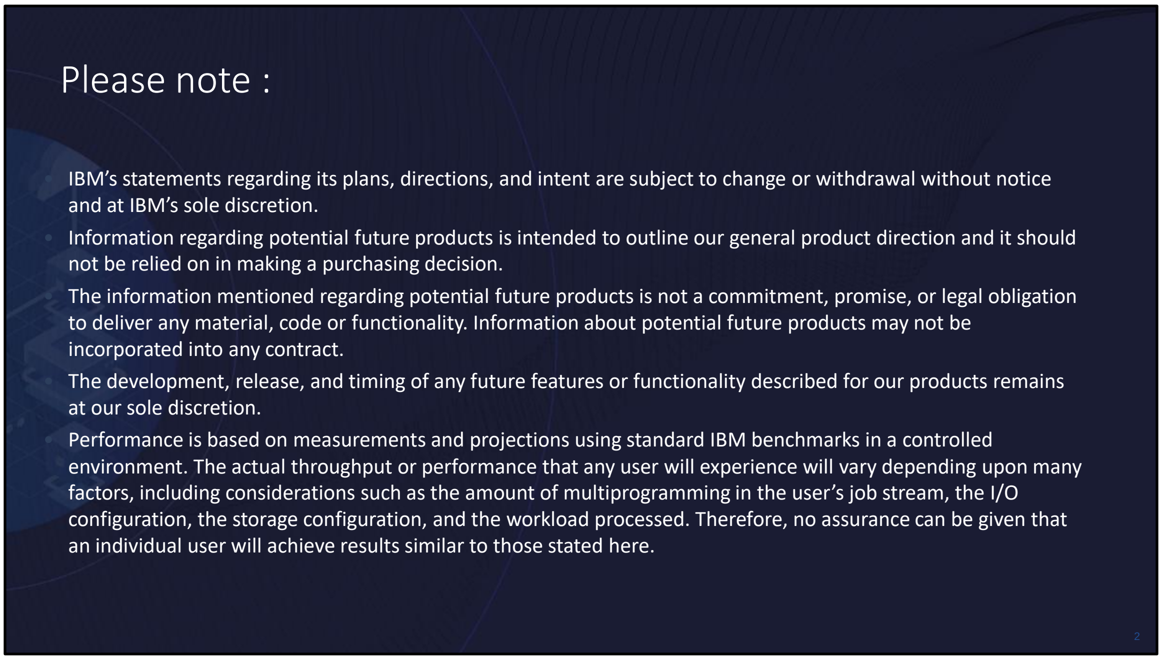
IDUG VIRTUAL
2021 NA Db2 Tech Conference

IDUG

External Tables a deep dive

Mike Springgay, IBM

Db2



Please note :

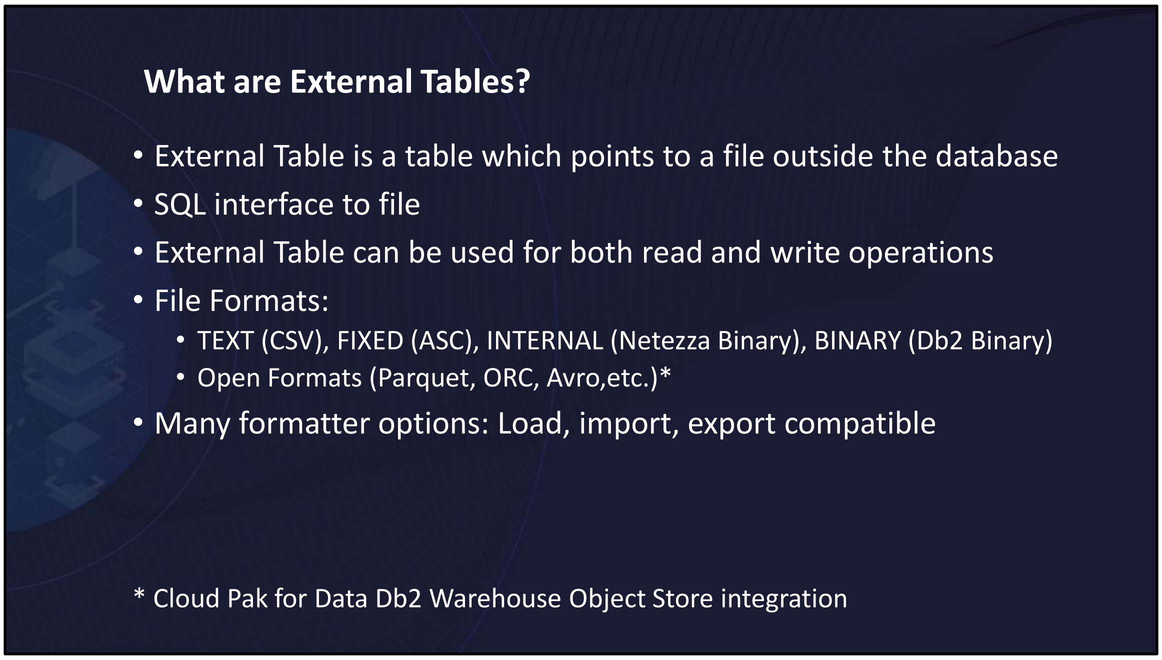
- IBM’s statements regarding its plans, directions, and intent are subject to change or withdrawal without notice and at IBM’s sole discretion.
- Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.
- The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.
- The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.
- Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user’s job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

2

Notices and disclaimers

Agenda

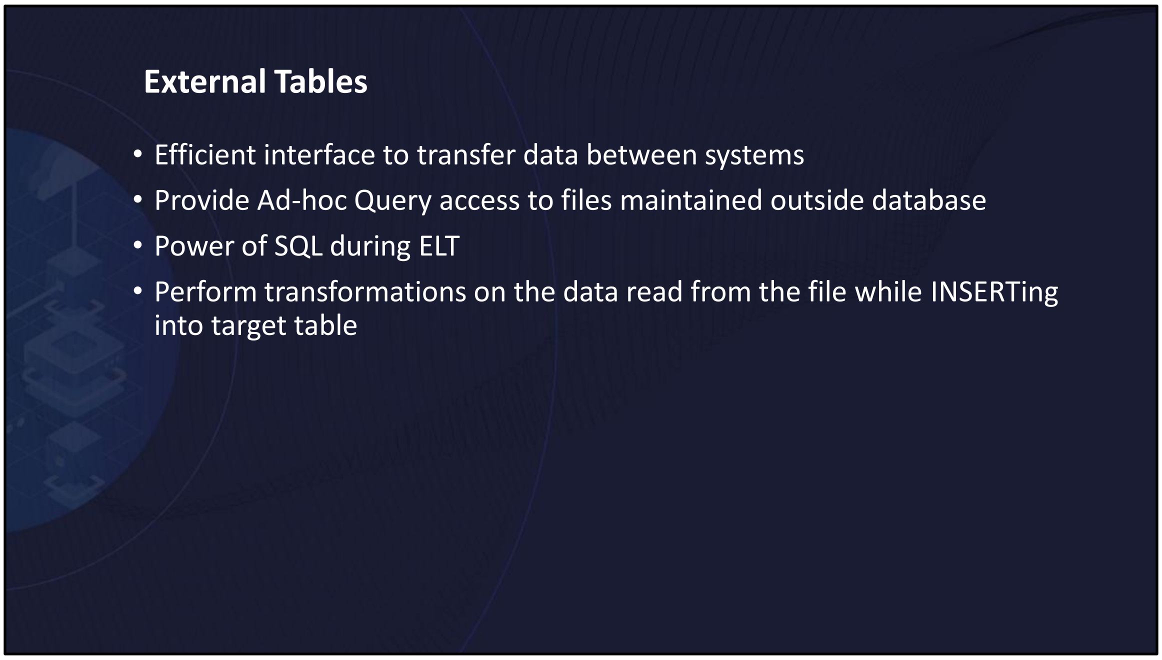
- What are External Tables
- Syntax and Usage
- Diagnostics, Security and Configuration
- Best Practices
- Summary



What are External Tables?

- External Table is a table which points to a file outside the database
- SQL interface to file
- External Table can be used for both read and write operations
- File Formats:
  - TEXT (CSV), FIXED (ASC), INTERNAL (Netezza Binary), BINARY (Db2 Binary)
  - Open Formats (Parquet, ORC, Avro,etc.)*
- Many formatter options: Load, import, export compatible

* Cloud Pak for Data Db2 Warehouse Object Store integration

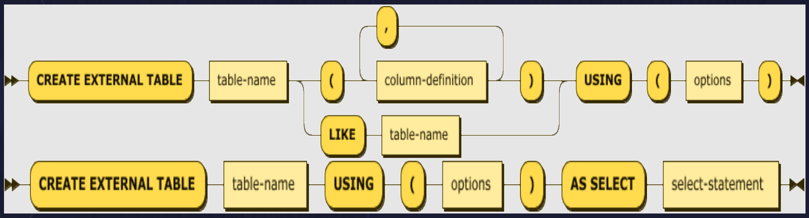


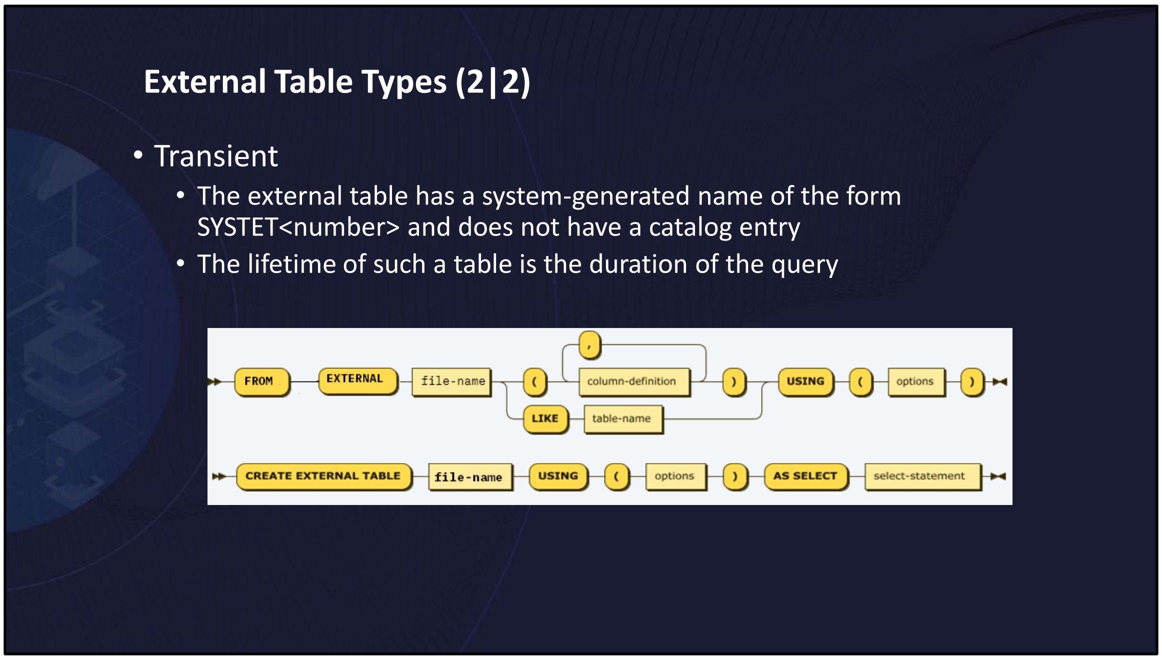
External Tables

- Efficient interface to transfer data between systems
- Provide Ad-hoc Query access to files maintained outside database
- Power of SQL during ELT
- Perform transformations on the data read from the file while INSERTing into target table

External Table Types (1|2)

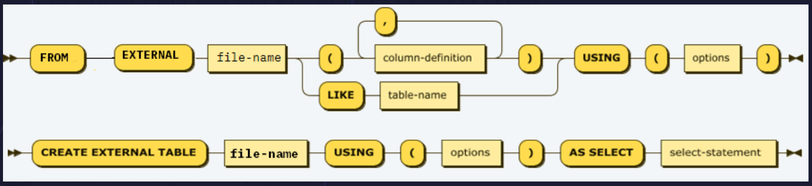
- Named
  - The external table has a name and catalog entry like a normal table

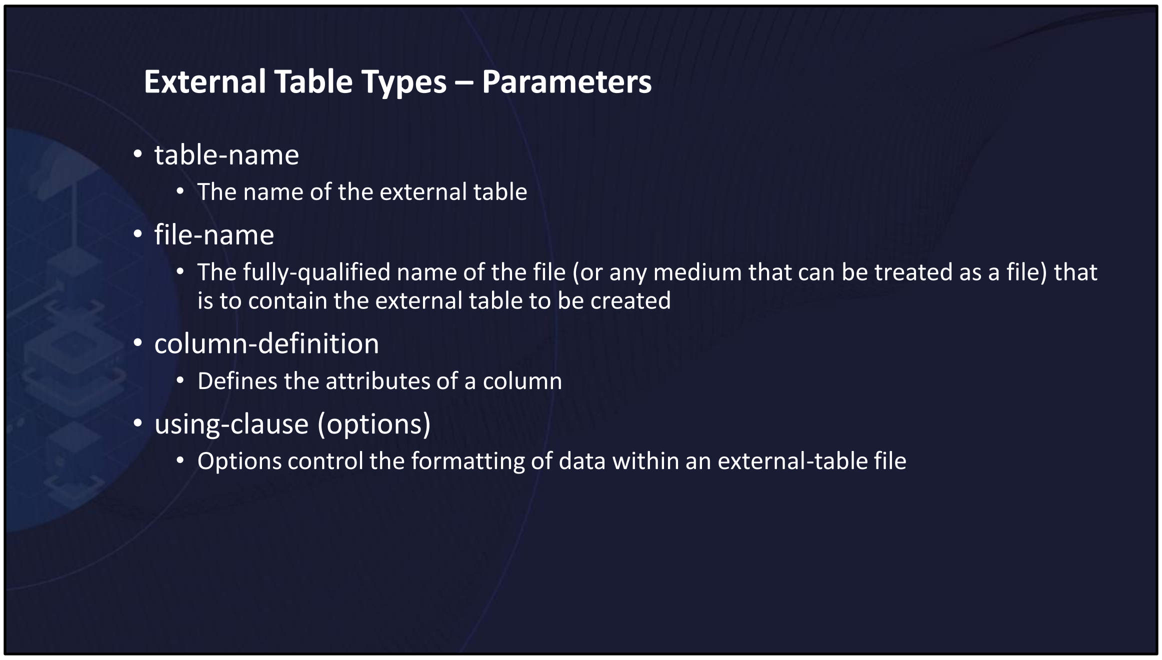




External Table Types (2|2)

