**>>>>>>>>>>>>>>>vasuki**

**CHECK INDEX**

The CHECK INDEX online utility tests whether indexes are consistent with the data that they index, and issues warning messages when it finds an inconsistency.

Run the CHECK INDEX utility after a conditional restart or a point-in-time recovery on all table spaces whose indexes might not be consistent with the data.

Also run CHECK INDEX before running CHECK DATA, especially if you specify DELETE YES. Running CHECK INDEX before CHECK DATA ensures that the indexes that CHECK DATA uses are valid. When checking an auxiliary table index, CHECK INDEX verifies that each LOB is represented by an index entry, and that an index entry exists for every LOB.

**Important:** Inaccurate statistics for tables, table spaces, or indexes can result in a sort failure during CHECK INDEX.

Start of change

**Running CHECK INDEX when the index has a VARBINARY column**

If you run CHECK INDEX against the index with the following characteristics, CHECK INDEX fails:

* The index was created on a VARBINARY column or a column with a distinct type that is based on a VARBINARY data type.
* The index column has the DESC attribute.

To fix the problem, alter the column data type to BINARY, and then rebuild the index.

End of change

**Output**

CHECK INDEX generates several messages that show whether the indexes are consistent with the data.

For unique indexes, any two null values are treated as equal values, unless the index was created with the UNIQUE WHERE NOT NULL clause. In that case, if the key is a single column, it can contain any number of null values, and CHECK INDEX does not issue an error message.

CHECK INDEX issues an error message if it finds two or more null values and the unique index was not created with the UNIQUE WHERE NOT NULL clause.

**Authorization required**

To execute this utility, you must use a privilege set that includes one of the following authorities:

* STATS privilege for the database
* DBADM, DBCTRL, or DBMAINT authority for the database. If the object on which the utility operates is in an implicitly created database, DBADM authority on the implicitly created database or DSNDB04 is required.
* SYSCTRL or SYSADM authority

An ID with installation SYSOPR authority can also execute CHECK INDEX, but only on a table space in the DSNDB01 or DSNDB06 databases.

Start of changeIf you are using SHRLEVEL CHANGE, the batch user ID that invokes COPY with the CONCURRENT option must provide the necessary authority to execute the DFDSS ADRSSU command. DFDSS will create a shadow data set with the authority of the utility batch address space. The submitter should have an RACF® ALTER authority, or its equivalent, for the shadow data set.End of change

**Execution phases of CHECK INDEX**

**Phase**

**Description**

**UTILINIT**

Performs initialization

**UNLOAD**

Unloads index entries

**SORT**

Sorts unloaded index entries

**CHECKIDX**

Scans data to validate index entries

**UTILTERM**

Performs cleanup

**Jcl-vasuki**

**Sample CHECK INDEX control statements**

Use the sample control statements as models for developing your own CHECK INDEX control statements.

**Example 1: Checking all indexes**

The control statement specifies that the CHECK INDEX utility is to check all indexes in sample table space DSN8D81A.DSN8S81E.

//STEP1 EXEC DSNUPROC,UID='IUIQU1UQ.CHK1',

// UTPROC='',

// SYSTEM='DSN'

//SYSUT1 DD DSN=IUIQU1UQ.CHK3.STEP1.SYSUT1,DISP=(MOD,DELETE,CATLG),

// UNIT=SYSDA,SPACE=(8000,(200,20),,,ROUND)

//SYSERR DD DSN=IUIQU1UQ.CHK3.SYSERR,DISP=(MOD,DELETE,CATLG),

// UNIT=SYSDA,SPACE=(6000,(20,20),,,ROUND)

//SORTOUT DD DSN=IUIQU1UQ.CHK3.STEP1.SORTOUT,DISP=(MOD,DELETE,CATLG),

// UNIT=SYSDA,SPACE=(6000,(20,20),,,ROUND)

//SYSIN DD \*

CHECK INDEX (ALL) TABLESPACE DSN8D91A.DSN8S91E

//\*

**Example 2: Checking one index**

The following control statement specifies that the CHECK INDEX utility is to check the project-number index (DSN8910.XPROJ1) on the sample project table. SORTDEVT SYSDA specifies that SYSDA is the device type for temporary data sets that are to be dynamically allocated by DFSORT™.

CHECK INDEX (DSN8910.XPROJ1)

SORTDEVT SYSDA

**Example 3: Checking more than one index**

The following control statement specifies that the CHECK INDEX utility is to check the indexes DSN8910.XEMPRAC1 and DSN8910.XEMPRAC2 on the employee-to-project-activity sample table.

CHECK INDEX NAME (DSN8910.XEMPRAC1, DSN8910.XEMPRAC2)

**Example 4: Checking partitions of all indexes**

In the following control statement, table space DB0S0301.TP0S0301 has one partitioned index (ADMF001.IP0S0301), one data-partitioned secondary index (ADMF001.ID0S0302), and one nonpartitioned secondary index (ADMF001.IX0S0303). The (ALL) option indicates that all three indexes on the table space are to be checked. PART 3 indicates that CHECK INDEX is to check the third physical partition of any partitioned indexes and the third logical partition of any nonpartitioned indexes.

CHECK INDEX(ALL) TABLESPACE DBOS0301.TPOS0301 PART 3 SORTDEVT SYSDA

In this case, CHECK INDEX checks the third physical partition of ADMF001.IP0S0301, the third physical partition of ADMF001.ID0S0302, and the third logical partition of ADMF001.IX0S0303, as indicated by the following output.

*Figure 1. CHECK INDEX output from a job that checks the third partition of all indexes.*

DSNU050I DSNUGUTC- CHECK INDEX(ALL) TABLESPACE DBOS0301.TPOS0301 PART 3 SORTDEVT SYSDA

DSNU700I= DSNUKGET- 10 INDEX ENTRIES UNLOADED FROM INDEX='ADMF001.IPOS0301' PARTITION=3

DSNU700I= DSNUKGET- 10 INDEX ENTRIES UNLOADED FROM INDEX='ADMF001.IDOS0302' PARTITION=3

DSNU701I= DSNUKGET- 10 INDEX ENTRIES UNLOADED FROM 'ADMF001.IXOS0303'

DSNU705I DSNUK001- UNLOAD PHASE COMPLETE - ELAPSED TIME=00:00:00

DSNU717I= DSNUKTER- 10 ENTRIES CHECKED FOR INDEX 'ADMF001.IPOS0301' PARTITION=3

DSNU717I= DSNUKTER- 10 ENTRIES CHECKED FOR INDEX 'ADMF001.IDOS0302' PARTITION=3

DSNU717I= DSNUKTER- 10 ENTRIES CHECKED FOR INDEX 'ADMF001.IXOS0303' PARTITION=3

DSNU720I DSNUK001- CHECKIDX PHASE COMPLETE, ELAPSED TIME=00:00:00

DSNU010I DSNUGBAC- UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0

**Example 5: Checking indexes in a list**

The LISTDEF control statement defines a list of indexes called CHKIDXB\_LIST. The CHECK INDEX control statement specifies that CHECK INDEX is to check all indexes that are included in the CHKIDXB\_LIST list. WORKDDN SYSUT1 specifies that SYSUT1 is the DD name of the temporary work file for sort input; SYSUT1 is the default. SORTDEVT SYSDA specifies that SYSDA is the device type for temporary data sets that are to be dynamically allocated by DFSORT. SORTNUM 4 specifies that four of these data sets are to be dynamically allocated.

