATION ALLOWED FOR COMMIT: 10 SEC.
```

5.2 Starting and stopping the started task

The member NSQCNTL(NSQTASK) should be copied into one of your system PROCLIBs.

The START command should be in the following form:

```
S NSQTASK,REUSASID=YES[,P=<NoSQLz system name]
```

The P= parameter stands for the name of your NoSQLz system. It should be a 4-byte name, for example TEST, or PRD1. Applications programs refer to this NoSQLz system name through a specific DD statement, for example:

```
//NSQLPRD1  DD   DUMMY  (for invoking the PRD1 NoSQLz system)
```

You may want to rename the NSQTASK to a name suffixed by the NoSQLz system name, for example NSQPRD1 if the NoSQLz system name is PRD1.

To stop the NoSQLz started task, you make use of the MVS STOP (P) command, for example:

```
P NSQTASK
```

Function calls will not work when the NoSQLz started task is inactive.

5.3 Commands to the started task

The MVS MODIFY (F) command can be used to display information or to modify the status of a database.

5.3.1 Display databases

```
F NSQTASK,LISTDB
```

This command will list all the databases declared to the NoSQLz system. For each database, the command displays: its dbname, access mode (R or W), dsname and the number of committed updates to this database.

```
f nsqtask,listdb

     NOSQLZ PRD1 DBCITIES W PROD.DBCITIES     0000000000

     NOSQLZ PRD1 DBCITIE2 W PROD.DBCITIE2     0000000051

     NOSQLZ PRD1 DBSTATES W PROD.DBSTATES     0000000000

     NOSQLZ PRD1 NSQCOMDS W PROD.COMDS        0000000000
```

5.3.2 Display current commit

```
F NSQTASK,LISTCC
```

This command will list the number of commits that have been achieved. If a commit is underway, it will display the time and date of the commit, and the involved job.

Example:

```
f nsqtask,listcc

     NOSQLZ PRD1 NUMBER OF COMMITS: 0000000001

     NOSQLZ PRD1 - NO COMMIT BEING DONE CURRENTLY
     NOSQLZ PRD1 - LAST COMMIT DONE FOR PROD02CC

f nsqtask,listcc

     NOSQLZ PRD1 NUMBER OF COMMITS: 0000000002

     NOSQLZ PRD1 COMMIT TIME=14090138 DATE=20140331 TOD=00CCEF7E0A085AD0

     NOSQLZ PRD1 FOR JOBNAME=PRODUPDT ASID=0039 (HEX)
```

5.3.3 Display tasks

```
F NSQTASK,LISTTSK
```

This command will list the subtasks of the started task, such as the commit task.

5.3.4 Display Sysplex information

```
F NSQTASK,LISTPLEX
```

This command will list all NoSQLz Sysplex members that are connected to this NoSQLz system. Current member is flagged with a '*'. The '0300' flag information indicates that the member is in active state.

Example:

```
f nsqtask,listplex

     NOSQLZ PRD1 SYSPLEX GROUP NSQGPRD1 CONTAINS 02 MEMBERS:

     NOSQLZ PRD1 *NSQGPRD1TST1     NSQTASK  01000069000E0001 0300

     NOSQLZ PRD1  NSQGPRD1TST2     NSQTASK2 0100001C000E0002 0300
```

5.3.5 Display log entries

```
F NSQTASK,LISTLOG
```

This command will list the more recent log entries, describing the recent events: start of transaction ('START'), transaction commit ('COMMIT'), end of transaction ('ENDnnn'), information about a commit operation occurring in another Sysplex member ('SYSPLX'). Displayed times are always UTC (GMT), not local.

"S=" displays the transaction start time, "C=" displays either the transaction commit time or the transaction end time, in tod clock format.

Example:

```
f nsqtask,listlog

     NOSQLZ PRD1 START  IBMUSER  S=CE42D0AB9C3560 11:23:45 C=00000000000000

     NOSQLZ PRD1 COMMIT IBMUSER  S=CE42D0AB9C3560 11:23:45 C=CE42D0ABF4A9B0
     NOSQLZ PRD1 END000 IBMUSER  S=CE42D0AB9C3560 11:23:45 C=CE42D0AC03CB60

     NOSQLZ PRD1 SYSPLX BTCHJOB  S=CE42D0ABF99B30 11:23:45 C=CE42D0ABF4A9B0

     NOSQLZ PRD1 SYSPLX BTCHJOB  S=CE42D0AC048570 11:23:45 C=00000000000000
```

5.3.6 Display communication area