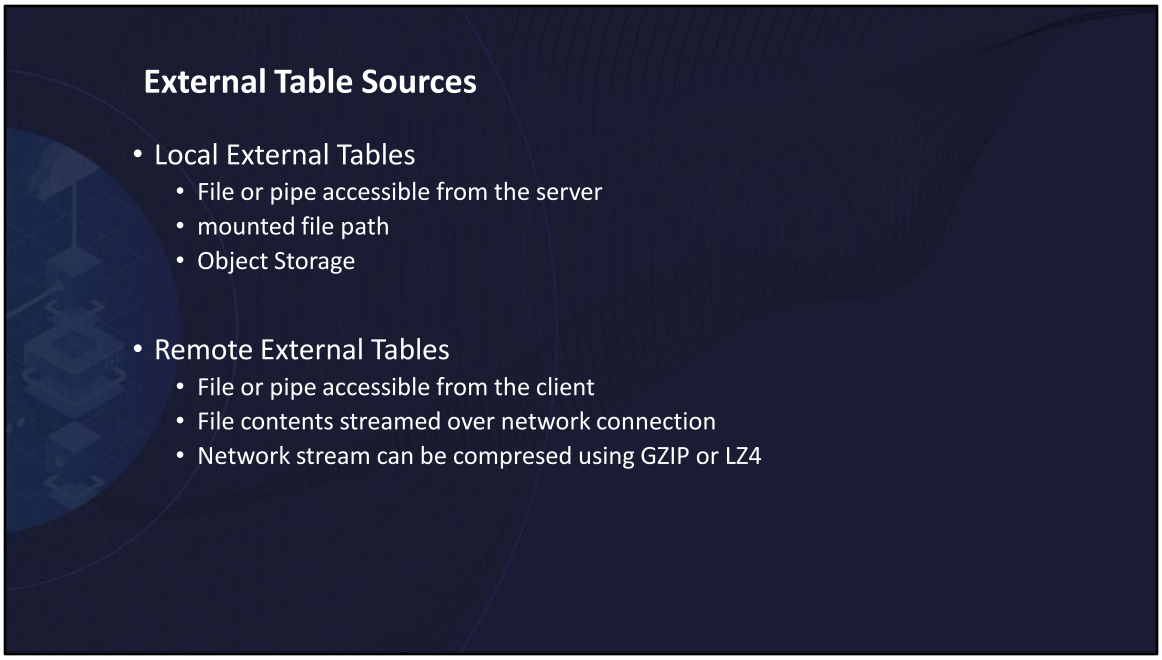
- Transient
  - The external table has a system-generated name of the form SYSTET<number> and does not have a catalog entry
  - The lifetime of such a table is the duration of the query





External Table Types – Parameters

- table-name
  - The name of the external table
- file-name
  - The fully-qualified name of the file (or any medium that can be treated as a file) that is to contain the external table to be created
- column-definition
  - Defines the attributes of a column
- using-clause (options)
  - Options control the formatting of data within an external-table file


External Table Sources

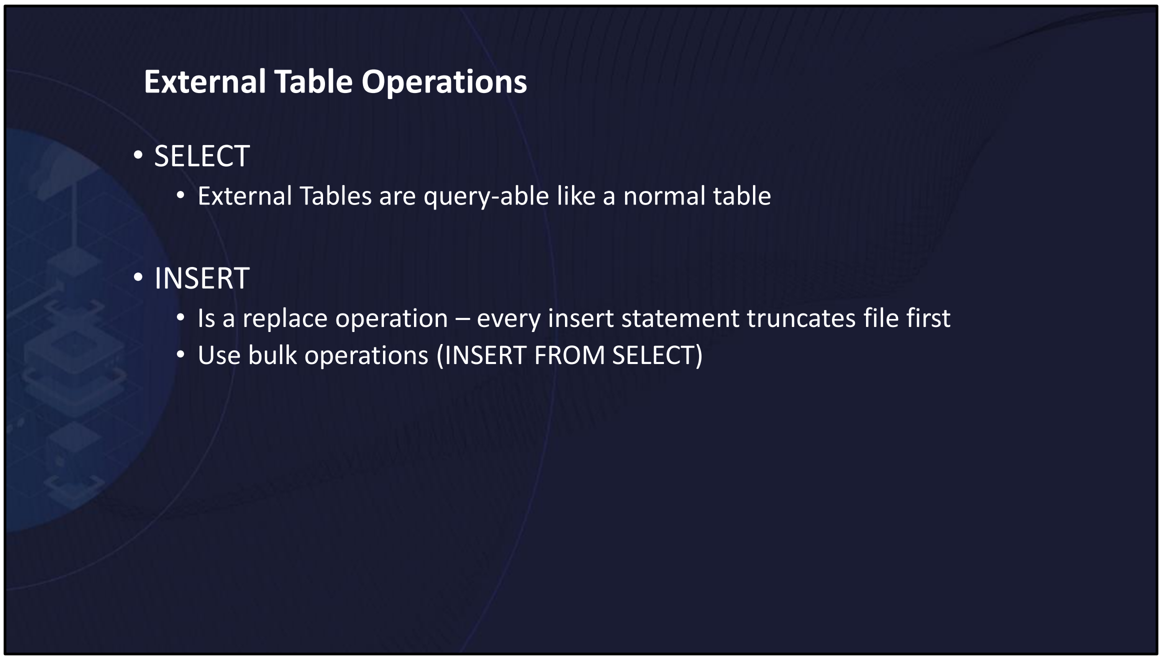
- Local External Tables
  - File or pipe accessible from the server
  - mounted file path
  - Object Storage
- Remote External Tables
  - File or pipe accessible from the client
  - File contents streamed over network connection
  - Network stream can be compresed using GZIP or LZ4

Local external tables refer to accessing files accessible directly form the Db2 engine.

Remote as in the file is on the client machine is also supported.
The REMOTESOURCE option specifies the file is remote and
DATAOBJECT or transient 'file-name' is then relative to the client machine.

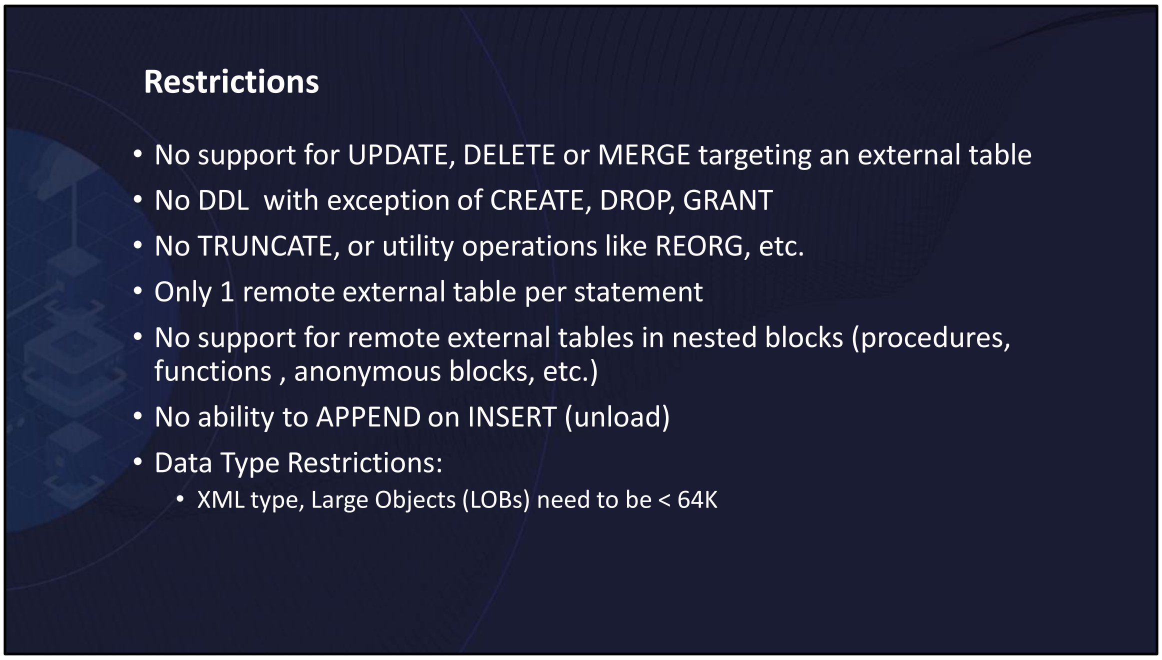
REMOTESOURCE YES, JDBC, ODBC are all equivalent
REMOTESOURCE LZ4 or GZIP indicates the network communication should be compressed.

Note that if file is already compressed then network compression should not be applied
as only one pass of uncompressing the data is applied



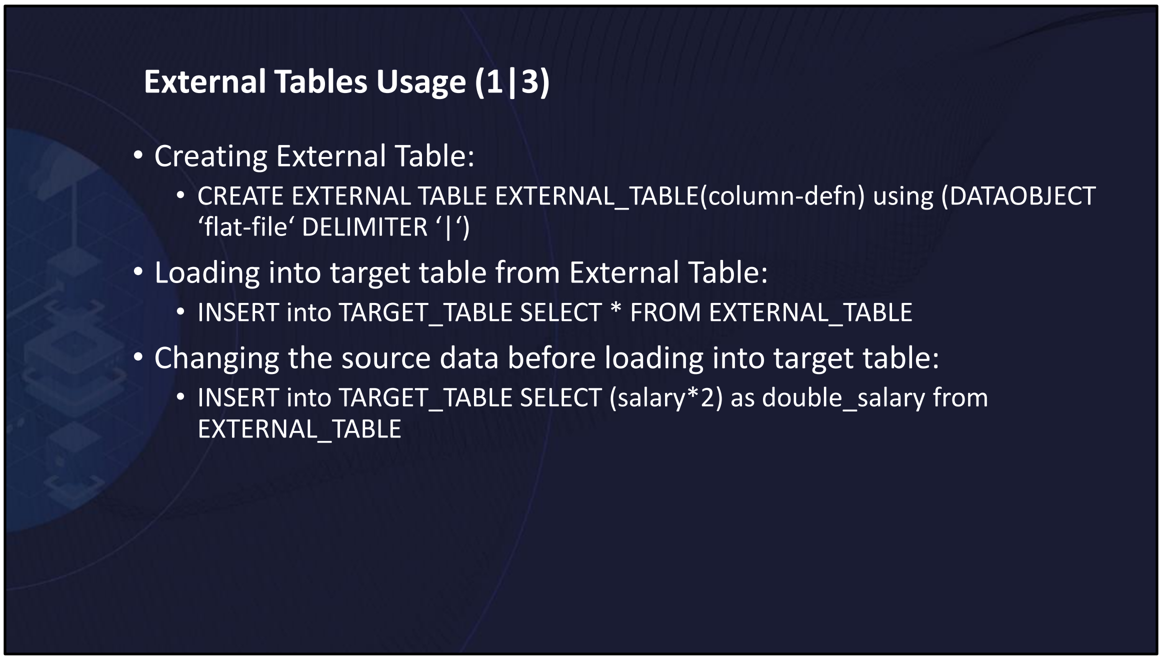
External Table Operations

- SELECT
  - External Tables are query-able like a normal table
- INSERT
  - Is a replace operation – every insert statement truncates file first
  - Use bulk operations (INSERT FROM SELECT)



Restrictions

- No support for UPDATE, DELETE or MERGE targeting an external table
- No DDL with exception of CREATE, DROP, GRANT
- No TRUNCATE, or utility operations like REORG, etc.
- Only 1 remote external table per statement
- No support for remote external tables in nested blocks (procedures, functions , anonymous blocks, etc.)
- No ability to APPEND on INSERT (unload)
- Data Type Restrictions:
  - XML type, Large Objects (LOBs) need to be < 64K



External Tables Usage (1|3)

- Creating External Table:
  - CREATE EXTERNAL TABLE EXTERNAL_TABLE(column-defn) using (DATAOBJECT ‘flat-file‘ DELIMITER ‘|‘)
- Loading into target table from External Table:
  - INSERT into TARGET_TABLE SELECT * FROM EXTERNAL_TABLE
- Changing the source data before loading into target table:
  - INSERT into TARGET_TABLE SELECT (salary*2) as double_salary from EXTERNAL_TABLE

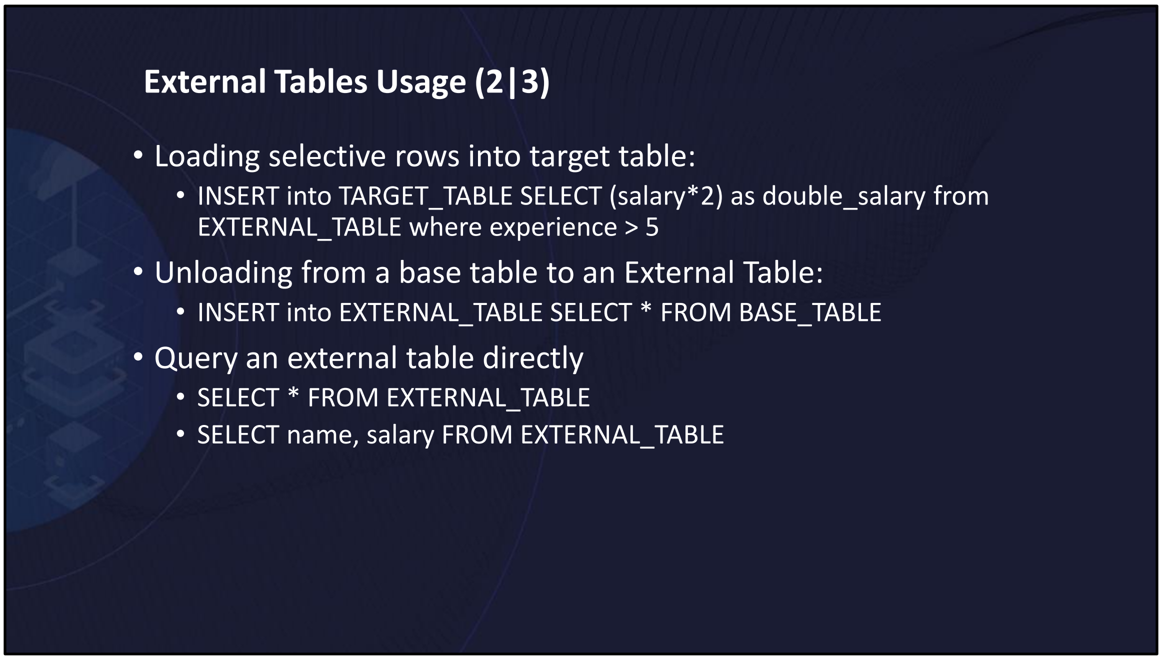
As discussed there are named versions of external tables and transient.