*Figure 2. Example of checking indexes in a list*Start of change

//CHKIDXB EXEC PGM=DSNUTILB,REGION=4096K,PARM='SSTR,CHKINDX1'

//SYSPRINT DD SYSOUT=A

//SYSUDUMP DD SYSOUT=A

//UTPRINT DD SYSOUT=A

//DSNTRACE DD SYSOUT=A

//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(5,2)),VOL=SER=SCR03

//SYSOUT DD UNIT=SYSDA,SPACE=(CYL,(5,2)),VOL=SER=SCR03

//SORTLIB DD DISP=SHR,DSN=SYS1.SORTLIB

//SORTOUT DD UNIT=SYSDA,SPACE=(CYL,(5,2)),VOL=SER=SCR03

//SYSERR DD UNIT=SYSDA,SPACE=(CYL,(5,2)),VOL=SER=SCR03

//SYSIN DD \*

LISTDEF CHKIDXB\_LIST INCLUDE INDEXSPACE DBOT55\*.\* ALL

CHECK INDEX LIST CHKIDXB\_LIST

WORKDDN SYSUT1

SORTDEVT SYSDA

SORTNUM 4

/\*

End of change

**Example 6: Checking all specified indexes on clone tables**

Start of changeThe following control statement specifies that the CHECK INDEX utility is to check all specified indexes that are on clone tables. End of change

CHECK INDEX (ALL) TABLESPACE DBLOB01.TSLOBC4 CLONE

**>>>>>>>>>>>>>>>vasuki**

**LOAD**

Use LOAD to load one or more tables of a table space. The LOAD utility loads records into the tables and builds or extends any indexes that are defined on them. If the table space already contains data, you can choose whether you want to add the new data to the existing data or replace the existing data.

The loaded data is processed by any edit or validation routine that is associated with the table, and any field procedure that is associated with any column of the table. The LOAD utility ignores and does not enforce informational referential constraints.

**Output**

LOAD DATA generates one or more of the following forms of output:

* A loaded table space or partition.
* A discard file of rejected records.
* A summary report of errors that were encountered during processing; this report is generated only if you specify ENFORCE CONSTRAINTS or if the LOAD involves unique indexes.

**Authorization required**

To execute this utility, you must use a privilege set that includes one of the following authorizations:

* Ownership of the table
* LOAD privilege for the database
* SYSCTRL or SYSADM authority
* STATS privilege for the database is required if STATISTICS keyword is specified

LOAD operates on a table space level, so you must have authority for all tables in the table space when you perform LOAD.

To run LOAD STATISTICS, the privilege set must include STATS authority on the database. To run LOAD STATISTICS REPORT YES, the privilege set must also include the SELECT privilege on the tables required.

If you use RACF® access control with multilevel security and LOAD is to process a table space that contains a table that has multilevel security with row-level granularity, you must be identified to RACF and have an accessible valid security label. You must also meet the following authorization requirements:

* To replace an entire table space with LOAD REPLACE, you must have the write-down privilege unless write-down rules are not in effect.
* You must have the write-down privilege to specify values for the security label columns, unless write-down rules are not in effect. If these rules are in effect and you do not have write-down privilege, DB2® assigns your security label as the value for the security label column for the rows that you are loading.

**Execution phases of LOAD**

The LOAD utility operates in the phases that are listed in the following table.

| *Table 1. LOAD phases of execution* | |
| --- | --- |
| **Phase** | **Description** |
| UTILINIT | Performs initialization. |
| RELOAD | Loads record types and writes temporary file records for indexes and foreign keys. RELOAD makes one pass through the sequential input data set. Check constraints are checked for each row. Internal commits provide commit points at which to restart in case operation should halt in this phase.  RELOAD creates inline copies if you specified the COPYDDN or RECOVERYDDN keywords.  A subtask is started at the beginning of the RELOAD phase to sort the keys. The sort subtask initializes and waits for the main RELOAD phase to pass its keys to SORT. RELOAD loads the data, extracts the keys, and passes them in memory for sorting. At the end of the RELOAD phase, the last key is passed to SORT, and record sorting completes.  Note that load partition parallelism starts subtasks. PREFORMAT for table spaces occurs at the end of the RELOAD phase. |
| SORT | Sorts temporary file records before creating indexes or validating referential constraints, if indexes or foreign keys exist. The SORT phase is skipped if all the following conditions apply for the data that is processed during the RELOAD phase:   * Each table has no more than one key. * All keys are the same type (index key only, indexed foreign key, or foreign key only). * The data that is being loaded or reloaded is in key order (if a key exists). If the key is an index key only and the index is a data-partitioned secondary index, the data is considered to be in order if the data is grouped by partition and ordered within partition by key value. If the key in question is an indexed foreign key and the index is a data-partitioned secondary index, the data is never considered to be in order. * The data that is being loaded or reloaded is grouped by table, and each input record is loaded into one table only.   SORT passes the sorted keys in memory to the BUILD phase, which builds the indexes. |
| BUILD | Creates indexes from temporary file records for all indexes that are defined on the loaded tables. Build also detects duplicate keys. PREFORMAT for indexes occurs at the end of the BUILD phase. |
| SORTBLD | Performs all activities that normally occur in both the SORT and BUILD phases, if you specify a parallel index build. |
| INDEXVAL | Corrects unique index violations or index evaluation errors from the information in SYSERR, if any exist. |
| ENFORCE | Checks referential constraints, except informational referential constraints, and corrects violations. Information about violations of referential constraints is stored in SYSERR. |
| DISCARD | Copies records that cause errors from the input data set to the discard data set. |
| REPORT | Generates a summary report, if you specified ENFORCE CONSTRAINT or if load index validation is performed. The report is sent to SYSPRINT. |
| UTILTERM | Performs cleanup. |

**>>>>>>>>>>>>>>>vasuki**

**Load -jcl**

**Sample LOAD control statements**

Use the sample control statements as models for developing your own LOAD control statements.

**Example 1: Specifying field positions**

The control statement specifies that the LOAD utility is to load the records from the data set that is defined by the SYSREC DD statement into table DSN8810.DEPT. SYSREC is the default input data set.

Each POSITION clause specifies the location of a field in the input record. In this example, LOAD accepts the input that is shown in [Figure 2](http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.ugref/db2z_loadsamples.htm#db2z_loadsamples__inputr) and interprets it as follows:

* The first 3 bytes of each record are loaded into the DEPTNO column of the table.
* The next 36 bytes, including trailing blanks, are loaded into the DEPTNAME column.

If this input column were defined as VARCHAR(36), the input data would need to contain a 2-byte binary length field preceding the data. This binary field would begin at position 4.

* The next three fields are loaded into columns that are defined as CHAR(6), CHAR(3), and CHAR(16).

The RESUME YES clause specifies that the table space does not need to be empty; new records are added to the end of the table.

*Figure 1. Example of a LOAD statement that specifies field positions*

LOAD DATA

RESUME YES

INTO TABLE DSN8910.DEPT

(DEPTNO POSITION (1:3) CHAR(3),

DEPTNAME POSITION (4:39) CHAR(36),

MGRNO POSITION (40:45) CHAR(6),

ADMRDEPT POSITION (46:48) CHAR(3),

LOCATION POSITION (49:64) CHAR(16))

[Figure 2](http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.ugref/db2z_loadsamples.htm#db2z_loadsamples__inputr). shows the input to the preceding LOAD job.

*Figure 2. Records in an input data set for LOAD*

A00SPIFFY COMPUTER SERVICE DIV. 000010A00USIBMSTODB21

B01PLANNING 000020A00USIBMSTODB21

C01INFORMATION CENTER 000030A00USIBMSTODB21

D01DEVELOPMENT CENTER A00USIBMSTODB21

The following table shows the result of executing the statement SELECT \* FROM DSN8910.DEPT after the preceding input records are loaded.

| *Table 1. Data that is loaded into a table* | | | | |
| --- | --- | --- | --- | --- |
| **DEPTNO** | **DEPTNAME** | **MGRNO** | **ADMRDEPT** | **LOCATION** |
| A00 | SPIFFY COMPUTER SERVICE DIV. | 000010 | A00 | USIBMSTODB21 |
| B01 | PLANNING | 000020 | A00 | USIBMSTODB21 |
| C01 | INFORMATION CENTER | 000030 | A00 | USIBMSTODB21 |
| D01 | DEVELOPMENT CENTER |  | A00 | USIBMSTODB21 |

**Example 2: Replacing data in a given partition**

The following control statement specifies that data from the data set that is defined by the SYSREC DD statement is to be loaded into the first partition of table DSN8810.DEPT. The default input data set is SYSREC. The REPLACE option indicates that the input data is to replace only the specified partition. If the REPLACE option was specified before the PART option, REPLACE would indicate that entire table space is to be replaced, and the data is to be loaded into the specified partition. Note that the keyword DATA does not need to be specified.