```
F NSQTASK,LISTDATA
```

This command will list the NoSQLz communication area. It may be used for debugging.

5.3.7 Close and free a database

```
F NSQTASK,F <dbname>
```

This command will close and free the database whose 8-byte dbname is mentioned. It must be used if you want to delete or to rebuild the database. The NoSQLz LISTDB command will display the database with a "X" tag, meaning it is no longer available to application programs.

Example:

```
f nsqtask,f dbcities

     MAIN - DATABASE DBCITIES HAS BEEN FREED

f nsqtask,listdb

     NOSQLZ PRD1 DBCITIES W PROD.DBCITIES     0000000000  X

     NOSQLZ PRD1 NSQCOMDS W PROD.COMDS        0000000000
```

5.3.8 Allocate and open a database

```
F NSQTASK,A <dbname>
```

This command will allocate and open the database whose 8-byte dbname is mentioned. The database must have been freed by a previous F command. This command is required if you want to put the database in use again. The database must have been previously declared in the NSQPARM parameters. You cannot put in use a database that has not been declared.

Example:

```
f nsqtask,a dbcities

     DBAS - ALLOCATING DBCITIES DSN=PROD.DBCITIES

     MAIN - DATABASE DBCITIES READY, DSN=PROD.DBCITIES
```

6. Batch utilities

Several batch utilies help you manage databases. All batch utilites run "offline", i.e. they don't need that the NoSQLz started task be active.

All input statements of batch utilities must being at column 1.

6.1 NSQDUT01: database formatting

This utility formats a new NoSQLz database. The database must have been previously created and must be empty.

The occurrence of a IEC161I message during the process is normal.

6.1.1 DD statements

SYSIN

Mandatory. Specifies the input dataset that contains the control statements.

NSQOUT

Mandatory. Specifies the dataset to be formatted. It must have been previously allocated by a DEFINE CLUSTER IDCAMS command.

NSQPRINT

Optional but recommended. Specifies the output dataset that receives all execution messages.

6.1.2 Control statements

TBLNAME=<name of database>

Mandatory. Specifies the 8-byte name of the database.

HASHALG=n

Optional. Specifies the hash algorithm chosen for the database. Three values are supported:

1: SHA-1 (default)

2: SHA-256

3: SHA-512

KEYHASHMASK=<8-byte hexadecimal value>

DATAHASHMASK=<8-byte hexadecimal value>

Mandatory. These parameters specify hash masks to apply for the index part and the data part of the database. Values depend on the number of records and the record size. You should let the ZCOMP exec set those parameters (see the sample JCL).

6.1.3 Sample JCL

See NSQCNTL(DBINIT).

6.2 NSQDUT02: database initial load

This utility loads data into a NoSQLz database that has previously been formatted by the NSQDUT01 utility. The database (ddname NSQOUT) needs not be empty. It is assumed that the database is not under the control of the NoSQLz started task, otherwise concurrent updating may take place and corrupt the data.

6.2.1 DD statements

SYSIN

Mandatory. Specifies the input dataset that contains the control statements.

NSQOUT

Mandatory. Specifies the NoSQLz dataset to receive data. It must have been previously formatted by NSQDUT01.

SYSUT1

Mandatory. Specifies the input sequential dataset that contains data to load into the NoSQLz dataset.

SYSUTERR

Optional. Specifies an output sequential dataset that will receive SYSUT1 records that could not be inserted in the NSQOUT dataset. This enables you to try to load again the remaining records.

NSQPRINT

Optional but recommended. Specifies the output dataset that receives all execution messages.

6.2.2 Control statements

KEYLEN=<key size>

Size of the key part of the record. This size is assumed to be fixed, and the key is assumed to be located at start of record. The key length must be from 1 to 250.

DATALEN=<data size>

Size of the data part of the record. This size is assumed to be fixed, and the key is assumed to be located at start of record. The data length must be from 1 to 32760.

DUP=INSERT/IGNORE

This option specifies whether duplicate keys are to be ignored (DUP=IGNORE) or to be inserted as new versions to override existing keys (DUP=INSERT).

Default is INSERT.

6.2.3 Sample JCL

See NSQCNTL(DBLOAD).

6.3 NSQDUT03: printing database statistics

This utility reads an existing database to provide varied statistics. Statistics are written in the NSQPRINT data set. For example:

```
NSQDUT03 2015.021 23:31:01 SOFT01BL STATS            z/OS 020100 SYS1     SYSPLX
- LRECL=00505 CISZ=00512 HIGH-ALLOC-RBA= 100853760 (000000000602E800 HEX)
- TBL=DBCITIES INDEX-MASK=0000FFFF DATA-MASK=0001FFFF HSHALGO=03
**  DATA SLOT NUMBER RANGE: 0000065538 - 0000196609
**   DATABASE RECORDS READ: 0000076120
**         - INDEX RECORDS: 0000032049 / 0000065536  48%
**         -  DATA RECORDS: 0000044069 / 0000131071  33%
** MAX NUMBER OF KEY CELLS: 006 / 017 IN SLOT: 0000007618
**    MAX DATA SLOT NUMBER: 0000196601
**  HIGHEST VERSION NUMBER: 001
**  HIGHEST VALUE FOR SALT: 07 (HEX)
**        OLDEST TIMESTAMP: 20150120 15201535
**        LATEST TIMESTAMP: 20150120 15344303
```

This is especially of interest to get the occupation rate of the database (index part and data part).

An 'INDEX RECORDS: ... 100 %' message is not an issue in itself as long as the 'MAX NUMBER OF KEY CELLS nnn/mmm' message displays a current value (nnn) not close to the maximum (mmm).

A 'DATA RECORDS: ' message that shows a percentage close to 100 % should entice you to run the cleanup utility.

6.3.1 DD statements

NSQIN

Mandatory. Specifies the NoSQLz dataset to process.

NSQPRINT

Optional but recommended. Specifies the output dataset that receives all execution messages.

6.3.2 Sample JCL

See NSQCNTL(DBSTATS).

6.4 NSQDUT04: database clean-up

This utility cleans up an existing database to automatically suppress unneeded records and keep only the last version of the records. It deletes uncommitted data and outdated data (except for recent records, depending on the LIMIT value). It also displays final statistics, for example:

```
NSQDUT04 2015.023 20:30:00 SOFT01CL CLEAN            z/OS 020100 SYS1     SYSPLX
 - DATA RECORDS READ      :0000125811
 - DATA RECORDS DELETED   :0000081754
 -         (NO INDEX ENTRY:0000000002)
 -               (OUTDATED:0000081742)
 -      (LOGICALLY DELETED:0000000010)
```

6.4.1 DD statements

SYSIN

Optional. Specifies the input dataset that contains the control statements.

NSQIN

Mandatory. Specifies the NoSQLz dataset to process.

NSQPRINT

Optional but recommended. Specifies the output dataset that receives all execution messages.

6.4.2 Control statements (optional)

SIMULATE=Y/N

Specifies whether physical deletes are carried on or not. Default is N (deletes are physically done).

LIMIT=<number of hours>

The utility must not delete the records that have been created less than <number of hours> ago. This number must be from 0 to 10000. Default is 1 (one hour).

With LIMIT=0, all unneeded records will be suppressed, no matter when they were created. This is in general not advisable, because uncommitted records may be suppressed just before the transaction COMMIT takes place.

6.4.3 Sample JCL

See NSQCNTL(DBCLEAN).

6.5 NSQDUT10: database record display (from key)

This utility reads a record from a database, based on a key value. It reads the more recent version of the record, and the oldest, should it exist.

6.5.1 DD statements

SYSIN

Mandatory. Specifies the input dataset that contains the control statements.

NSQIN

Mandatory. Specifies the NoSQLz dataset to read data from.

NSQPRINT

Optional but recommended. Specifies the output dataset that receives all execution messages.

6.5.2 Control statements

KEYLEN=<key size>

Size of the record key specified in the V= statement. It must be from 1 to 250.

A value of 0 has a special meaning: it means you want to read all records sequentially. Because it is a physical read, even outdated or uncommitted records are accessed and displayed, so duplicate keys may be listed if no clean-up has been done in the meantime. KEYLEN=0 should be used with care, since it may result in a large amount of data being displayed. For example:

```
//READ1    EXEC PGM=NSQDUT10

//NSQIN    DD   DISP=SHR,DSN=PROD.DBSTATES

//NSQPRINT DD   SYSOUT=*

//SYSIN    DD   *

KEYLEN=0
```

results in the following display:

```
+DUT10 - 0000000182 V=01 S=00 KL=075 K=Georgia

+DUT10 - 0000000187 V=01 S=01 KL=075 K=Vermont

+DUT10 - 0000000190 V=01 S=01 KL=075 K=Idaho
```

V=<key value>

Value of the key for the record to be searched. Up to 78 bytes of key may be specified.

DELETE=Y/N

Specifies whether the record that matches the key must be logically deleted or not (default: N).

6.5.3 Sample JCL

See NSQCNTL(RECREAD).

6.6 NSQDUT11: database record display (from record-id)

This utility reads a record from a database, based on its record-id. It displays the index entry that matches the record-id. It is equivalent to a READTEST function call.

The record-id associated with a given record key is a 26-digit hexadecimal value that uniquely identifies the record key in the database.

6.6.1 DD statements

SYSIN

Mandatory. Specifies the input dataset that contains the control statements.

NSQIN

Mandatory. Specifies the NoSQLz dataset to read data from.

NSQPRINT

Optional but recommended. Specifies the output dataset that receives all execution messages.

6.6.2 Control statements

RECID=<record_id (26 hexadecimal digits)>

6.6.3 Sample JCL

See NSQCNTL(RECRBYID).

6.7 NSQDUT20: database tailoring

This utility generates all the parameters required to create a new database. Those parameters will be further used by IDCAMS (disk file allocation) and NSQDUT01 (NoSQLz database preformatting).

6.7.1 DD statements

NSQSYSIN

Mandatory. Specifies the input dataset that contains the control statements.

NSQOUTP

Optional. Specifies an output sequential dataset (LRECL=80) that will receive statements (KEYHASHMASK= and DATAHASHMASK=) destined to the NSQDUT01 utility.

NSQOUTI

Optional. Specifies an output sequential dataset (LRECL=80) that will receive statements destined to the IDCAMS utility. These statements will be part of a AMS DEFINE CLUSTER command. The generated AMS DEFINE CLUSTER parameters are: RECORDS, RECORDSIZE, CISZ, NUMBERED, SHR.

NSQPRINT

Optional but recommended. Specifies the output dataset that receives all execution messages.

6.7.2 Control statements

RECLEN=<record_length> (theorical maximum record size, from 1 to 75497472)

Note that once the database is created, records with size greater than RECLEN may be written. RECLEN stands in fact for the maximum record size of a vast majority of the records. This parameter is used to determine the CI size of the database and the required allocation space for the database.

Note also that because RECLEN is rounded to upper value, the real record size may be larger than RECLEN (as displayed in the "- POTENTIAL RECORD LENGTH" message).

RECNUM=<total_number_of_records> (from 1 to 2147483648)

Note that the maximum number of records (or rather keys) during the lifetime of the database can never exceed RECNUM (rounded to the upper power of 2, as displayed in the "- ADJUSTED NUMBER OF KEYS" message).

6.7.3 Sample JCL

See NSQCNTL(DBINIT).

6.7.4 Example

Initializing a database with the following parameters:

```
RECLEN=15000
RECNUM=100000
```

will result in a database larger than 4 GB. The reasons for this allocation are these:

- the 15000-byte RECLEN will be rounded up to 16359

- the number of records (100000) will be rounded up to 262144 to accomodate for updates.

The generated AMS commands might look like this:

```
DEF  CL(NAME('NSQ.SMS.DBLARGE') -

     DATACLASS(VSAMEXT)   -

     RECORDS(00262657 00000000) NUMBERED SHR(4 4) -

     RECORDSIZE(16377 16377) CISZ(16384))
```

The mentioned dataclass (VSAMEXT here) should support extended addressability.

Such class may be defined as follows:

```
Data Class Name : VSAMEXT

Data Set Name Type  . . . : EXTENDED

  If Extended . . . . . . : REQUIRED

  Extended Addressability : YES
```

7. Wikitext: a sample NoSQLz application

Wikitext is a set of sample programs written in REXX to illustrate how NoSQLz can be used to quickly write a database application. The purpose is to display under TSO some excerpts of the well-known Internet encyclopedia, Wikipedia.

All samples can be found in the dedicated NSQ.NSQWIKI library.

7.1 License

'Wikipedia as text' is derived from Wikipedia. As such, it is distributed under the *Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0)* license.

'Wikipedia as text' is described in the following article: Pataki, M., Vajna, M. and Marosi, A.– Wikipedia as Text. *Ercim News - Special theme: Big Data*. 2012, Vols. 89, pp. 48-49.

7.2 Data denormalization

Typical of the NoSQL paradigm is data denormalization: to enhance the read performance, the databases contain redundant data.

A normalized table such as the following one:

Employee table (Employee number is the primary key, Department is a foreign key)

Employee number	Last name	First name	Department
000005	Smith	John	A00
000122	Brown	Peter	A00
000569	Smith	Paul	B01

could possibly be denormalized as follows:

Table 1 (Employee number is the table key, the other fields of the original table are values)

Key	Value: last name	Value: first name	Value: department
000005	Smith	John	A00
000122	Brown	Peter	A00
000569	Smith	Paul	B01

Table 2 (Employee's last name is the table key, the value is the employee number or numbers)

Key	Value: employee number(s)
Smith	000005, 000569
Brown	000122

Table 3 (Department is the table key, the value is the employee number)

Key	Value: employee number(s)
A00	000005, 000122
B01	000569

Another possibility would be to merge Table 1 and Table 2 into the same table, mimicking "columns" ('EMPNO' for Employee number, 'EMPNAME' for Employee name):

Table 1-2

Key	Value	
EMPNO-000005	Smith, John, A00	**Value: last name, first name, department**
EMPNO-000122	Brown, Peter, A00	
EMPNO-000569	Smith, Paul, B01	
EMPNAME-Smith	000005, 000569	**Value: employee number(s)**
EMPNAME-Brown	000122	

7.3 Database design and application elements

Two NoSQLz databases can address the application purpose:

Database	Description	Record Key	Record Data
DBWIKIDI	Contains the names of the articles that share the same digram (the same first characters). Example: 'Octal', 'Octet', 'Ocean', 'Oco' will be part of	digram (the 2 first characters of the article name) Example: 'Oc'	One-byte binary zone, followed by all names, in no order. All names have the same length, this length is stored in the one-byte binary zone. Names are right-padded by

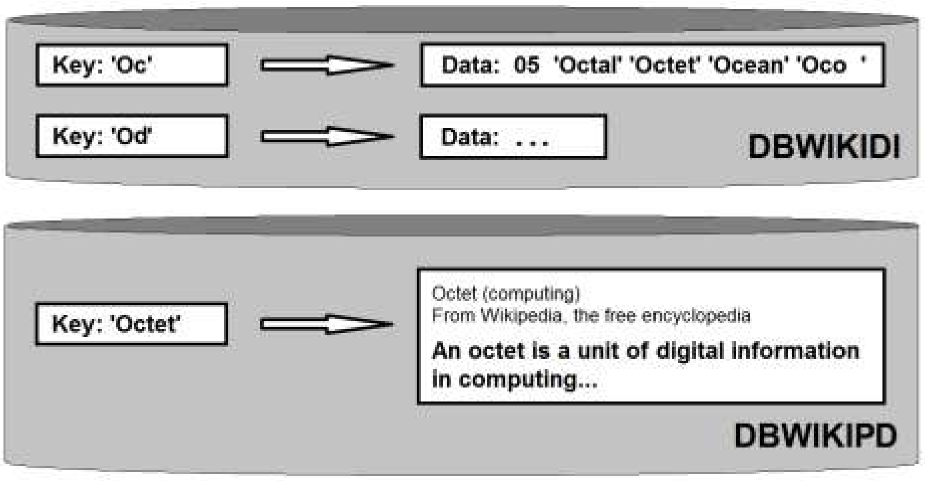
	the same logical record (common key '**Oc**')		blanks. Example: X'05', 'Octal', 'Octet', 'Ocean', 'Oco '
DBWIKIPD	Contains the text of the articles (one logical record per article) Example: text of the Wikipedia article named 'Octet'	name (variable-length) of the article Example: 'Octet'	Text of the article (variable-length)

The DBWIKIDI digram database is destined to provide a high-level index, to be search in order to access all existing article names by their two first characters.

The DBWIKIPD database is destined to store all articles, whatever their size.

All data remains stored in ASCII, the REXX application translates the articles in EBCDIC before displaying them.

The figure below shows the structure of the databases.


Key: 'Oc'
Data: 05 'Octal' 'Octet' 'Ocean' 'Oco '
Key: 'Od'
Data: ...
DBWIKIDI
Key: 'Octet'
Octet (computing)
From Wikipedia, the free encyclopedia
An octet is a unit of digital information in computing...
DBWIKIPD

The WIKIREXX application will first display a list of all digrams stored in the DBWIKIDI database. After a digram is selected, all names that match this digram are displayed. The user may then select one name to display the text of the corresponding article.

The programs of NSQ.NSQWIKI that compose the application are as follows:

name	type	description
WIKIREXX	REXX exec	highest level program to display digrams and articles

WIKIRX2	REXX exec	display list of articles named by one digram
WIKIRX3	REXX exec	display the article chosen by the end user
WIKIRXLD	REXX exec	database initial loading
WIKIPAN1	ISPF panel	panel to display all digrams
WIKIPAN2	ISPF panel	panel to display list of articles
WIKIPAN3	ISPF panel	panel to display article text

The JCLs provided in NSQ.NSQWIKI are as follows:

name	**description**
ALLOCDB	Create and initialize database DBWIKIPD
ALLOCDI	Create and initialize database DBWIKIDI
LOADDB	Load some Wikipedia articles into the database
STATS	Statistics about the databases

The external functions used by the application are these:

name	**description**
NSQAREXX	NoSQLz interface for REXX
NSQACA2E	ASCII to EBCDIC character conversion
NSQAUTF8	Conversion of some UTF-8 characters to readable form

7.4 Installing the application

The ALLOCDB and ALLOCDI JCLs of NSQ.NSQWIKI must be used to allocate and initialize the databases. The DBWIKIPD database is tailored for a limited number of records. The RECNUM parameter should be increased to accommodate for a larger number of records.

After initialization, the databases must be declared in the NSQPARM parameters:

```
*---+----1----+----2----+----3----+----4----+----5----+----6----+
DBNAME   DBWIKIPD TST1.DBWIKIPD                                W
DBNAME   DBWIKIDI TST1.DBWIKIDI                                W
```

The NSQTASK started task may then be started to bring the databases online.

7.5 Loading Wikipedia text

You may upload to the mainframe any file available on the Wikipedia as text web site. You'll end up with gzip files such as: 20140615-wiki-en_000031.txt.gz.

Each gzip file must be transferred in binary (no ASCII translation, no CRLF) to a MVS RECFM=VB existing dataset with any LRECL (preferably large).

The LOADDB JCL of NSQ.NSQWIKI must be adapted and run to load the Wikipedia material into the two databases. Note that a clean-up step is required for the DBWIKIDI database, because of very frequent updates.

7.6 Using the application

The WIKIREXX REXX exec should be customized as follows:

- the NSQLSYS variable must reflect the NoSQLz system name (as specified in the NSQTASK started task)

- the THISDS variable must reflect the name of the NSQ.NSQWIKI dataset.

The WIKIREXX REXX exec can now be executed under ISPF.

A first menu is displayed for the user to choose a digram.

```
------------------------ NoSQLz - Wikitext application ---- Row 18 to 27 of 27
Option ===> _____________________________________________________ Scroll: CSR_
                    Wikipedia, the free encyclopedia                    IBMUSER
                     Creative Commons BY-SA license
 S   Digram  Hexa
 _     MP    4D50
 _     MU    4D55
 S     MV    4D56
 _     M1    4D31
 _     M8    4D38
 _     Pr    5072
 _     Sh    5368
 _     Sy    5379
 _     Un    556E
 _     Wi    5769
******************************* Bottom of data ********************************
```

The next menu displays all articles that match this digram.

```
------------------------ NoSQLz - Wikitext application ------- Row 1 to 2 of 2
Option ===> _____________________________________________________ Scroll: CSR_
                                                                       IBMUSER
   Line commands: S=Select
     Digram is 'MV' X'4D56'
 _   MV Tampa
 S   MVS
******************************* Bottom of data ********************************
```

The last menu displays the text of the article.

```
------------------------ NoSQLz - Wikitext application ---- Row 1 to 21 of 433
Option ===> _____________________________________________________ Scroll: CSR_
 MVS


 Multiple Virtual Storage, more commonly called MVS, was the most commonly
 used operating system on the System/370 and System/390 IBM mainframe
 computers. It was developed by IBM, but is unrelated to IBM's other
 mainframe operating systems, e.g., VSE, VM, TPF.
```

7.7 Some programming details about the application

All REXX execs use the NSQAREXX interface to access the databases.

The WIKIREXX REXX sequentially reads the DBWIKIDI database. The digrams are retrieved in no specific order, the ISPEXEC TBSORT function is actually invoked to sort them in alphabetic order.

When one DBWIKIDI record is selected, all matching article names are obtained from this record and displayed (WIKIRX2 EXEC).

Finally, when an article is selected, the text is retrieved in the DBWIKIPD database, converted from ASCII to EBCDIC, formatted in lines and displayed as an ISPF table (WIKIRX3 EXEC).

8. Appendix

8.1 The parameter block

The parameter block must be provided for each function call.

8.1.1 Cobol description

Here is an example for a parameter block whereby a table named MYTABLE is accessed to retrieve ('READ') a record whose key is 'MYKEY' (length 5 bytes). The newest ('N') version of the record is required.