Here we show creation of a named external tabled called EXTERNAL_TABLE.

You can then use a catalog external table just like you would a regular or temporary table.

In an INSERT FROM SELECT to load a permeant table with the new rows

You can change values as they are processed. Here the salary information is first doubled before storing



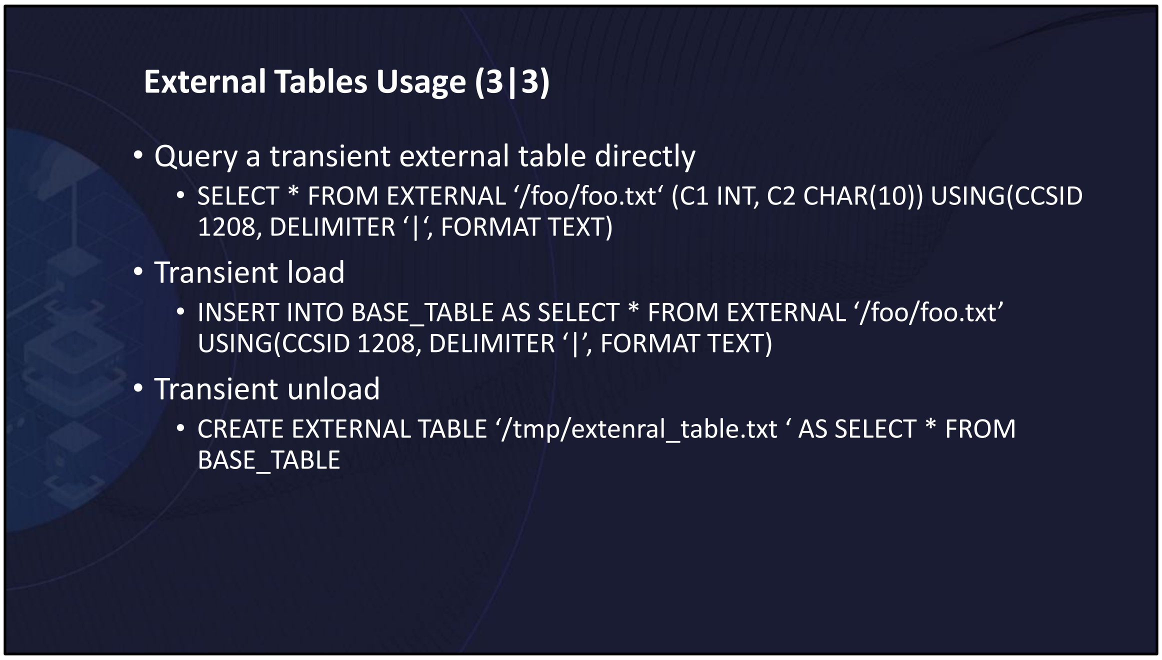
External Tables Usage (2|3)

- Loading selective rows into target table:
  - INSERT into TARGET_TABLE SELECT (salary*2) as double_salary from EXTERNAL_TABLE where experience > 5
- Unloading from a base table to an External Table:
  - INSERT into EXTERNAL_TABLE SELECT * FROM BASE_TABLE
- Query an external table directly
  - SELECT * FROM EXTERNAL_TABLE
  - SELECT name, salary FROM EXTERNAL_TABLE

You can apply predicates to load only those rows of interest and apply additional functions to the qualifying rows columns.

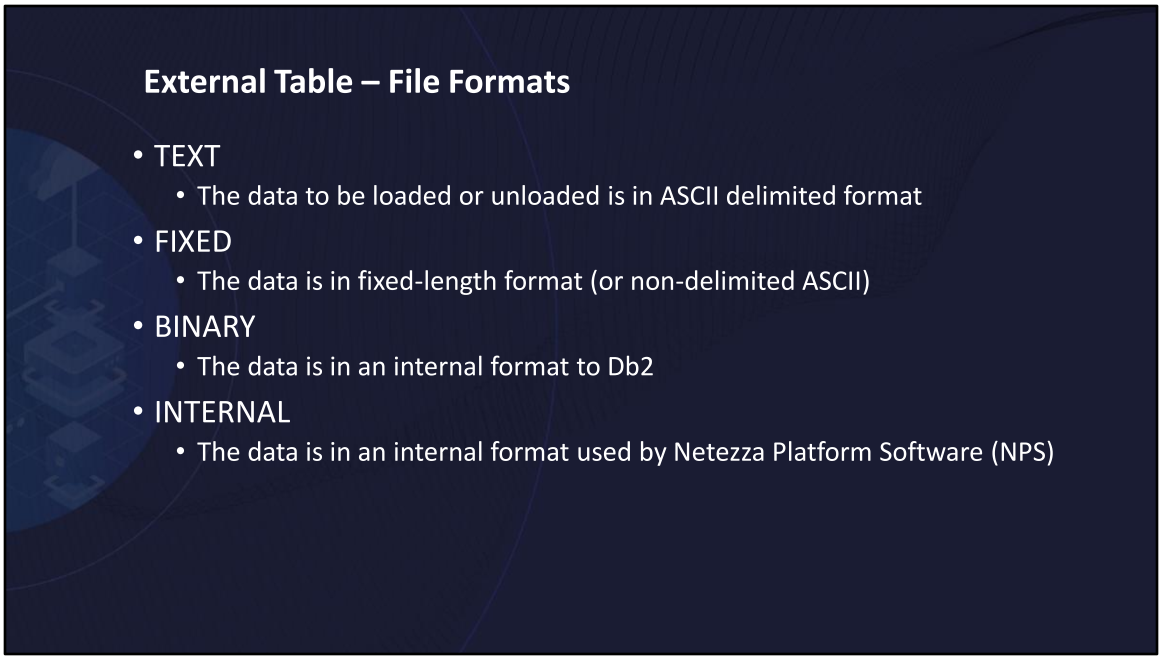
Can INSERT into an external table to unload a table.   INSERTS are distinct operations.
When a External Table is opened for write if it exists it is first truncated.
So you must use INSERT FROM SELECT not singleton inserts to populate a single file.

You can also just query an external table like any other table.



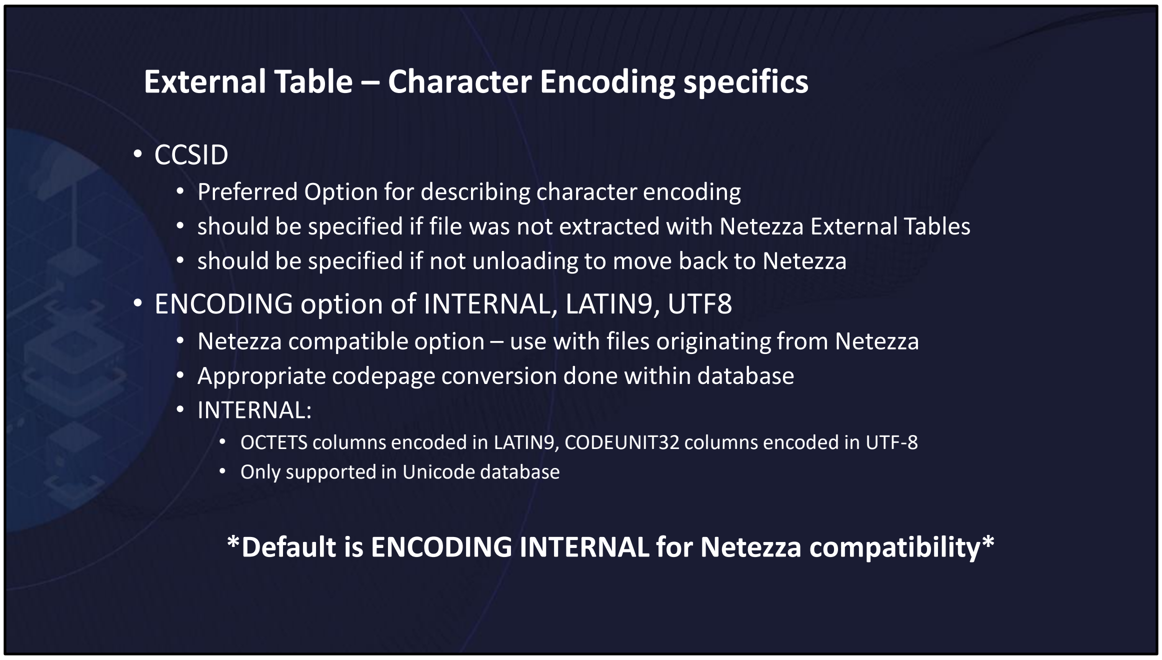
External Tables Usage (3|3)

- Query a transient external table directly
  - SELECT * FROM EXTERNAL ‘/foo/foo.txt‘ (C1 INT, C2 CHAR(10)) USING(CCSID 1208, DELIMITER ‘|‘, FORMAT TEXT)
- Transient load
  - INSERT INTO BASE_TABLE AS SELECT * FROM EXTERNAL ‘/foo/foo.txt’ USING(CCSID 1208, DELIMITER ‘|’, FORMAT TEXT)
- Transient unload
  - CREATE EXTERNAL TABLE ‘/tmp/extenral_table.txt ‘ AS SELECT * FROM BASE_TABLE



External Table – File Formats

- TEXT
  - The data to be loaded or unloaded is in ASCII delimited format
- FIXED
  - The data is in fixed-length format (or non-delimited ASCII)
- BINARY
  - The data is in an internal format to Db2
- INTERNAL
  - The data is in an internal format used by Netezza Platform Software (NPS)



External Table – Character Encoding specifics

- CCSID
  - Preferred Option for describing character encoding
  - should be specified if file was not extracted with Netezza External Tables
  - should be specified if not unloading to move back to Netezza
- ENCODING option of INTERNAL, LATIN9, UTF8
  - Netezza compatible option – use with files originating from Netezza
  - Appropriate codepage conversion done within database
  - INTERNAL:
    - OCTETS columns encoded in LATIN9, CODEUNIT32 columns encoded in UTF-8
    - Only supported in Unicode database

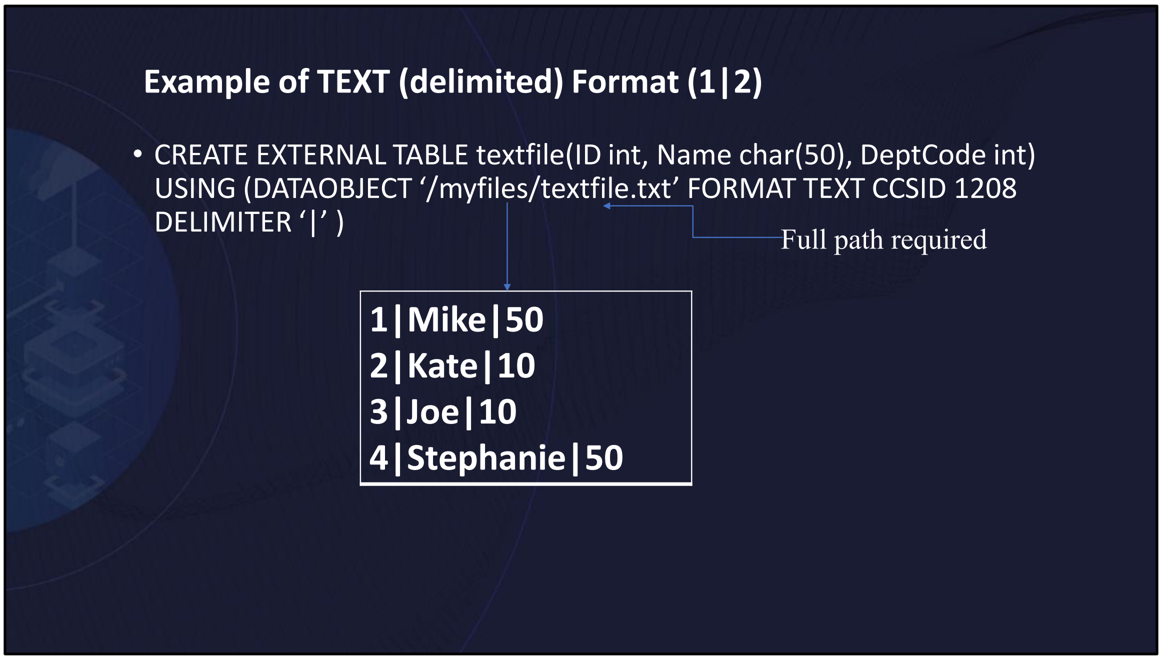
Default is ENCODING INTERNAL for Netezza compatibility

Default is ENCODING INTERNAL just like on Netezza. This implies files can be a mix of Latain-9 (codepage 923) or UTF-8 depending on column definition.
When processing the file any char / varchar tagged as CODEUNITS32 is assumed to be in UTF-8 instead of Latin-9.