LOAD

INTO TABLE DSN8910.DEPT PART 1 REPLACE

**Example 3: Loading selected records into multiple tables**

The control statement in specifies that the LOAD utility is to load certain data from the EMPLDS input data set into tables DSN8910.EMP, SMITH.EMPEMPL, and DSN8810.DEPT. The input data set is identified by the INDDN option. The WHEN clauses indicate which records are to be loaded into each table. For the EMP and DEPT tables, the utility is to load only records that begin with the string LKA. For the EMPEMPL table, the utility is to load only records that begin with the string ABC. The RESUME YES option indicates that the table space does not need to be empty for the LOAD job to proceed. The new rows are added to the end of the tables. This example assumes that the first two tables being loaded have exactly the same forma, and that the input data matches that format; therefore, no field specifications are needed for those two INTO TABLE clauses. The third table has a different format, so field specifications are required and are supplied in the example.

The POSITION clauses specify the location of the fields in the input data for the DEPT table. For each source record that is to be loaded into the DEPT table:

* The characters in positions 7 through 9 are loaded into the DEPTNO column.
* The characters in positions 10 through 35 are loaded into the DEPTNAME column.
* The characters in positions 36 through 41 are loaded into the MGRNO column.
* The characters in positions 42 through 44 are loaded into the ADMRDEPT column.

*Figure 3. Example LOAD statement that loads selected records into multiple tables*

LOAD DATA INDDN EMPLDS

RESUME YES

INTO TABLE DSN8910.EMP

WHEN (1:3)='LKA'

INTO TABLE SMITH.EMPEMPL

WHEN (1:3)='ABC'

INTO TABLE DSN8910.DEPT

WHEN (1:3)='LKA'

(DEPTNO POSITION (7:9) CHAR,

DEPTNAME POSITION (10:35) CHAR,

MGRNO POSITION (36:41) CHAR,

ADMRDEPT POSITION (42:44) CHAR)

**Example 4: Loading data of different data types**

The control statement specifies that LOAD is to load data from the SYSRECPJ input data set into table DSN8910.PROJ. The input data set is identified by the INDDN option. Assume that the table space that contains table DSN8910.PROJ is currently empty.

For each input record, data is loaded into the specified columns (that is, PROJNO, PROJNAME, DEPTNO, and so on) to form a table row. Any other PROJ columns that are not specified in the LOAD control statement are set to the default value.

The POSITION clauses define the starting positions of the fields in the input data set. The ending positions of the fields in the input data set are implicitly defined either by the length specification of the data type (CHAR *length*) or the length specification of the external numeric data type (LENGTH).

The numeric data that is represented in SQL constant format (EXTERNAL format) is converted to the correct internal format by the LOAD process and placed in the indicated column names. The two dates (PRSTDATE and PRENDATE) are assumed to be represented by eight digits and two separator characters, as in the USA format (for example, 11/15/2006). The length of the date fields is given as 10 explicitly, although in many cases, the default is the same value.

*Figure 4. Example of loading data of different data types*

LOAD DATA INDDN(SYSRECPJ)

INTO TABLE DSN8910.PROJ

(PROJNO POSITION (1) CHAR(6),

PROJNAME POSITION (8) CHAR(22),

DEPTNO POSITION (31) CHAR(3),

RESPEMP POSITION (35) CHAR(6),

PRSTAFF POSITION (42) DECIMAL EXTERNAL(5),

PRSTDATE POSITION (48) DATE EXTERNAL(10),

PRENDATE POSITION (59) DATE EXTERNAL(10),

MAJPROJ POSITION (70) CHAR(6))

**Example 5: Loading data in delimited file format**

The control statement specifies that data in delimited format is to be loaded into the specified columns (FILENO, DATE1, TIME1, and TIMESTMP) in table TBQB0103. The FORMAT DELIMITED option indicates that the data is in delimited format. The data is to be loaded from the SYSREC data set, which is the default.

The COLDEL option indicates that the column delimiter is a comma (,). The CHARDEL option indicates that the character string delimiter is a double quotation mark ("). The DECPT option indicates that the decimal point character is a period (.). You are not required to explicitly specify these particular characters, because they are all defaults.

*Figure 5. Example of loading data in delimited file format*

//\*

//STEP3 EXEC DSNUPROC,UID='JUQBU101.LOAD2',TIME=1440,

// UTPROC='',

// SYSTEM='SSTR'

//SYSERR DD DSN=JUQBU101.LOAD2.STEP3.SYSERR,

// DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,

// SPACE=(4096,(20,20),,,ROUND)

//SYSDISC DD DSN=JUQBU101.LOAD2.STEP3.SYSDISC,

// DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,

// SPACE=(4096,(20,20),,,ROUND)

//SYSMAP DD DSN=JUQBU101.LOAD2.STEP3.SYSMAP,

// DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,

// SPACE=(4096,(20,20),,,ROUND)

//SYSUT1 DD DSN=JUQBU101.LOAD2.STEP3.SYSUT1,

// DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,

// SPACE=(4096,(20,20),,,ROUND)

//UTPRINT DD SYSOUT=\*

//SORTOUT DD DSN=JUQBU101.LOAD2.STEP3.SORTOUT,

// DISP=(MOD,DELETE,CATLG),UNIT=SYSDA,

// SPACE=(4096,(20,20),,,ROUND)

//SYSIN DD \*

LOAD DATA

FORMAT DELIMITED COLDEL ',' CHARDEL '"' DECPT '.'

INTO TABLE TBQB0103

(FILENO CHAR,

DATE1 DATE EXTERNAL,

TIME1 TIME EXTERNAL,

TIMESTMP TIMESTAMP EXTERNAL)

/\*

//SYSREC DD \*

"001", 2000-02-16, 00.00.00, 2000-02-16-00.00.00.0000

"002", 2001-04-17, 06.30.00, 2001-04-17-06.30.00.2000

"003", 2002-06-18, 12.30.59, 2002-06-18-12.30.59.4000

"004", 1991-08-19, 18.59.30, 1991-08-19-18.59.30.8000

"005", 2000-12-20, 24.00.00, 2000-12-20-24.00.00.0000

/\*

**Example 6: Concatenating multiple input records**

The control statement specifies that data from the SYSRECOV input data set is to be loaded into table DSN8910.TOPTVAL. The input data set is identified by the INDDN option. The table space that contains the TOPTVAL table is currently empty.

Some of the data that is to be loaded into a single row spans more than one input record. In this situation, an X in column 72 indicates that the input record contains fields that are to be loaded into the same row as the fields in the next input record. In the LOAD control statement, CONTINUEIF(72:72)='X' indicates that LOAD is to concatenate any input records that have an X in column 72 with the next record before loading the data.

For each assembled input record (that is, after the concatenation), fields are loaded into the DSN8910.TOPTVAL table columns (that is, MAJSYS, ACTION, OBJECT …, DSPINDEX) to form a table row. Any columns that are not specified in the LOAD control statement are set to the default value.

The POSITION clauses define the starting positions of the fields in the assembled input records. Starting positions are numbered from the first column of the internally assembled input record, not from the start of the input records in the sequential data set. The ending positions of the fields are implicitly defined by the length specification of the data type (CHAR *length*).

No conversions are required to load the input character strings into their designated columns, which are also defined to be fixed-length character strings. However, because columns INFOTXT, HELPTXT, and PFKTXT are defined as 79 characters in length and the strings that are being loaded are 71 characters in length, those strings are padded with blanks as they are loaded.