```
01  NSQ-BLOCK.
    02 NSQ-VRSN             PIC X    VALUE '1'.
    02 NSQ-OPT              PIC X    VALUE 'N'.
    02 NSQ-ACT              PIC X(8) VALUE 'READ'.
    02 NSQ-TBLNM            PIC X(8) VALUE 'MYTABLE'.
    02 NSQ-INLEN            PIC S9(5) COMP VALUE 100.
    02 NSQ-OUTLN            PIC S9(5) COMP.
    02 NSQ-RETCD            PIC S9(5) COMP VALUE 0.
    02 NSQ-MESSG            PIC X(100) VALUE SPACES.
    02 NSQ-KYLEN            PIC S9(2) COMP VALUE 5.
    02 NSQ-KYVAL.
       05 NSQ-KYVAL05       PIC X(5) VALUE 'MYKEY'.
       05 FILLER            PIC X(245).
```

8.1.2 Assembler description

```
NSQ_BLOCK EQU  *
NSQ_VRSN  DS   CL1           VERSION OF PARAMETER BLOCK: C'1'
NSQ_OPT   DS   CL1           READ OPTION: 'N'EWEST (RECOMMENDED)
NSQ_ACT   DS   CL8        ACTION ("START", "CREATE", "READ", "UPDATE",
*                                "DELETE", "COMMIT", "ROLLBACK")
NSQ_TBLNM DS   CL8           NAME OF TABLE DATA SET TO BE ACCESSED
* DATA ZONE INFORMATION
NSQ_INLEN DS   CL4   IN      LENGTH OF DATA ZONE SUPPLIED BY CALLER
*                            (FOR ALL OPERATIONS)
NSQ_OUTLN DS   CL4   OUT     LENGTH OF DATA COPIED INTO ZONE BY NOSQLZ
*                            (ONLY FOR "READ" OPERATIONS)
* ERROR INFORMATION
NSQ_RETCD DS   CL4           RETURN CODE
NSQ_MESSG DS   CL100         ERROR MESSAGE (IF RETURN CODE NOT ZERO)
* KEY PARAMETER
NSQ_KYLEN DS   CL2           LENGTH OF KEY (1-250)
NSQ_KYVAL DC   XL250'00'     VALUE OF KEY
```

8.2 Return codes

Here is a list of return codes you may find in field NSQ_RETCD. Additional information is also provided in the NSQ_MESSG field.

Code (decimal)	Code (hexadecimal)	Meaning
0	00	Operation successful
4	04	Record partially read (READ-type operation)
8	08	Record not found
9	09	Record already exists, cannot be created
10	0A	Record not accessible, being updated
12	0C	Record has been logically deleted
13	0D	COMMIT error, cannot modify/delete record
14	0E	End of file (sequential processing)
15	0F	READ option is in error
16	10	VSAM error when accessing database record
20	14	Key length passed is zero or > 250
22	16	NOSQLZ system not found or not active
23	17	Database not found (no identify point)
24	18	Database is full, cannot insert index / data record
25	19	Database in read-only, write not allowed
27	1B	Cannot get CTRL info (hash) from db
28	1C	Cannot get dsname of db (undefined db)
29	1D	Database name error, dbname differs from tbname
30	1E	Name/token create or retrieve error
32	20	Routine error: NSQTENTY routine not found, or parameter block version error, or parameter block action unknown, or parameter block

		action unsupported, or transaction commit time not set
34	22	Database error: incorrect module (OODAT), or allocation error, or open error
36	24	Zone passed by caller is unusable
40	28	Started task NOSQLZ missing

9. Readers' Comments — We'd Like to Hear from You

NoSQLz - v2 User's Guide, 2015 edition

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses.

Name ____________________ Address ______________________________

Company or Organization ______________________________

Phone No. ______________________________