Initially because of the Netezza behavior ENCODING LATIN-9 or ENCODING UTF-8 would also strictly adhere to the codeunits tagging in database.
However since Db2 is a codepage aware database this restriction has been lifted and both encoding can now be properly ingested into either
codeunits or octets applying the appropriate constraints as required

CCSID is an extension added to conform with Db2 means providing codepage information and to support codepage transformations within the database itself - receiver makes right!
Note that when specifying CCSID it will change some defaults around date,time,timestamp formatting to align closer to Db2 expectations.

CCSID option should always be used on unload if not targeting a Netezza system. And in general on load unless the source file came from Netezza.

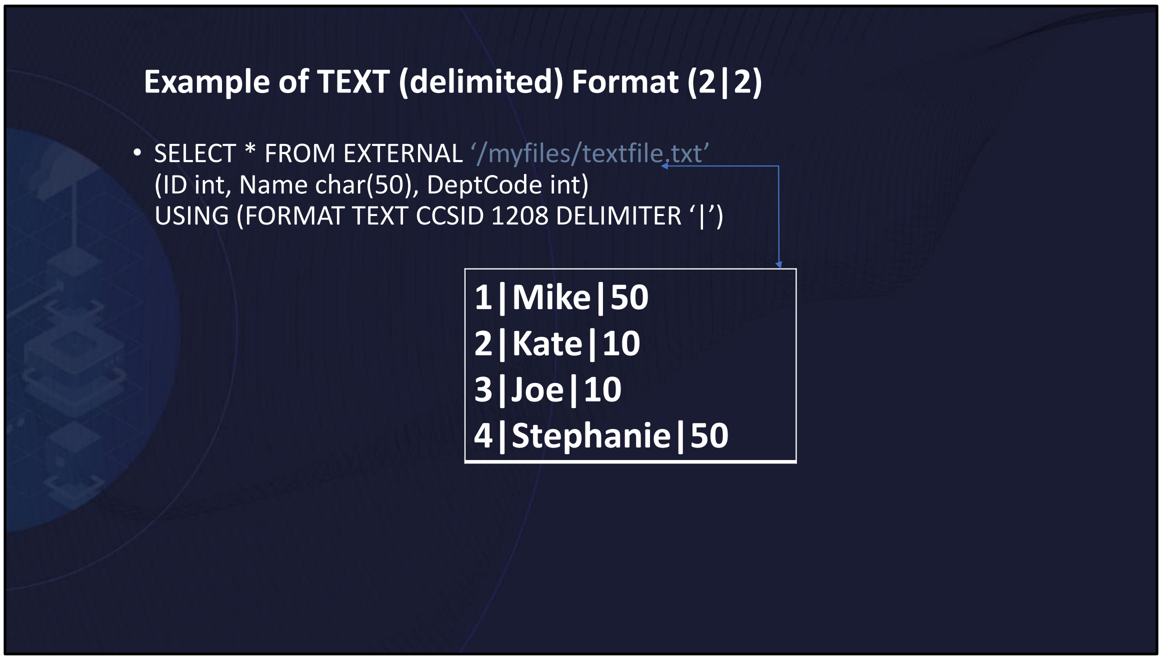


Example of TEXT (delimited) Format (1|2)

- CREATE EXTERNAL TABLE textfile(ID int, Name char(50), DeptCode int) USING (DATAOBJECT ‘/myfiles/textfile.txt’ FORMAT TEXT CCSID 1208 DELIMITER ‘|’)

Full path required

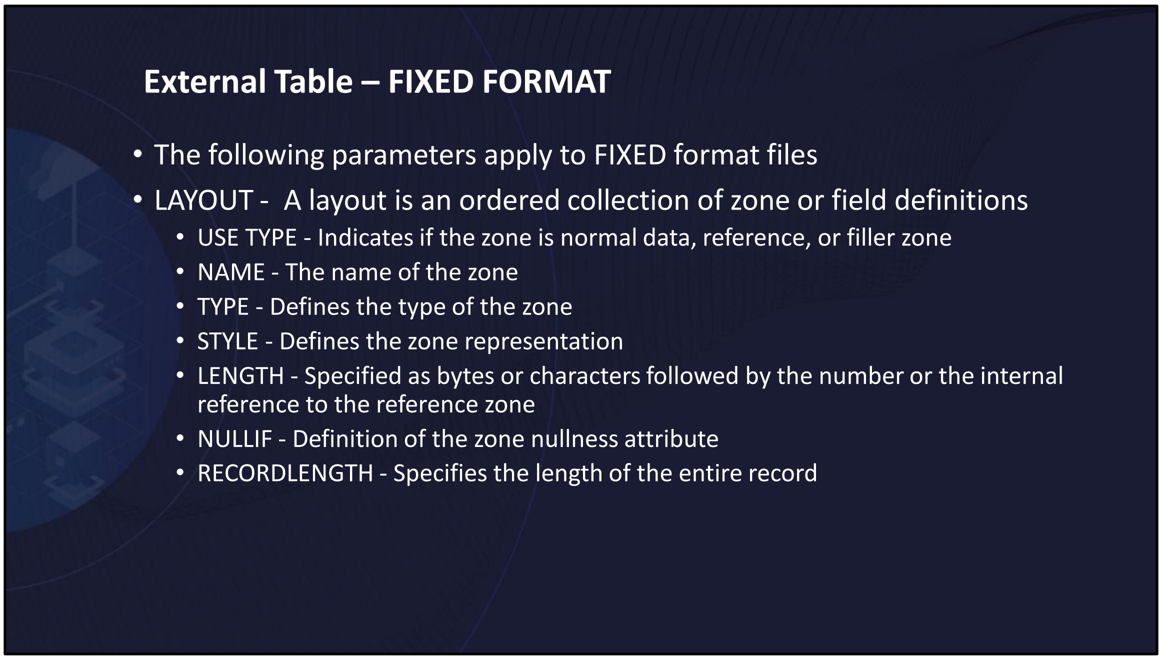
```
1|Mike|50
2|Kate|10
3|Joe|10
4|Stephanie|50
```



Example of TEXT (delimited) Format (2|2)

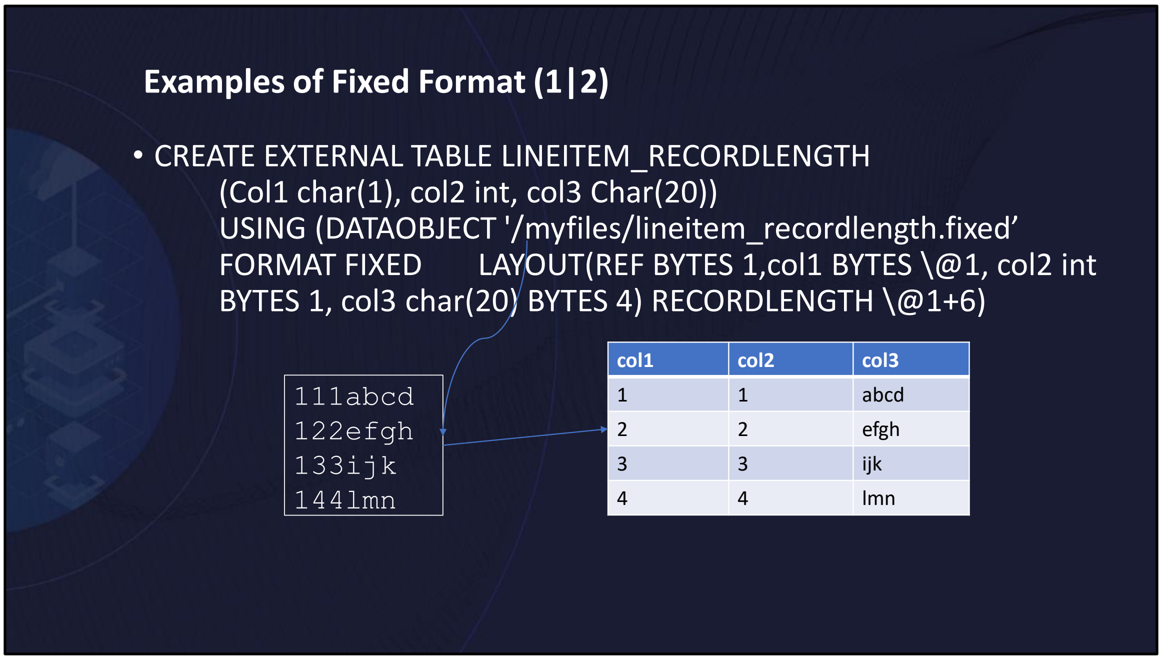
- SELECT * FROM EXTERNAL '/myfiles/textfile.txt'
  (ID int, Name char(50), DeptCode int)
  USING (FORMAT TEXT CCSID 1208 DELIMITER '|')

```
1|Mike|50
2|Kate|10
3|Joe|10
4|Stephanie|50
```



External Table – FIXED FORMAT

- The following parameters apply to FIXED format files
- LAYOUT - A layout is an ordered collection of zone or field definitions
  - USE TYPE - Indicates if the zone is normal data, reference, or filler zone
  - NAME - The name of the zone
  - TYPE - Defines the type of the zone
  - STYLE - Defines the zone representation
  - LENGTH - Specified as bytes or characters followed by the number or the internal reference to the reference zone
  - NULLIF - Definition of the zone nullness attribute
  - RECORDLENGTH - Specifies the length of the entire record

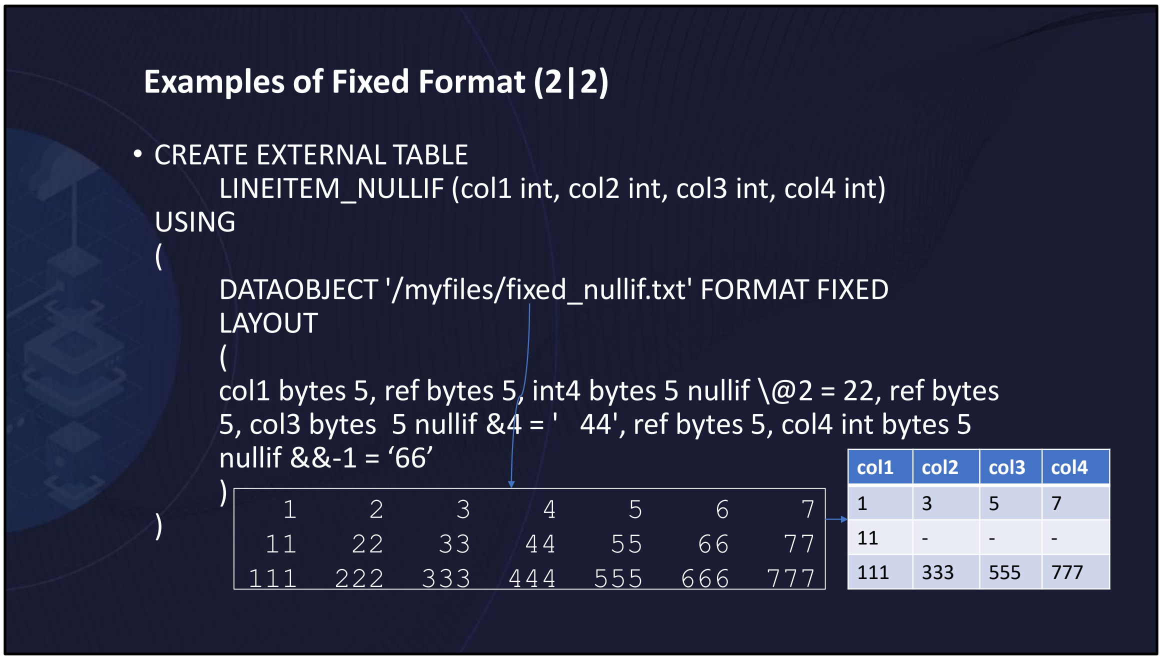


Examples of Fixed Format (1|2)