*Figure 6. Example of concatenating multiple input records before loading the data*

LOAD DATA INDDN(SYSRECOV) CONTINUEIF(72:72)='X'

INTO TABLE DSN8910.TOPTVAL

(MAJSYS POSITION (2) CHAR(1),

ACTION POSITION (4) CHAR(1),

OBJECT POSITION (6) CHAR(2),

SRCHCRIT POSITION (9) CHAR(2),

SCRTYPE POSITION (12) CHAR(1),

HEADTXT POSITION (80) CHAR(50),

SELTXT POSITION (159) CHAR(50),

INFOTXT POSITION (238) CHAR(71),

HELPTXT POSITION (317) CHAR(71),

PFKTXT POSITION (396) CHAR(71),

DSPINDEX POSITION (475) CHAR(2))

**Example 7: Loading null values**

The control statement specifies that data from the SYSRECST data set is to be loaded into the specified columns in table SYSIBM.SYSSTRINGS. The input data set is identified by the INDDN option. The NULLIF option for the ERRORBYTE and SUBBYTE columns specifies that if the input field contains a blank, LOAD is to place a null value in the indicated column for that particular row. The DEFAULTIF option for the TRANSTAB column indicates that the utility is to load the default value for this column if the input field value is GG. The CONTINUEIF option indicates that LOAD is to concatenate any input records that have an X in column 80 with the next record before loading the data.

*Figure 7. Example of loading null values*

LOAD DATA INDDN(SYSRECST) CONTINUEIF(80:80)='X' RESUME(YES)

INTO TABLE SYSIBM.SYSSTRINGS

(INCCSID POSITION( 1) INTEGER EXTERNAL(5),

OUTCCSID POSITION( 7) INTEGER EXTERNAL(5),

TRANSTYPE POSITION( 13) CHAR(2),

ERRORBYTE POSITION( 16) CHAR(1) NULLIF(ERRORBYTE=' '),

SUBBYTE POSITION( 18) CHAR(1) NULLIF(SUBBYTE=' '),

TRANSPROC POSITION( 20) CHAR(8),

IBMREQD POSITION( 29) CHAR(1),

TRANSTAB POSITION( 31) CHAR(256) DEFAULTIF(TRANSTYPE='GG'))

**Example 8: Enforcing referential constraints when loading data**

The control statement specifies that data from the SYSREC input data set is to be loaded into table DSN8910.PROJ. The default input data set is SYSREC. The table space that contains the PROJ table is not empty. RESUME YES indicates that the records are to be added to the end of the table.

The ENFORCE CONSTRAINTS option indicates that LOAD is to enforce referential constraints on the data that is being added. This option is also the default. All violations are reported in the output. All records causing these violations are not loaded and placed in the SYSDISC data set, which is the default data set for discarded records.

The CONTINUEIF option indicates that before loading the data LOAD is to concatenate any input records that have an X in column 72 with the next record.

*Figure 8. Example of enforcing referential constraints when loading data*

LOAD DATA INDDN(SYSREC) CONTINUEIF(72:72)='X'

RESUME YES

ENFORCE CONSTRAINTS

INTO TABLE DSN8910.PROJ

(PROJNO POSITION (1) CHAR (6),

PROJNAME POSITION (8) VARCHAR,

DEPTNO POSITION (33) CHAR (3),

RESPEMP POSITION (37) CHAR (6),

PRSTAFF POSITION (44) DECIMAL EXTERNAL (5),

PRSTDATE POSITION (50) DATE EXTERNAL,

PRENDATE POSITION (61) DATE EXTERNAL,

MAJPROJ POSITION (80) CHAR (6) NULLIF(MAJPROJ=' '))

**Example 9: Loading data without enforcing referential constraints**

The control statement specifies that data from the SYSRECAC input data set is to be loaded into table DSN8810.ACT. The INDDN option identifies the input data set.

ENFORCE NO indicates that the LOAD utility is not to enforce referential constraints and places the table in CHECK-pending status. Use this option if you are loading data into several tables that are related in such a way that the referential constraints cannot be checked until all tables are loaded. For example, a column in table A depends on a column in table B; a column in table B depends on a column in table C; and a column in table C depends on a column in table A.

The POSITION clauses define the starting positions of the fields in the input data set. The ending positions of the fields in the input data set are implicitly defined by the length specification of the data type (CHAR *length*). In this case, the characters in positions 1 through 3 are loaded into the ACTNO column, the characters in positions 5 through 10 are loaded into the ACTKWD column, and the characters in position 13 onward are loaded into the ACTDESC column. Because the ACTDESC column is of type VARCHAR, the input data needs to contain a 2-byte binary field that contains the length of the character field. This binary field begins at position 13.

*Figure 9. Example of loading data without enforcing referential constraints*

//STEP1 EXEC DSNUPROC,UID='IUIQU2UB.LOAD',

// UTPROC='',

// SYSTEM='DSN'

//SYSRECAC DD DSN=IUIQU2UB.LOAD.DATA,DISP=SHR,VOL=SER=SCR03,

// UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)

//SYSUT1 DD DSN=IUIQU2UB.LOAD.STEP1.SYSUT1,

// DISP=(MOD,DELETE,CATLG),

// UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)

//SORTOUT DD DSN=IUIQU2UB.LOAD.STEP1.SORTOUT,

// DISP=(MOD,DELETE,CATLG),

// UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)

//SYSIN DD \*

LOAD DATA INDDN(SYSRECAC) RESUME YES

INTO TABLE DSN8910.ACT

(ACTNO POSITION(1) INTEGER EXTERNAL(3),

ACTKWD POSITION(5) CHAR(6),

ACTDESC POSITION(13) VARCHAR)

ENFORCE NO

//\*

**Example 10: Loading data by using a parallel index build**

The control statement specifies that data from the SYSREC input data set is to be loaded into table DSN8810.DEPT. Assume that 22 000 rows need to be loaded into table DSN8910.DEPT, which has three indexes. In this example, the SORTKEYS option is used to improve performance by forcing a parallel index build. The SORTKEYS option specifies 66 000 as an estimate of the number keys to sort in parallel during the SORTBLD phase. (This estimate was computed by using the calculation that is described in [Improved performance with SORTKEYS](http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.ugref/db2z_improveperfsortkeys.htm#db2z_improveperfsortkeys).) Because more than one index needs to be built, LOAD builds the indexes in parallel.

The SORTDEVT and SORTNUM keywords specify that DFSORT™ is to dynamically allocate the required data sets. If sufficient virtual storage resources are available, one utility subtask pair is started to build each index. This example does not require UTPRIN*nn* DD statements because it uses DSNUPROC to invoke utility processing, which includes a DD statement that allocates UTPRINT to SYSOUT.

The CONTINUEIF option indicates that, before loading the data, LOAD is to concatenate any input records that have a plus sign (+) in column 79 and a plus sign (+) in column 80 with the next record.

*Figure 10. Example of loading data by using a parallel index build*

//SAMPJOB JOB …

//STEP1 EXEC DSNUPROC,UID='SAMPJOB.LOAD',UTPROC='',SYSTEM='DSN'

//SORTOUT DD DSN=SAMPJOB.LOAD.STEP1.SORTOUT,DISP=(MOD,DELETE,CATLG),

// UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)

//SYSUT1 DD DSN=SAMPJOB.LOAD.STEP1.SYSUT1,DISP=(MOD,DELETE,CATLG),

// UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)

//SYSERR DD DSN=SAMPJOB.LOAD.STEP1.SYSERR,DISP=(MOD,DELETE,CATLG),

// UNIT=SYSDA,SPACE=(2000,(20,20),,,ROUND)

// DCB=(RECFM=FB,LRECL=80,BLKSIZE=2400)

//SYSMAP DD DSN=SAMPJOB.LOAD.STEP1.SYSMAP,DISP=(MOD,DELETE,CATLG),

// UNIT=SYSDA,SPACE=(2000,(20,20),,,ROUND),

// DCB=(RECFM=FB,LRECL=80,BLKSIZE=2400)

//SYSREC DSN=SAMPJOB.TEMP.DATA,DISP=SHR,UNIT=SYSDA

//SYSIN DD \*

LOAD DATA REPLACE INDDN SYSREC CONTINUEIF(79:80)='++'

SORTKEYS 66000 SORTDEVT SYSDA SORTNUM 3

INTO TABLE DSN8910.DEPT

/\*

**Example 11: Creating inline copies**

The control statement specifies that the LOAD utility is to load data from the SYSREC data set into the specified columns of table ADMF001.TB0S3902.

COPYDDN(COPYT1) indicates that LOAD is to create inline copies and write the primary image copy to the data set that is defined by the COPYT1 template. This template is defined in one of the preceding TEMPLATE control statements. To create an inline copy, you must also specify the REPLACE option, which indicates that any data in the table space is to be replaced.

CONTINUEIF(79:80)='++' indicates that, before loading the data, LOAD is to concatenate any input records that have a plus sign (+) in column 79 and a plus sign (+) in column 80 with the next record.

The ERRDDN(ERRDDN) and MAPDDN(MAP) options indicate that information about errors is to be written to the data sets that are defined by the ERRDDN and MAP templates. DISCARDDN(DISCARD) specifies that discarded records (those that violate referential constraints) are to be written to the data set that is defined by the DISCARD template. WORKDDN(UT1,OUT) specifies the temporary work files for sort input and output; LOAD is to use the data set that is defined by the UT1 template for sort input and the data set that is defined by the OUT template for sort output.

*Figure 11. Example of creating inline copies*

//STEP1 EXEC DSNUPROC,UID='JUOSU339.LOAD1',TIME=1440,

// UTPROC='',

// SYSTEM='SSTR'

//SYSREC DD DSN=CUST.FM.CINT135.DATA,DISP=SHR,VOL=SER=FORDMD,

// UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)

//SYSIN DD \*

TEMPLATE ERRDDN UNIT(SYSDA)

DSN(JUOSU339.T&TI..&ST..ERRDDN)

SPACE(50,10) TRK

TEMPLATE UT1 UNIT(SYSDA)

DSN(JUOSU339.T&TI..&ST..SYSUT1)

SPACE(50,10) TRK

TEMPLATE OUT UNIT(SYSDA)

DSN(JUOSU339.T&TI..&ST..SYSOUT)

SPACE(50,10) TRK

TEMPLATE MAP UNIT(SYSDA)

DSN(JUOSU339.T&TI..&ST..SYSMAP)

SPACE(50,10) TRK

TEMPLATE DISCARD UNIT(SYSDA)

DSN(JUOSU339.T&TI..&ST..DISCARD)

SPACE(50,10) TRK

TEMPLATE COPYT1

UNIT(SYSDA)

DSN(JUOSU339.COPY1.STEP1.&SN..COPY&LR.&PB.)

DISP(MOD,CATLG,CATLG)

SPACE(60,30) TRK

LOAD DATA INDDN SYSREC REPLACE

CONTINUEIF(79:80)='++'

COPYDDN(COPYT1)

ERRDDN(ERRDDN)

WORKDDN(UT1,OUT)

MAPDDN(MAP)

DISCARDDN(DISCARD)

INTO TABLE

ADMF001.TBOS3902

( ID\_PARTITION POSITION(1) CHAR(1),

CD\_PLANT POSITION(2) CHAR(5),

NO\_PART\_BASE POSITION(7) CHAR(9),

NO\_PART\_PREFIX POSITION(16) CHAR(7),

NO\_PART\_SUFFIX POSITION(23) CHAR(8),

NO\_PART\_CONTROL POSITION(31) CHAR(3),

DT\_TRANS\_EFFECTIVE POSITION(34) DATE EXTERNAL(10),

CD\_INV\_TRANSACTION POSITION(44) CHAR(3),

TS\_PROCESS POSITION(47) TIMESTAMP EXTERNAL(26),

QT\_INV\_TRANSACTION POSITION(73) INTEGER,

CD\_UNIT\_MEAS\_USAGE POSITION(77) CHAR(2),

CD\_USER\_ID POSITION(79) CHAR(7),

NO\_DEPT POSITION(86) CHAR(4),

NO\_WORK\_CENTER POSITION(90) CHAR(6))

/\*

**Example 12: Collecting statistics**

This example is similar to the previous example, except that the STATISTICS option and other related options have been added so that during the LOAD job, DB2® also gathers statistics for the table space. Gathering these statistics eliminates the need to run the RUNSTATS utility after completing the LOAD operation.

The TABLE, COLUMN, and INDEX options specify that information is to be gathered for columns QT\_INV\_TRANSACTION, NO\_DEPT, NO\_PART\_PREFIX, DT\_TRANS\_EFFECTIVE and index ID0S3902 for table TB0S3902. SAMPLE 53 indicates that LOAD is to sample 53% of the rows when gathering non-indexed column statistics. For the index, the KEYCARD option specifies that all of the distinct values in all of the key column combinations are to be collected. FREQVAL NUMCOLS 4 COUNT 20 indicates that 20 frequent values are to be collected on the concatenation of the first four key columns.

REPORT YES indicates that the statistics are to be sent to SYSPRINT as output. UPDATE ALL and HISTORY ALL indicate that all collected statistics are to be updated in the catalog and catalog history tables.

*Figure 12. Example of collecting statistics*

//STEP1 EXEC DSNUPROC,UID='JUOSU339.LOAD1',TIME=1440,

// UTPROC='',

// SYSTEM='SSTR'

//SYSREC DD DSN=CUST.FM.CINT135.DATA,DISP=SHR,VOL=SER=FORDMD,

// UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)

//SYSIN DD \*

TEMPLATE ERRDDN UNIT(SYSDA)

DSN(JUOSU339.T&TI..&ST..ERRDDN)

SPACE(50,10) TRK

TEMPLATE UT1 UNIT(SYSDA)

DSN(JUOSU339.T&TI..&ST..SYSUT1)

SPACE(50,10) TRK

TEMPLATE OUT UNIT(SYSDA)

DSN(JUOSU339.T&TI..&ST..SYSOUT)

SPACE(50,10) TRK

TEMPLATE MAP UNIT(SYSDA)

DSN(JUOSU339.T&TI..&ST..SYSMAP)

SPACE(50,10) TRK

TEMPLATE DISCARD UNIT(SYSDA)

DSN(JUOSU339.T&TI..&ST..DISCARD)

SPACE(50,10) TRK

TEMPLATE COPYT1

UNIT(SYSDA)

DSN(JUOSU339.COPY1.STEP1.&SN..COPY&LR.&PB.)

DISP(MOD,CATLG,CATLG)

SPACE(60,30) TRK

LOAD DATA INDDN SYSREC REPLACE

CONTINUEIF(79:80)='++'

COPYDDN(COPYT1)

STATISTICS

TABLE (TBOS3902) SAMPLE 53

COLUMN (QT\_INV\_TRANSACTION,

NO\_DEPT,

NO\_PART\_PREFIX,

DT\_TRANS\_EFFECTIVE)

INDEX (IDOS3902 KEYCARD

FREQVAL NUMCOLS 4 COUNT 20)

REPORT YES UPDATE ALL HISTORY ALL

ERRDDN(ERRDDN)

WORKDDN(UT1,OUT)

MAPDDN(MAP)

DISCARDDN(DISCARD)

INTO TABLE

ADMF001.TBOS3902

( ID\_PARTITION POSITION(1) CHAR(1),

CD\_PLANT POSITION(2) CHAR(5),

NO\_PART\_BASE POSITION(7) CHAR(9),

NO\_PART\_PREFIX POSITION(16) CHAR(7),

NO\_PART\_SUFFIX POSITION(23) CHAR(8),

NO\_PART\_CONTROL POSITION(31) CHAR(3),

DT\_TRANS\_EFFECTIVE POSITION(34) DATE EXTERNAL(10),

CD\_INV\_TRANSACTION POSITION(44) CHAR(3),

TS\_PROCESS POSITION(47) TIMESTAMP EXTERNAL(26),

QT\_INV\_TRANSACTION POSITION(73) INTEGER,

CD\_UNIT\_MEAS\_USAGE POSITION(77) CHAR(2),

CD\_USER\_ID POSITION(79) CHAR(7),

NO\_DEPT POSITION(86) CHAR(4),

NO\_WORK\_CENTER POSITION(90) CHAR(6))

/\*

**Example 13: Loading Unicode data**

The following control statement specifies that Unicode data from the REC1 input data set is to be loaded into table ADMF001.TBMG0301. The UNICODE option specifies the type of input data. Only data that satisfies the condition that is specified in the WHEN clause is to be loaded. The CCSID option specifies the three coded character set identifiers for the input file: one for SBCS data, one for mixed data, and one for DBCS data. LOG YES indicates that logging is to occur during the LOAD job.