- CREATE EXTERNAL TABLE LINEITEM_RECORDLENGTH
  (Col1 char(1), col2 int, col3 Char(20))
  USING (DATAOBJECT '/myfiles/lineitem_recordlength.fixed'
  FORMAT FIXED LAYOUT(REF BYTES 1,col1 BYTES \@1, col2 int
  BYTES 1, col3 char(20) BYTES 4) RECORDLENGTH \@1+6)

```
111abcd
122efgh
133ijk
144lmn
```

col1	col2	col3
1	1	abcd
2	2	efgh
3	3	ijk
4	4	lmn

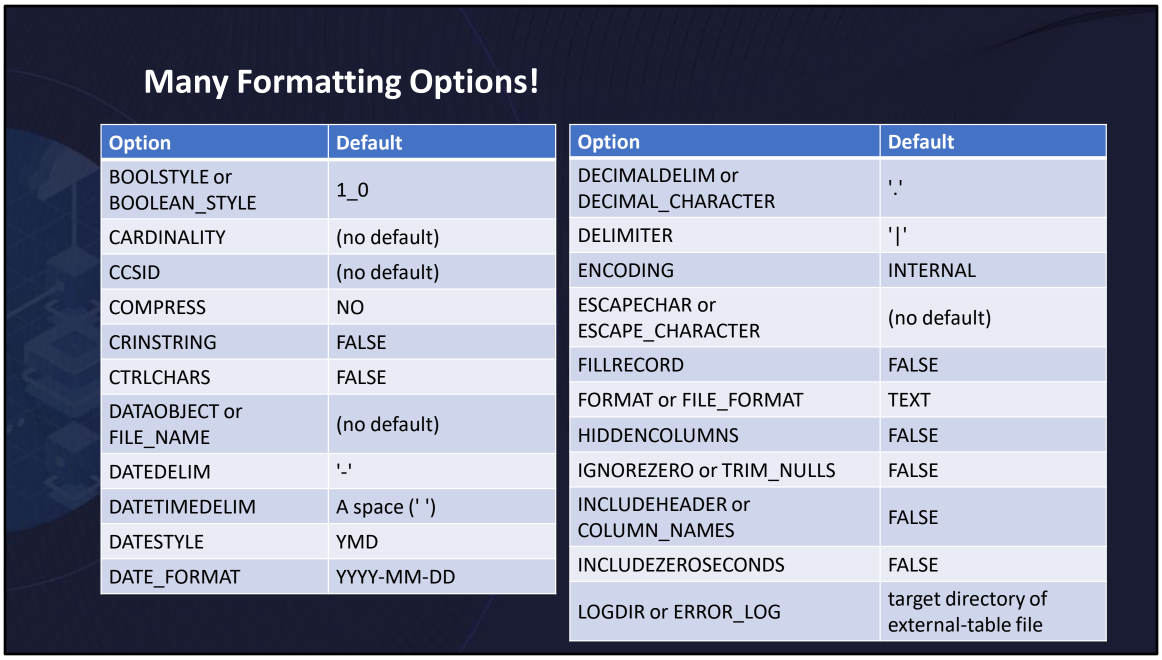


Examples of Fixed Format (2|2)

```
• CREATE EXTERNAL TABLE
        LINEITEM_NULLIF (col1 int, col2 int, col3 int, col4 int)
  USING
  (
        DATAOBJECT '/myfiles/fixed_nullif.txt' FORMAT FIXED
        LAYOUT
        (
        col1 bytes 5, ref bytes 5, int4 bytes 5 nullif \@2 = 22, ref bytes
        5, col3 bytes  5 nullif &4 = '   44', ref bytes 5, col4 int bytes 5
        nullif &&-1 = '66'
        )
  )
```

```
    1    2    3    4    5    6    7
   11   22   33   44   55   66   77
  111  222  333  444  555  666  777
```

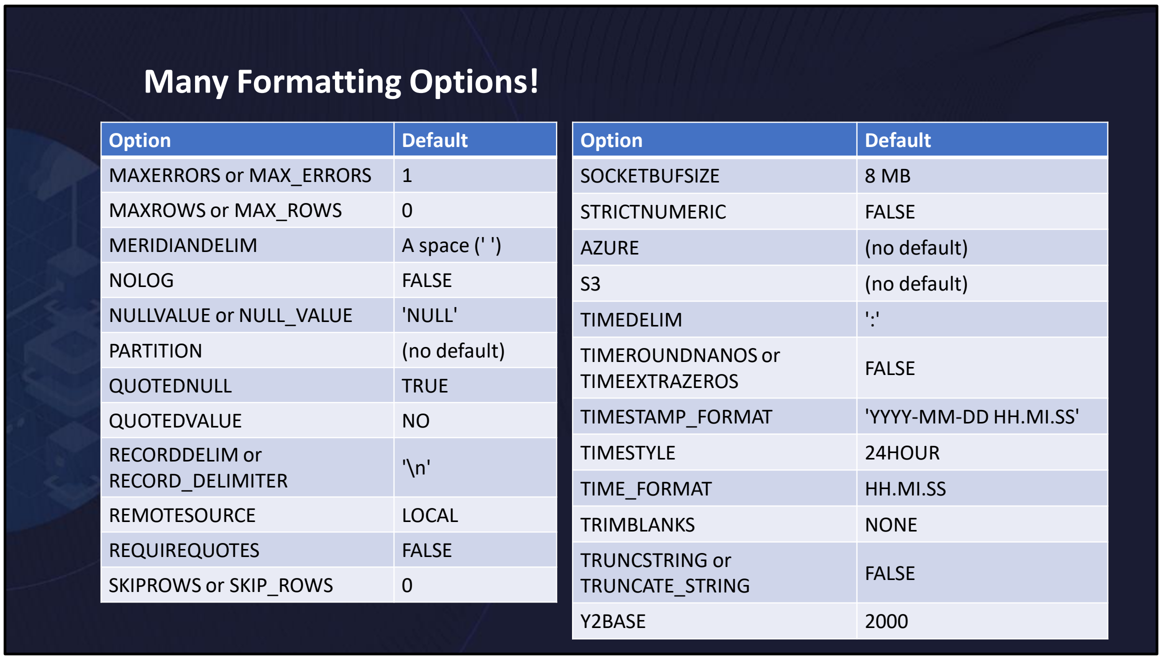
col1	col2	col3	col4
1	3	5	7
11	-	-	-
111	333	555	777



Many Formatting Options!

Option	Default
BOOLSTYLE or BOOLEAN_STYLE	1_0
CARDINALITY	(no default)
CCSID	(no default)
COMPRESS	NO
CRINSTRING	FALSE
CTRLCHARS	FALSE
DATAOBJECT or FILE_NAME	(no default)
DATEDELIM	'-'
DATETIMEDELIM	A space (' ')
DATESTYLE	YMD
DATE_FORMAT	YYYY-MM-DD

Option	Default
DECIMALDELIM or DECIMAL_CHARACTER	'.'
DELIMITER	'|'
ENCODING	INTERNAL
ESCAPECHAR or ESCAPE_CHARACTER	(no default)
FILLRECORD	FALSE
FORMAT or FILE_FORMAT	TEXT
HIDDENCOLUMNS	FALSE
IGNOREZERO or TRIM_NULLS	FALSE
INCLUDEHEADER or COLUMN_NAMES	FALSE
INCLUDEZEROSECONDS	FALSE
LOGDIR or ERROR_LOG	target directory of external-table file



Many Formatting Options!

Option	Default
MAXERRORS or MAX_ERRORS	1
MAXROWS or MAX_ROWS	0
MERIDIANDELIM	A space (' ')
NOLOG	FALSE
NULLVALUE or NULL_VALUE	'NULL'
PARTITION	(no default)
QUOTEDNULL	TRUE
QUOTEDVALUE	NO
RECORDDELIM or RECORD_DELIMITER	'\n'
REMOTESOURCE	LOCAL
REQUIREQUOTES	FALSE
SKIPROWS or SKIP_ROWS	0

Option	Default
SOCKETBUFSIZE	8 MB
STRICTNUMERIC	FALSE
AZURE	(no default)
S3	(no default)
TIMEDELIM	':'
TIMEROUNDNANOS or TIMEEXTRAZEROS	FALSE
TIMESTAMP_FORMAT	'YYYY-MM-DD HH.MI.SS'
TIMESTYLE	24HOUR
TIME_FORMAT	HH.MI.SS
TRIMBLANKS	NONE
TRUNCSTRING or TRUNCATE_STRING	FALSE
Y2BASE	2000

DATE/TIME/TIMESTAMP options - A number of options available some for Netezza compat some for Db2 - Recommendation is to use DATE_FORMAT, TIME_FORMAT, TIMESTAMP_FORMAT for most flexibility. These provide similar formatting options as can be achieved in SQL using VARCHAR_FORMAT scalar functions

Numeric formatting options for Boolean, decimal delimter, decplus blank for export/unload operations, strict numeric for preventing decimals being cast when scale does not strictly match definition

Number of options for string formatting - CCSID is the most important to specify

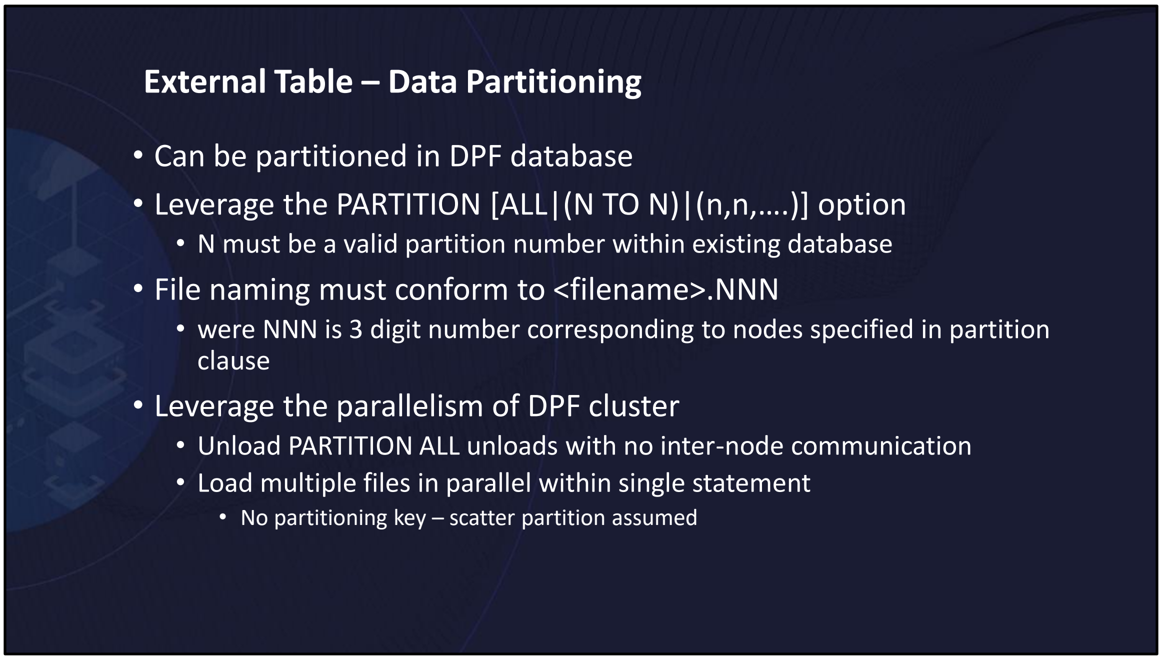
General processing options - COMPRESS to specify data is LZ4 or GZIP compressed , control character processing options, delimiter, escape character, NULL value

LOGDIR to specify location of logfile and NOLOG to suppress the log file

CARDINLITY to provide a hint as to how many rows the external file contains

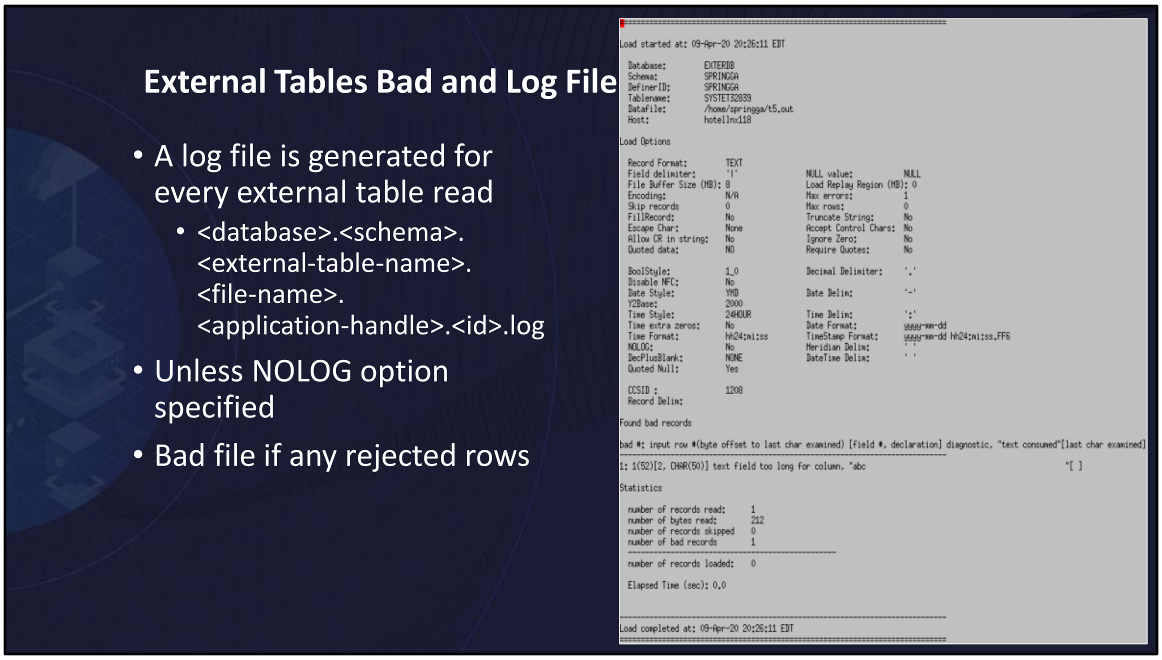
PARTITION for exploiting DPF partitioning processing

File location options REMOTESOURCE and Object Store (S3, AZURE)



External Table – Data Partitioning

- Can be partitioned in DPF database
- Leverage the PARTITION [ALL|(N TO N)|(n,n,….)] option
  - N must be a valid partition number within existing database
- File naming must conform to <filename>.NNN
  - were NNN is 3 digit number corresponding to nodes specified in partition clause
- Leverage the parallelism of DPF cluster
  - Unload PARTITION ALL unloads with no inter-node communication
  - Load multiple files in parallel within single statement
    - No partitioning key – scatter partition assumed