LOAD DATA INDDN REC1 LOG YES REPLACE

UNICODE CCSID(00367,01208,01200)

INTO TABLE "ADMF001 "."TBMG0301"

WHEN(00004:00005 = X'0003')

**Example 14: Loading data from multiple input data sets by using partition parallelism**

The LOAD control statement in this example contains a series of INTO TABLE statements that specify which data is to be loaded into which partitions of table DBA01.TBLX3303. For each INTO TABLE statement:

* Data is to be loaded into the partition that is identified by the PART option. For example, the first INTO TABLE statement specifies that data is to be loaded into the first partition of table DBA01.TBLX3303.
* Data is to be loaded from the data set that is identified by the INDDN option. For example, the data from the PART1 data set is to be loaded into the first partition.
* Any discarded rows are to be written to the data set that is specified by the DISCARDDN option. For example, rows that are discarded during the loading of data from the PART1 data set are written to the DISC1 data set.
* The data is loaded into the specified columns (EMPNO, LASTNAME, and SALARY).

LOAD uses partition parallelism to load the data into these partitions.

The TEMPLATE utility control statement defines the data set naming convention for the data set that is to be dynamically allocated during the following LOAD job. The name of the template is ERR3. The ERRDDN option in the LOAD statement specifies that any errors are to be written to the data set that is defined by this ERR3 template.

*Figure 13. Example of loading data from individual data sets*

TEMPLATE ERR3

DSN &UT..&JO..&ST..ERR3&MO.&DAY.

UNIT SYSDA DISP(NEW,CATLG,CATLG)

LOAD DATA

REPLACE

ERRDDN ERR3

INTO TABLE DBA01.TBLX3303

PART 1

INDDN PART1

DISCARDDN DISC1

(EMPNO POSITION(1) CHAR(6),

LASTNAME POSITION(8) VARCHAR(15),

SALARY POSITION(25) DECIMAL(9,2))

.

.

.

INTO TABLE DBA01.TBLX3303

PART 5

INDDN PART5

DISCARDDN DISC5

(EMPNO POSITION(1) CHAR(6),

LASTNAME POSITION(8) VARCHAR(15),

SALARY POSITION(25) DECIMAL(9,2))

/\*

**Example 15: Loading data from another table in the same system by using a declared cursor**

The following LOAD control statement specifies that all rows that are identified by cursor C1 are to be loaded into table MYEMP. The INCURSOR option is used to specify cursor C1, which is defined in the EXEC SQL utility control statement. Cursor C1 points to the rows that are returned by executing the statement SELECT \* FROM DSN8810.EMP. In this example, the column names in table DSN8810.EMP are the same as the column names in table MYEMP. Note that the cursor cannot be defined on the same table into which DB2 is to load the data.

EXEC SQL

DECLARE C1 CURSOR FOR SELECT \* FROM DSN8810.EMP

ENDEXEC

LOAD DATA

INCURSOR(C1)

REPLACE

INTO TABLE MYEMP

STATISTICS

**Example 16: Loading data partitions in parallel from a remote site by using a declared cursor**

The LOAD control statement in this example specifies that for each specified partition of table MYEMPP, the rows that are identified by the specified cursor are to be loaded. In each INTO TABLE statement, the PART option specifies the partition number, and the INCURSOR option specifies the cursor. For example, the rows that are identified by cursor C1 are to be loaded into the first partition. The data for each partition is loaded in parallel.

Each cursor is defined in a separate EXEC SQL utility control statement and points to the rows that are returned by executing the specified SELECT statement. These SELECT statement are being executed on a table at a remote server, so the three-part name is used to identify the table. In this example, the column names in table CHICAGO.DSN8810.EMP are the same as the column names in table MYEMPP.

*Figure 14. Example of loading data partitions in parallel using a declared cursor*

EXEC SQL

DECLARE C1 CURSOR FOR SELECT \* FROM CHICAGO.DSN8810.EMP

WHERE EMPNO <= '099999'

ENDEXEC

EXEC SQL

DECLARE C2 CURSOR FOR SELECT \* FROM CHICAGO.DSN8810.EMP

WHERE EMPNO > '099999' AND EMPNO <= '199999'

ENDEXEC

EXEC SQL

DECLARE C3 CURSOR FOR SELECT \* FROM CHICAGO.DSN8810.EMP

WHERE EMPNO > '199999' AND EMPNO <= '299999'

ENDEXEC

EXEC SQL

DECLARE C4 CURSOR FOR SELECT \* FROM CHICAGO.DSN8810.EMP

WHERE EMPNO > '299999' AND EMPNO <= '999999'

ENDEXEC

LOAD DATA

INTO TABLE MYEMPP PART 1 REPLACE INCURSOR(C1)

INTO TABLE MYEMPP PART 2 REPLACE INCURSOR(C2)

INTO TABLE MYEMPP PART 3 REPLACE INCURSOR(C3)

INTO TABLE MYEMPP PART 4 REPLACE INCURSOR(C4)

**Example 17: Loading LOB data from a file**

The LOAD control statement in this example specifies that data from 000130DSN!10.SDSNIVPD(DSN8R130) is to be loaded into the MY\_EMP\_PHOTO\_RESUME table. The characters in positions 1 through 6 are loaded into the EMPNO column, and the characters starting from position 7 are to be loaded into the RESUME column. CLOBF indicates that the characters in position 7 are the name of a file from which a CLOB is to be loaded.

REPLACE indicates that the new data will replace any existing data. Although no logging is to be done, as indicated by the LOG NO option, the table space is not to be set in CHECK-pending state, because NOCOPYPEND is specified.

SORTKEYS 1 indicates that one index key is to be sorted.

*Figure 15. Example of loading LOB data from a file*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* LOAD LOB from file

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//LOADIT EXEC DSNUPROC,UID='LOADIT',TIME=1440,

// UTPROC='',

// SYSTEM='DSN'

//SYSREC DD\*

000130DSN!10.SDSNIVPD(DSN8R130)

//SYSUT1 DD DSN=SYSADM.LOAD.SYSUT1,DISP=(MOD,DELETE,CATLG),

// UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)

//SORTOUT DD DSN=SYSADM.LOAD.SORTOUT,DISP=(MOD,DELETE,CATLG),

// UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)

//SYSIN DD \*

LOAD DATA

REPLACE LOG NO NOCOPYPEND

SORTKEYS 1

INTO TABLE MY\_EMP\_PHOTO\_RESUME

(EMPNO POSITION(1:6) CHAR(6),

RESUME POSITION(7:31) CHAR CLOBF)

**>>>>>>>>>>>>>>>vasuki**

Unload

**UNLOAD**

The online UNLOAD utility unloads data from one or more source objects to one or more BSAM sequential data sets in external formats. The source can be DB2® table spaces or DB2 image copy data sets. The source cannot be a concurrent copy. UNLOAD must be run on the system where the definitions of the table space and the table exists.

UNLOAD is an enhancement of the REORG UNLOAD EXTERNAL function. With UNLOAD, you can unload rows from an entire table space or select specific partitions or tables to unload. You can also select columns by using the field specification list. If a table space is partitioned, you can unload all of the selected partitions into a single data set, or you can unload each partition in parallel into physically distinct data sets.

The output records that the UNLOAD utility writes are compatible as input to the LOAD utility; as a result, you can reload the original table or different tables.

**Output**

UNLOAD generates an unloaded table space or partition.

**Authorization required**

To execute this utility, you must use a privilege set that includes one of the following authorities:

* Ownership of the tables
* SELECT privilege on the tables
* Start of changeDBADM authority for the database. If the object on which the utility operates is in an implicitly created database, DBADM authority on DSNDB04 or the implicitly created database is sufficient.End of change
* SYSADM authority
* SYSCTRL authority (catalog tables only)

If you use RACF® access control with multilevel security and UNLOAD is to process a table space that contains a table that has multilevel security with row-level granularity, you must be identified to RACF and have an accessible valid security label. Each row is unloaded only if your security label dominates the data security label. If your security label does not dominate the data security label, the row is not unloaded, but DB2 does not issue an error message.

**Execution phases of UNLOAD**

The UNLOAD utility operates in these phases:

**Phase**

**Description**

**UTILINIT**

Performs initialization.

**UNLOAD**

Unloads records to sequential data sets. One pass through the input data set is made. If UNLOAD is processing a table space or partition, DB2 takes internal commits. These commits provide commit points at which the utility can be restarted in case operation should halt in this phase.

**UTILTERM**

Performs cleanup.

**>>>>>>>>>>>>>>>vasuki**

Jcl

**>>>>>>>>>>>>>>>vasuki**

**Sample UNLOAD control statements**

Use the sample control statements as models for developing your own UNLOAD control statements.

**Example 1: Unloading all columns of specified rows**

The control statement specifies that all columns of rows that meet the following criteria are to be unloaded from table DSN8810.EMP in table space DSN8D91A.DSN8S71E:

* The value in the WORKDEPT column is D11.
* The value in the SALARY column is greater than 25 000.

*Figure 1. Example of unloading all columns of specified rows*

//STEP1 EXEC DSNUPROC,UID='SMPLUNLD',UTPROC='',SYSTEM='DSN'

//SYSREC DD DSN=USERID.SMPLUNLD.SYSREC,

// DISP=(NEW,CATLG,CATLG),

// UNIT=SYSDA,SPACE=(TRK,(2,1))

//SYSPUNCH DD DSN=USERID.SMPLUNLD.SYSPUNCH,

// DISP=(NEW,CATLG,CATLG),

// UNIT=SYSDA,SPACE=(TRK,(1,1))

//SYSPRINT DD SYSOUT=\*

//SYSIN DD \*

UNLOAD TABLESPACE DSN8D91A.DSN8S81E

FROM TABLE DSN8910.EMP

WHEN (WORKDEPT = 'D11' AND SALARY > 25000)

**Example 2: Unloading specific columns by using a field specification list**

The following control statement specifies that columns EMPNO, LASTNAME, and SALARY are to be unloaded, in that order, for all rows that meet the specified conditions. These conditions are specified in the WHEN clause and are the same as those conditions in example 1. The SALARY column is to be unloaded as type DECIMAL EXTERNAL. The NOPAD option indicates that variable-length fields are to be unloaded without any padding.

UNLOAD TABLESPACE DSN8D91A.DSN8S81E NOPAD

FROM TABLE DSN8910.EMP

(EMPNO, LASTNAME, SALARY DECIMAL EXTERNAL)

WHEN (WORKDEPT = 'D11' AND SALARY > 25000)

The output from this example might look similar to the following output:

000060@@STERN# 32250.00

000150@@ADAMSON# 25280.00

000200@@BROWN# 27740.00

000220@@LUTZ# 29840.00

200220@@JOHN# 29840.00

In this output:

* '@@' before the last name represents the 2-byte binary field that contains the length of the VARCHAR field LASTNAME (for example, X'0005' for STERN).
* '#' represents the NULL indicator byte for the nullable SALARY field.
* Because the SALARY column is declared as DECIMAL (9,2) on the table, the default output length of the SALARY field is 11 (9 digits + sign + decimal point), not including the NULL indicator byte.
* LASTNAME is unloaded as a variable-length field because the NOPAD option is specified.

**Example 3: Unloading data from an image copy**

The FROMCOPY option in the following control statement specifies that data is to be unloaded from a single image copy data set, JUKWU111.FCOPY1.STEP1.FCOPY1.

PUNCHDDN SYSPUNCH specifies that the UNLOAD utility is to generate LOAD utility control statements and write them to the data set that is defined by the SYSPUNCH DD statement; SYSPUNCH is the default. UNLDDN SYSREC specifies that the data is to be unloaded to the data set that is defined by the SYSREC DD statement; SYSREC is the default.

UNLOAD TABLESPACE DBKW1101.TPKW1101

FROMCOPY JUKWU111.FCOPY1.STEP1.FCOPY1

PUNCHDDN SYSPUNCH UNLDDN SYSREC

**Example 4: Unloading a sample of rows and specifying a header.**

The following control statement specifies that a sample of rows is to be unloaded from table ADMF001.TBKW1605. Unloading a sample of rows is useful for building a test system. The SAMPLE option indicates that 75% of the rows are to be sampled. The HEADER option indicates that the string ’sample’ is to be used as the header field in the output file. The PUNCHDDN option indicates that UNLOAD is to generate LOAD utility control statements and write them to the SYSPUNCH data set, which is the default. UNLOAD specifies the header field as a criterion in the WHEN clause of these LOAD statements.

UNLOAD TABLESPACE DBKW1603.TPKW1603

PUNCHDDN SYSPUNCH UNLDDN SYSREC

FROM TABLE ADMF001.TBKW1605

HEADER CONST 'sample'

SAMPLE 75

**Example 5: Unloading data from two tables in a segmented table space**

The following control statement specifies that data from table ADMF001.TBKW1504 and table ADMF001.TBKW1505 is to be unloaded from the segmented table space DBKW1502.TSKW1502. The PUNCHDDN option indicates that UNLOAD is to generate LOAD utility control statements and write them to the SYSPUNCH data set, which is the default. The UNLDDN option specifies that the data is to be unloaded to the data set that is defined by the SYSREC DD statement, which is also the default.

UNLOAD TABLESPACE DBKW1502.TSKW1502

PUNCHDDN SYSPUNCH UNLDDN SYSREC

FROM TABLE ADMF001.TBKW1504

FROM TABLE ADMF001.TBKW1505

**Example 6: Unloading data in parallel from a partitioned table space**

The UNLOAD control statement specifies that data from table TCRT.TTBL is to be unloaded to data sets that are defined by the UNLDDS template. These data sets are to be dynamically allocated and named according to the naming convention that is defined by the DSN option of the TEMPLATE utility control statement. This naming convention indicates that a data set is to be allocated for each table space partition.

Assume that table space TDB1.TSP1, which contains table TCRT.TTBL, has three partitions. Because the table space is partitioned and each partition is associated with an output data set that is defined by the UNLDDS template, the UNLOAD job runs in parallel in a multi-processor environment. The number of parallel tasks are determined by the number of available processors.

*Figure 2. Example of unloading data in parallel from a partitioned table space*

//STEP1 EXEC DSNUPROC,UID='SMPLUNLD',UTPROC='',SYSTEM='DSN'

//SYSPUNCH DD DSN=USERID.SMPLUNLD.SYSPUNCH,

// DISP=(NEW,CATLG,CATLG),

// UNIT=SYSDA,SPACE=(TRK,(1,1))

//SYSPRINT DD SYSOUT=\*

//SYSIN DD \*

TEMPLATE UNLDDS DSN &USERID..SMPLUNLD.&TS..P&PART.

UNIT SYSDA DISP (NEW,CATLG,CATLG) SPACE (2,1) CYL

UNLOAD TABLESPACE TDB1.TSP1

UNLDDN UNLDDS

FROM TABLE TCRT.TTBL

Assume that the user ID is USERID. This UNLOAD job creates the following three data sets to store the unloaded data:

* USERID.SMPLUNLD.TSP1.P00001 ... contains rows from partition 1.
* USERID.SMPLUNLD.TSP1.P00002 ... contains rows from partition 2.
* USERID.SMPLUNLD.TSP1.P00003 ... contains rows from partition 3.