External Tables Bad and Log File

- A log file is generated for every external table read
  - <database>.<schema>.<external-table-name>.<file-name>.<application-handle>.<id>.log
- Unless NOLOG option specified
- Bad file if any rejected rows

```
===========================================================================

Load started at: 09-Apr-20 20:26:11 EDT

  Database:         EXTERDB
  Schema:           SPRINGGA
  DefinerID:        SPRINGGA
  Tablename:        SYSTET32839
  Datafile:         /home/springga/t5.out
  Host:             hotellnx118

Load Options

  Record Format:         TEXT
  Field delimiter:       '|'                NULL value:             NULL
  File Buffer Size (MB): 8                  Load Replay Region (MB): 0
  Encoding:              N/A                Max errors:             1
  Skip records           0                  Max rows:               0
  FillRecord:            No                 Truncate String:        No
  Escape Char:           None               Accept Control Chars:   No
  Allow CR in string:    No                 Ignore Zero:            No
  Quoted data:           NO                 Require Quotes:         No

  BoolStyle:             1_0                Decimal Delimiter:      '.'
  Disable NFC:           No
  Date Style:            YMD                Date Delim:             '-'
  Y2Base:                2000
  Time Style:            24HOUR             Time Delim:             ':'
  Time extra zeros:      No                 Date Format:            yyyy-mm-dd
  Time Format:           hh24:mi:ss         TimeStamp Format:       yyyy-mm-dd hh24:mi:ss.FF6
  NOLOG:                 No                 Meridian Delim:         ' '
  DecPlusBlank:          NONE               DateTime Delim:         ' '
  Quoted Null:           Yes

  CCSID :                1208
  Record Delim:

Found bad records

bad #: input row #(byte offset to last char examined) [field #, declaration] diagnostic, "text consumed"[last char examined]
---------------------------------------------------------------------------
1: 1(52)[2, CHAR(50)] text field too long for column, "abc                                    "[ ]

Statistics

  number of records read:      1
  number of bytes read:        212
  number of records skipped    0
  number of bad records        1
  -----------------------------------------------
  number of records loaded:    0

  Elapsed Time (sec): 0.0


---------------------------------------------------------------------------
Load completed at: 09-Apr-20 20:26:11 EDT
===========================================================================
```

Every external table query generates a log file summarizing the query activity.

File contains the options used for accessing the file.
As in this example if a bad row(s) is found a summary if the issue will be output indicating row # byte offset column # error condition and offending data.
This makes it very easy to diagnosis why a query/load of the external table failed

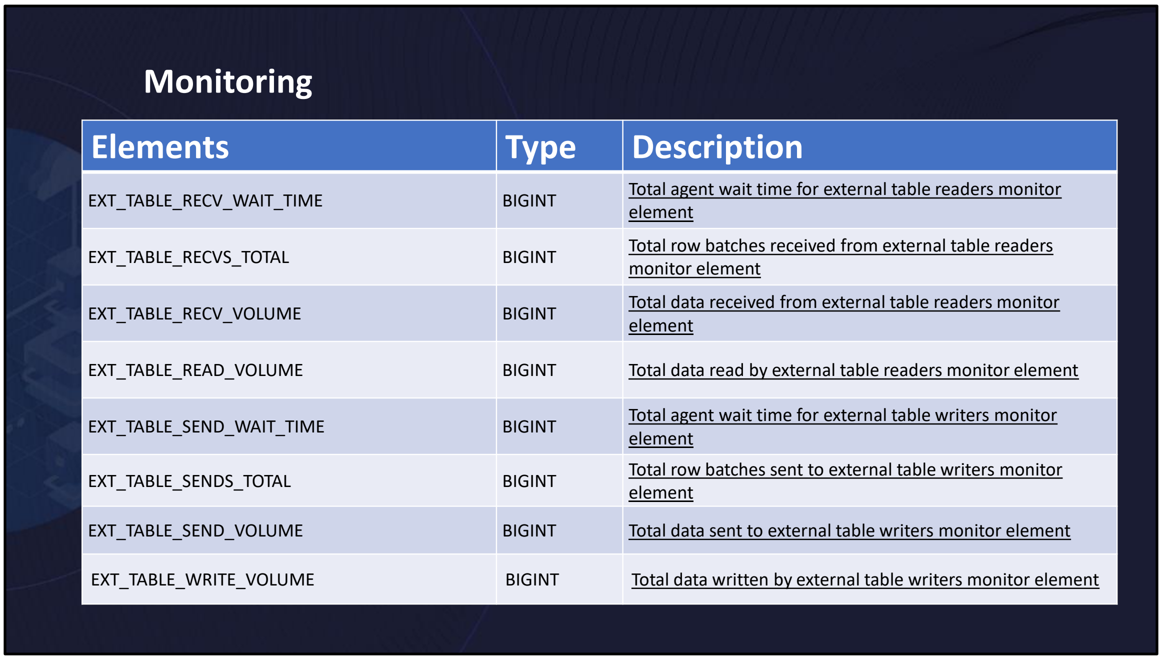
You can suppress the log file by using the NOLOG option

File naming convention is <database>.<schema>.<external-table-name>.<file-name>.<application-handle>.<id>.log

If a row is rejected a bad file is also produced similarly name but with ".bad" extension.
This file will contain the rejected row(s) as they appeared in the original file.
Thus one can correct the bad data in the bad file itself and then us it to complete the load if desired.

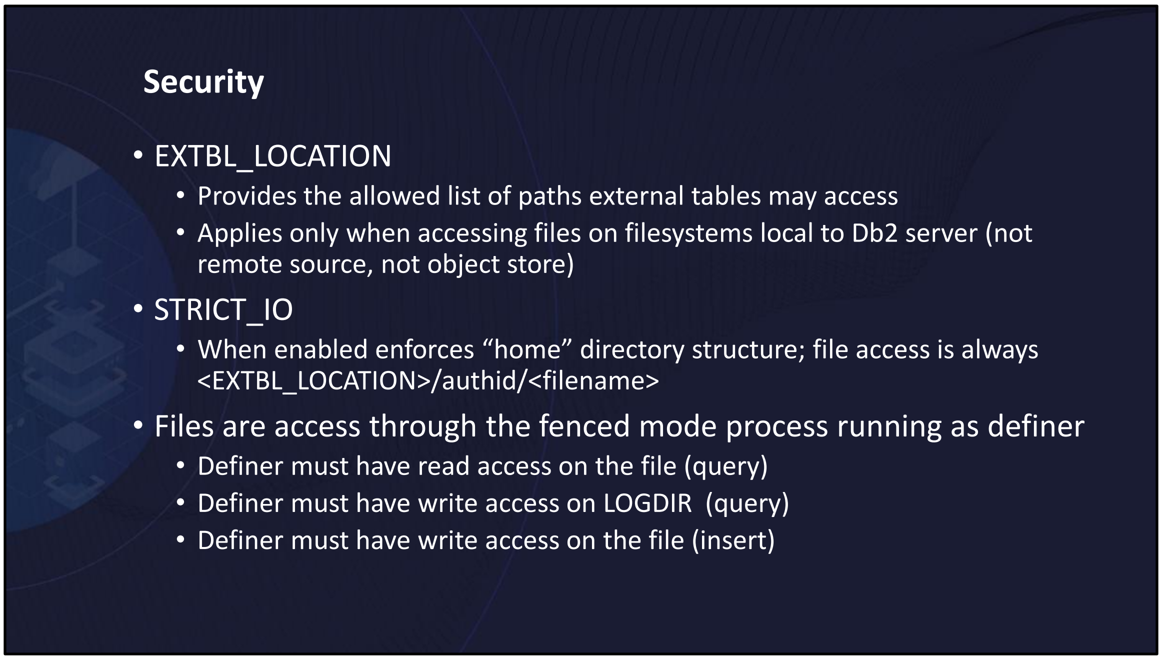
Only rows rejected from by formatting of file into internal form are captured in the bad files.
Any row later failing some SQL processing condition will not be recorded in the .bad file nor count towards maxerrors.

Similarly the timings reported are for the scan of the external tables.
Not the entire SQL processing if the scan happens to be early closed or potentially temped.



Monitoring

Elements	Type	Description
EXT_TABLE_RECV_WAIT_TIME	BIGINT	Total agent wait time for external table readers monitor element
EXT_TABLE_RECVS_TOTAL	BIGINT	Total row batches received from external table readers monitor element
EXT_TABLE_RECV_VOLUME	BIGINT	Total data received from external table readers monitor element
EXT_TABLE_READ_VOLUME	BIGINT	Total data read by external table readers monitor element
EXT_TABLE_SEND_WAIT_TIME	BIGINT	Total agent wait time for external table writers monitor element
EXT_TABLE_SENDS_TOTAL	BIGINT	Total row batches sent to external table writers monitor element
EXT_TABLE_SEND_VOLUME	BIGINT	Total data sent to external table writers monitor element
EXT_TABLE_WRITE_VOLUME	BIGINT	Total data written by external table writers monitor element



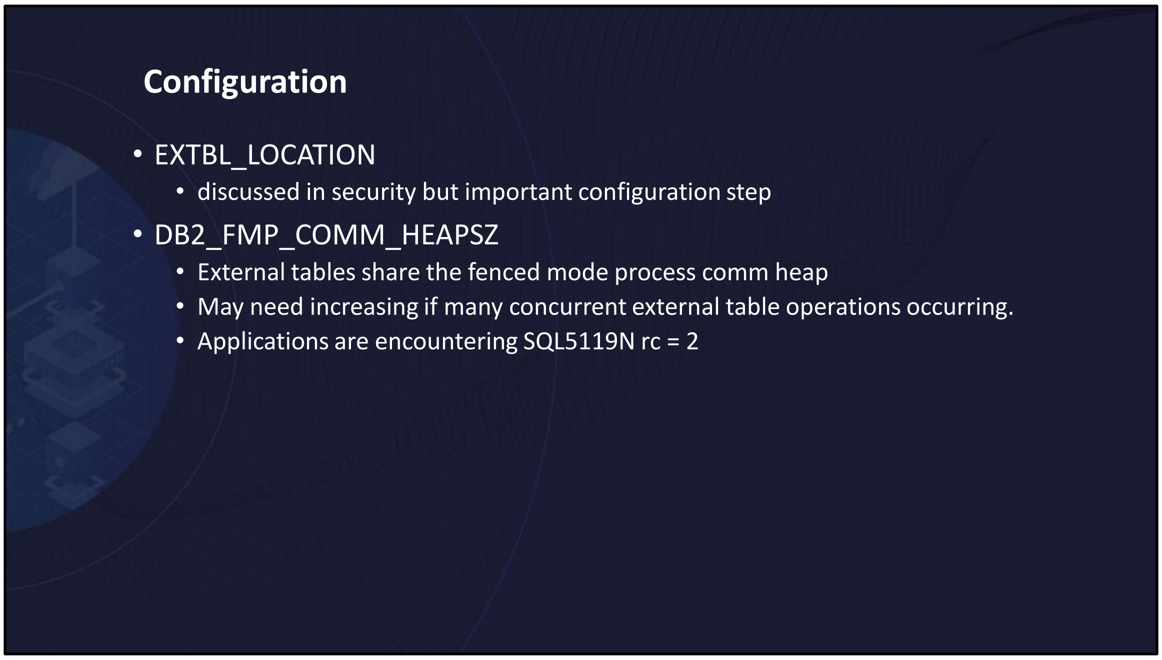
Security

- EXTBL_LOCATION
  - Provides the allowed list of paths external tables may access
  - Applies only when accessing files on filesystems local to Db2 server (not remote source, not object store)
- STRICT_IO
  - When enabled enforces "home" directory structure; file access is always <EXTBL_LOCATION>/authid/<filename>
- Files are access through the fenced mode process running as definer
  - Definer must have read access on the file (query)
  - Definer must have write access on LOGDIR (query)
  - Definer must have write access on the file (insert)

EXTBL_LOCATION DB CFG provides the set of paths that are accessible to external tables.
Both the path to files and path to log directories must be a path within the EXTBL_LOCATION
for external table operation to be successful.   Applies only to LOCAL external tables.

STRICT_IO DB CFG is provided to lock down even tighter access to files.
When enable it enforces a home directory like structure.
The EXTBL_LOCATION may contain only a single path which is then appended
at runtime with the definers AUTHID.
Thus giving each user access to a single directory path only.

To successfully access a file the definer of the external table must have
read access on the file for querying and write access to the LOGDIR location.
To successfully unload data into a external table write access to the file is required.