**Example 7: Using a LISTDEF utility statement to specify partitions to unload**

The UNLOAD control statement specifies that data that is included in the UNLDLIST list is to be unloaded. UNLDLIST is defined in the LISTDEF utility control statement and contains partitions one and three of table space TDB1.TSP1. The LIST option of the UNLOAD statement specifies that the UNLOAD utility is to use this list.

The data is to be unloaded to data sets that are defined by the UNLDDS template.

*Figure 3. Example of using a LISTDEF utility statement to specify partitions to unload*

//SAMPJOB JOB ...

//STEP1 EXEC DSNUPROC,UID='SMPLUNLD',UTPROC='',SYSTEM='DSN'

//SYSPUNCH DD DSN=USERID.SMPLUNLD.SYSPUNCH,

// DISP=(NEW,CATLG,CATLG),

// UNIT=SYSDA,SPACE=(TRK,(1,1))

//SYSPRINT DD SYSOUT=\*

//SYSIN DD \*

LISTDEF UNLDLIST

INCLUDE TABLESPACE TDB1.TSP1 PARTLEVEL(1)

INCLUDE TABLESPACE TDB1.TSP1 PARTLEVEL(3)

TEMPLATE UNLDDS DSN &USERID..SMPLUNLD.&TS..P&PART.

UNIT SYSDA DISP (NEW,CATLG,CATLG) SPACE (2,1) CYL

UNLOAD LIST UNLDLIST -- LIST name

UNLDDN UNLDDS -- TEMPLATE name

Assume that the user ID is USERID. This UNLOAD job creates the following two data sets to store the unloaded data:

* USERID.SMPLUNLD.TSP1.P00001 ... contains rows from partition 1.
* USERID.SMPLUNLD.TSP1.P00003 ... contains rows from partition 3.

**Example 8: Unloading multiple table spaces by using LISTDEF**

The UNLOAD control statement specifies that data from multiple table spaces is to be unloaded. These table spaces are specified in the LISTDEF utility control statement. Assume that the database TDB1 contains two table spaces that can be expressed by the pattern-matching string 'TSP\*', (for example, TSP1 and TSP2). These table spaces are both included in the list named UNLDLIST, which is defined in the LISTDEF statement. The LIST option of the UNLOAD statement specifies that the UNLOAD utility is to use this list.

The UNLDDN option specifies that the data is to be unloaded to data sets that are defined by the UNLDDS template. The PUNCHDDN option specifies that UNLOAD is to generate LOAD utility control statements and write them to the data sets that are defined by the PUNCHDS template.

*Figure 4. Example of unloading multiple table spaces*

//SAMPJOB JOB ...

//STEP1 EXEC DSNUPROC,UID='SMPLUNLD',UTPROC='',SYSTEM='DSN'

//SYSPRINT DD SYSOUT=\*

//SYSIN DD \*

LISTDEF UNLDLIST

INCLUDE TABLESPACE TDB1.TSP\*

TEMPLATE UNLDDS DSN &USERID..SMPLUNLD.&TS.

UNIT SYSDA DISP (NEW,CATLG,CATLG) SPACE (2,1) CYL

TEMPLATE PUNCHDS DSN &USERID..SMPLPUNC.&TS.

UNIT SYSDA DISP (NEW,CATLG,CATLG) SPACE (1,1) CYL

UNLOAD LIST UNLDLIST

PUNCHDDN PUNCHDS -- TEMPLATE name

UNLDDN UNLDDS -- TEMPLATE name

Assume that the user ID is USERID. This UNLOAD job creates the following two data sets to store the unloaded data:

* USERID.SMPLUNLD.TSP1 ... contains rows from table space TDB1.TSP1.
* USERID.SMPLUNLD.TSP2 ... contains rows from table space TDB1.TSP2.

**Example 9: Unloading data into a delimited file.**

The control statement specifies that data from the specified columns (RECID, CHAR7SBCS, CHAR7BIT, VCHAR20, VCHAR20SBCS, VCHAR20BIT) in table TBQB0501 is to be unloaded into a delimited file. This output format is indicated by the DELIMITED option. The POSITION(\*) option indicates that each field in the output file is to start at the first byte after the last position of the previous field.

The column delimiter is specified by the COLDEL option as a semicolon (;), the character string delimiter is specified by the CHARDEL option as a pound sign (#), and the decimal point character is specified by the DECPT option as an exclamation point (!).

PUNCHDDN SYSPUNCH specifies that UNLOAD is to generate LOAD utility control statements and store them in the SYSPUNCH data set, which is the default. UNLDDN SYSREC indicates that the data is to be unloaded to the SYSREC data set, which is the default.

The EBCDIC option indicates that all output character data is to be in EBCDIC.

*Figure 5. Example of unloading data into a delimited file.*

//\*

//STEP3 EXEC DSNUPROC,UID='JUQBU105.UNLD1',

// UTPROC='',

// SYSTEM='SSTR'

//UTPRINT DD SYSOUT=\*

//SYSREC DD DSN=JUQBU105.UNLD1.STEP3.TBQB0501,DISP=(MOD,DELETE,CATLG),

// UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)

//SYSPUNCH DD DSN=JUQBU105.UNLD1.STEP3.SYSPUNCH

// DISP=(MOD,CATLG,CATLG)

// UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)

//SYSIN DD\*

UNLOAD TABLESPACE DBQB0501.TSQB0501

DELIMITED CHARDEL '#' COLDEL ';' DECPT '!'

PUNCHDDN SYSPUNCH

UNLDDN SYSREC EBCDIC

FROM TABLE ADMF001.TBQB0501

(RECID POSITION(\*) CHAR,

CHAR7SBCS POSITION(\*) CHAR,

CHAR7SBIT POSITION(\*) CHAR(7),

VCHAR20 POSITION(\*) VARCHAR,

VCHAR20SBCS POSITION(\*) VARCHAR,

VCHAR20BIT POSITION(\*) VARCHAR)

/\*

**Example 10: Converting character data**

For this example, assume that table DSN8810.DEMO\_UNICODE contains character data in Unicode. The UNLOAD control statement specifies that the utility is to unload the data in this table as EBCDIC data.

*Figure 6. Example of unloading Unicode table data into EBCDIC*

UNLOAD

EBCDIC

TABLSPACE DSN8D81E.DSN8S81U

FROM TABLE DSN8810.DEMO\_UNICODE

**Example 11: Unloading LOB data to a file**

The UNLOAD control statement specifies that the utility is to unload data from table DSN8910.EMP\_PHOTO\_RESUME into the data set that is identified by the SYSREC DD statement. Data in the EMPNO field is six bytes of character data, as indicated by the CHAR(6) option, and is unloaded directly into the SYSREC data set. Data in the RESUME column is CLOB data as indicated by the CLOBF option. This CLOB data is to be unloaded to the files identified by the LOBFRV template, which is defined in the preceding TEMPLATE statement. If these files do not already exist, DB2® creates them. The names of these files are stored in the SYSREC data set. The length of the file name to be stored in this data set can be up to 255 bytes as specified by the VARCHAR option.

*Figure 7. Example of unloading LOB data into a file*

TEMPLATE LOBFRV DSN 'UNLDTEST.&DB..&TS..RESUME'

DSNTYPE(PDS) UNIT(SYSDA)

UNLOAD DATA

FROM TABLE DSN8910.EMP\_PHOTO\_RESUME

(EMPNO CHAR(6),

RESUME VARCHAR(255) CLOBF LOBFRV)

SHRLEVEL CHANGE

**Example 12: Unloading data from clone tables**

Start of changeThe UNLOAD control statement specifies that the utility is to unload data from only clone tables in the specified table spaces. The PUNCHDDN option specifies that the SYSPUNCH data set is to receive the LOAD utility control statements that the UNLOAD utility generates. End of change

UNLOAD TABLESPACE DBKQRE01.TPKQRE01

FROM TABLE ADMF001.TBKQRE01\_CLONE

PUNCHDDN SYSPUNCH UNLDDN SYSREC

CLONE