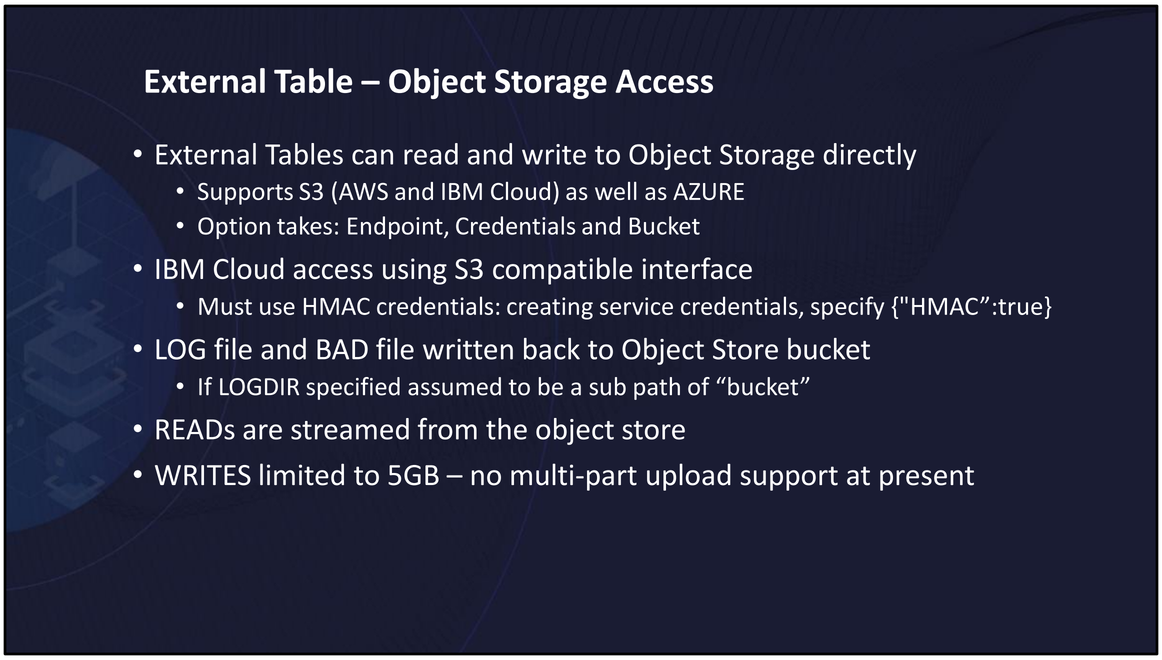


Configuration

- EXTBL_LOCATION
  - discussed in security but important configuration step
- DB2_FMP_COMM_HEAPSZ
  - External tables share the fenced mode process comm heap
  - May need increasing if many concurrent external table operations occurring.
  - Applications are encountering SQL5119N rc = 2

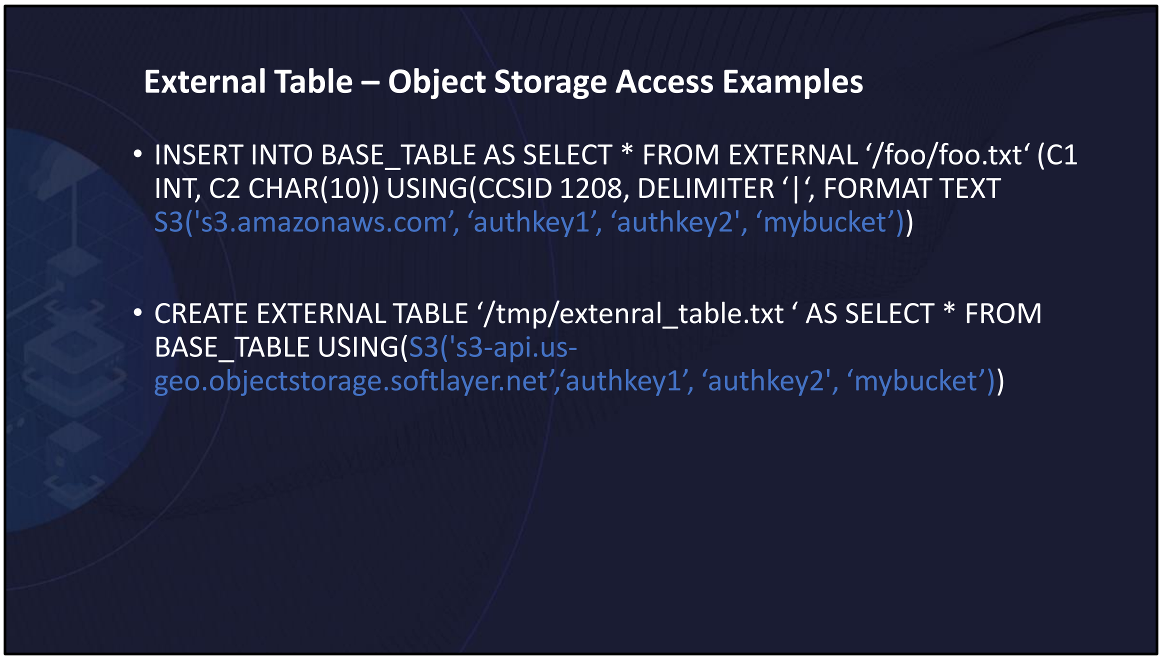
CLI LOAD Extension

- Enhanced to leverage external tables
  - Specify SQL_USE_LOAD_WITH_ET to use External tables instead of LOAD
- Supports “LOAD INSERT” only
- Implicitly used for “LOAD INSERT“ within Data Server Driver
  - i.e when no load API available
- Leverages Db2’s binary format



External Table – Object Storage Access

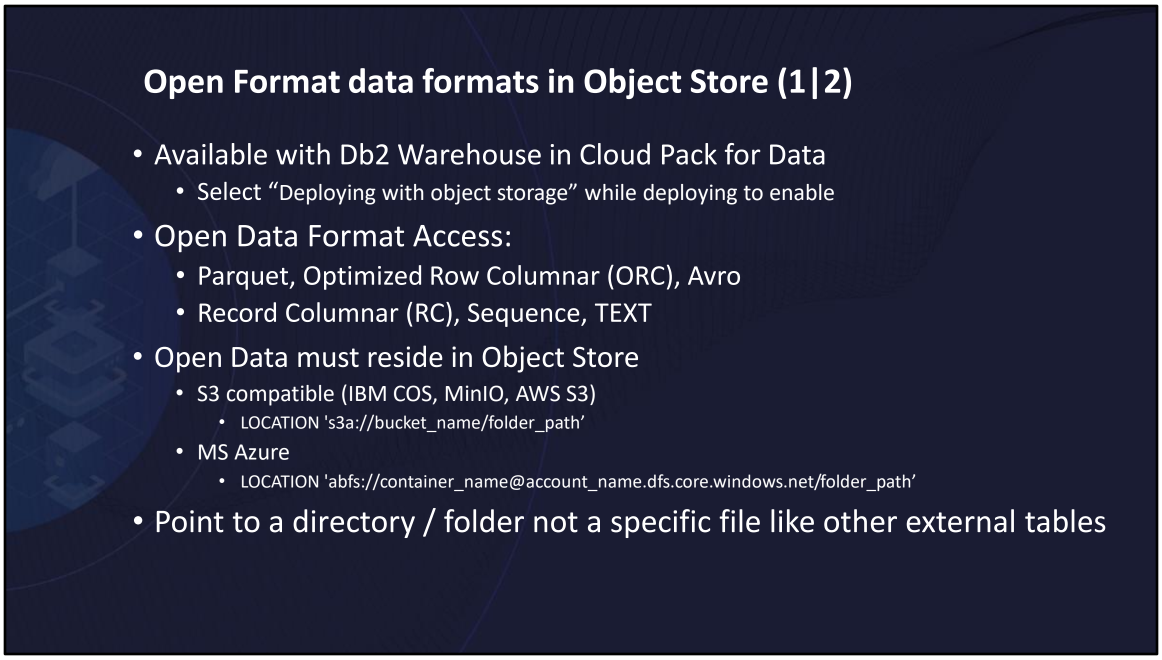
- External Tables can read and write to Object Storage directly
  - Supports S3 (AWS and IBM Cloud) as well as AZURE
  - Option takes: Endpoint, Credentials and Bucket
- IBM Cloud access using S3 compatible interface
  - Must use HMAC credentials: creating service credentials, specify {"HMAC":true}
- LOG file and BAD file written back to Object Store bucket
  - If LOGDIR specified assumed to be a sub path of "bucket"
- READs are streamed from the object store
- WRITES limited to 5GB – no multi-part upload support at present



External Table – Object Storage Access Examples

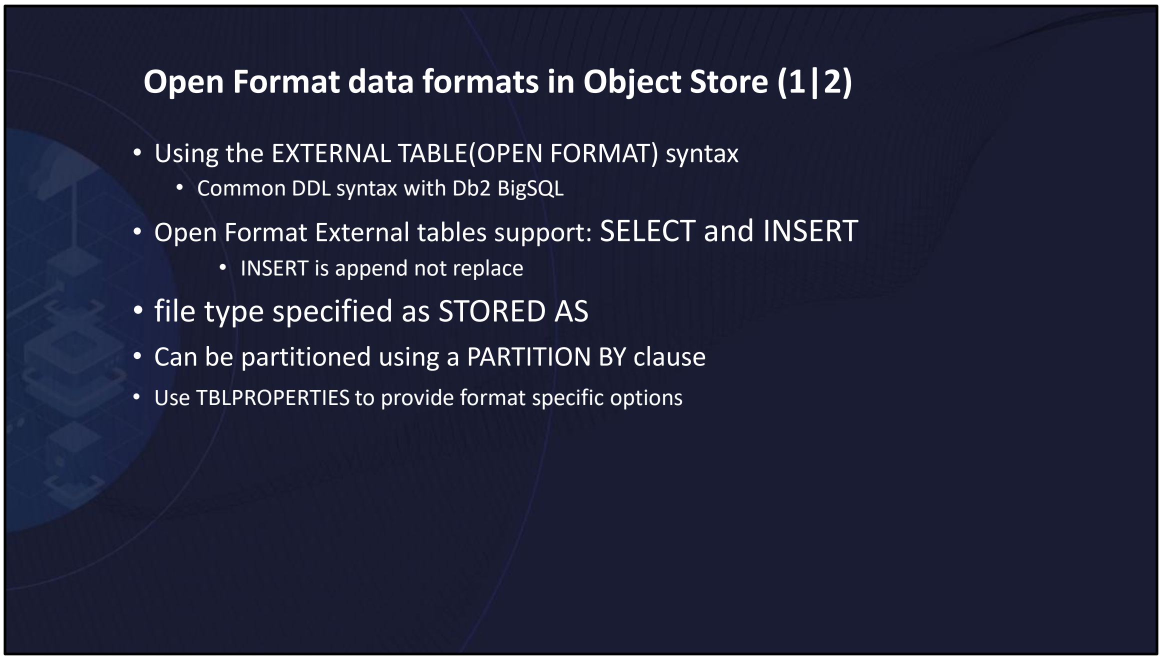
- INSERT INTO BASE_TABLE AS SELECT * FROM EXTERNAL '/foo/foo.txt' (C1 INT, C2 CHAR(10)) USING(CCSID 1208, DELIMITER '|', FORMAT TEXT S3('s3.amazonaws.com', 'authkey1', 'authkey2', 'mybucket'))

- CREATE EXTERNAL TABLE '/tmp/extenral_table.txt ' AS SELECT * FROM BASE_TABLE USING(S3('s3-api.us-geo.objectstorage.softlayer.net','authkey1', 'authkey2', 'mybucket'))



Open Format data formats in Object Store (1|2)

- Available with Db2 Warehouse in Cloud Pack for Data
  - Select “Deploying with object storage” while deploying to enable
- Open Data Format Access:
  - Parquet, Optimized Row Columnar (ORC), Avro
  - Record Columnar (RC), Sequence, TEXT
- Open Data must reside in Object Store
  - S3 compatible (IBM COS, MinIO, AWS S3)
    - LOCATION 's3a://bucket_name/folder_path’
  - MS Azure
    - LOCATION 'abfs://container_name@account_name.dfs.core.windows.net/folder_path’
- Point to a directory / folder not a specific file like other external tables

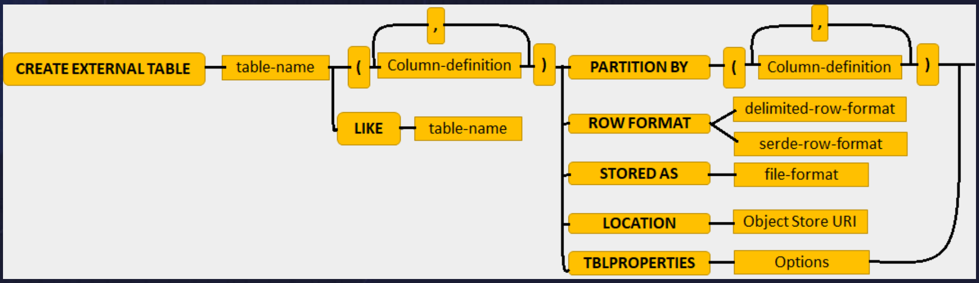


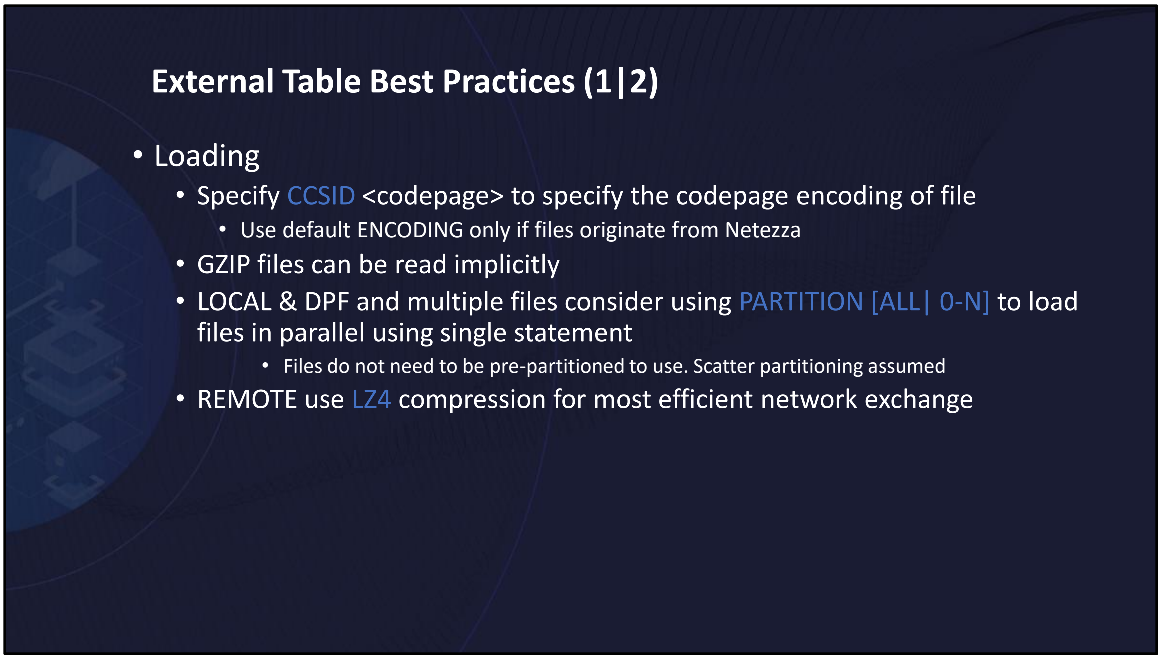
Open Format data formats in Object Store (1|2)

- Using the EXTERNAL TABLE(OPEN FORMAT) syntax
  - Common DDL syntax with Db2 BigSQL
- Open Format External tables support: SELECT and INSERT
  - INSERT is append not replace
- file type specified as STORED AS
- Can be partitioned using a PARTITION BY clause
- Use TBLPROPERTIES to provide format specific options

CREATE EXTERNAL TABLE (OPEN FORMATS)

- The external table has a name and catalog entry like a normal table
  - File formats only available through “named” external tables (ie. no transient format)
- Used to access open data file formats Parquet, ORC, Avro
  - data files must be stored in Object Storage

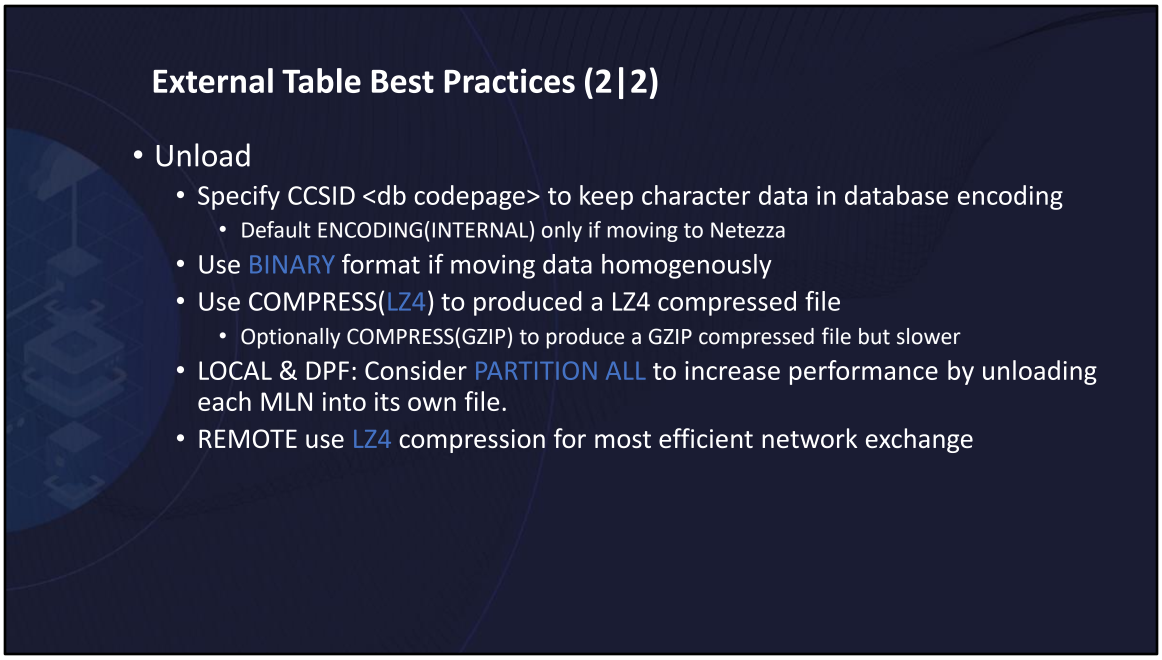



External Table Best Practices (1|2)

- Loading
  - Specify CCSID <codepage> to specify the codepage encoding of file
    - Use default ENCODING only if files originate from Netezza
  - GZIP files can be read implicitly
  - LOCAL & DPF and multiple files consider using PARTITION [ALL| 0-N] to load files in parallel using single statement
    - Files do not need to be pre-partitioned to use. Scatter partitioning assumed
  - REMOTE use LZ4 compression for most efficient network exchange

There are a few best practices for using external tables

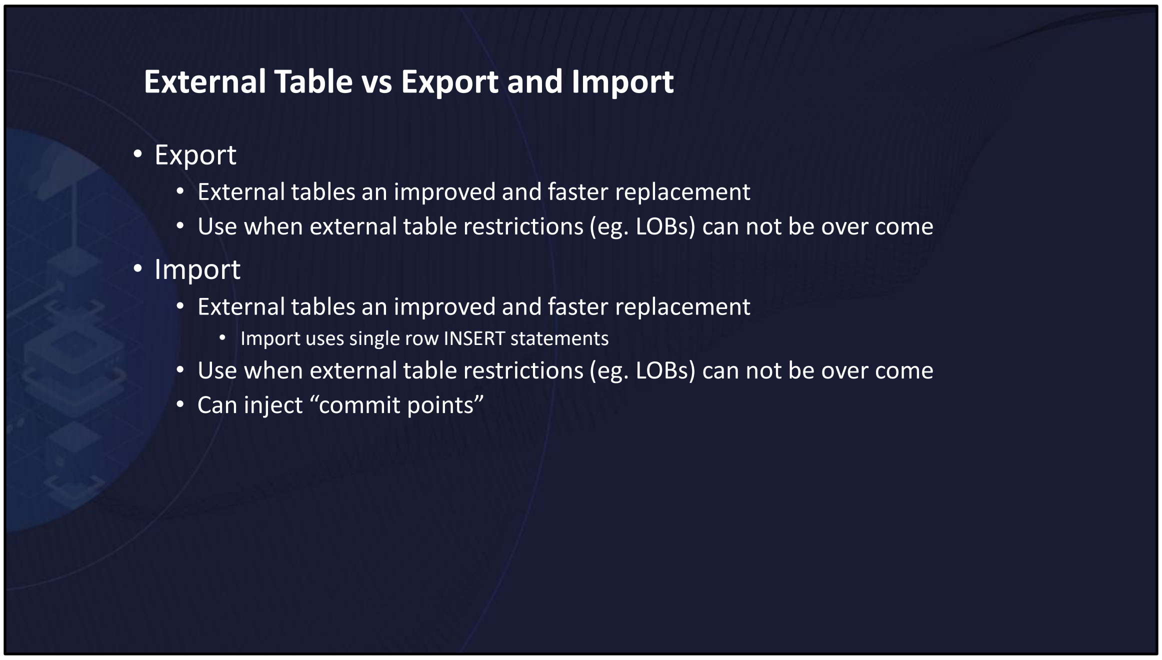
Specify the CCSID option to explicitly indicate the encoding of the file.
Default option assumes Netezza internal format which is really only valid if the data originated from Netezza
GZIP files can be loaded directly no need to unzip the files first
LZ4 files can also be loaded but compress option needs to be specified
LOCAL in a DPF environment splitting the files with .NNN extension where NNN is within the range of defined nodes can speed up querying .
REMOTE external tables with network distance may benefit from using REMOTESOURCE LZ4 to compress the network traffic
CLI LOAD will be enhanced to leverage ET implicitly within the DS Driver but can be explicitly requested with new option in full client as well



External Table Best Practices (2|2)

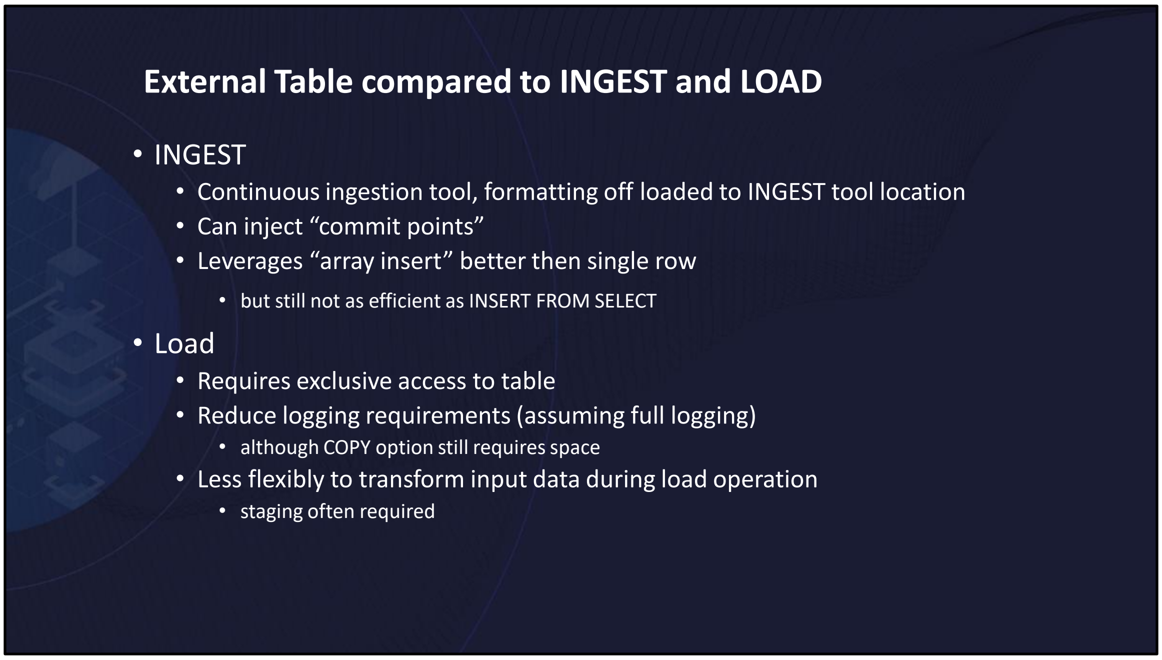
- Unload
  - Specify CCSID <db codepage> to keep character data in database encoding
    - Default ENCODING(INTERNAL) only if moving to Netezza
  - Use BINARY format if moving data homogenously
  - Use COMPRESS(LZ4) to produced a LZ4 compressed file
    - Optionally COMPRESS(GZIP) to produce a GZIP compressed file but slower
  - LOCAL & DPF: Consider PARTITION ALL to increase performance by unloading each MLN into its own file.
  - REMOTE use LZ4 compression for most efficient network exchange

1) LOCAL unload should again always specify CCSID and in most cases want equal to the database codepage to avoid codepage conversion on unload
2) Consider the COMPRESS option LZ4 or GZIP to unload into a compressed file for space savings
3) LOCAL Unload in DPF should use PARTITION ALL to generate a file per node to remove data movement across nodes
4) REMOTE again leveraging REMOUTESOURCE LZ4 may benefit the network traffic



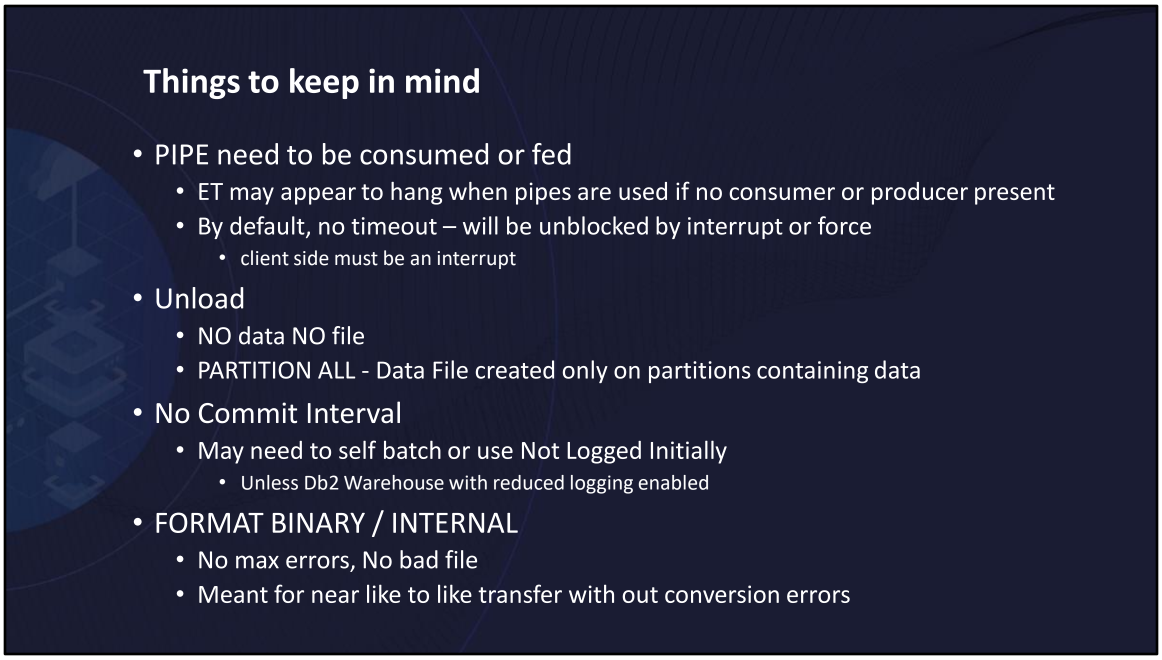
External Table vs Export and Import

- Export
  - External tables an improved and faster replacement
  - Use when external table restrictions (eg. LOBs) can not be over come
- Import
  - External tables an improved and faster replacement
    - Import uses single row INSERT statements
  - Use when external table restrictions (eg. LOBs) can not be over come
  - Can inject “commit points”



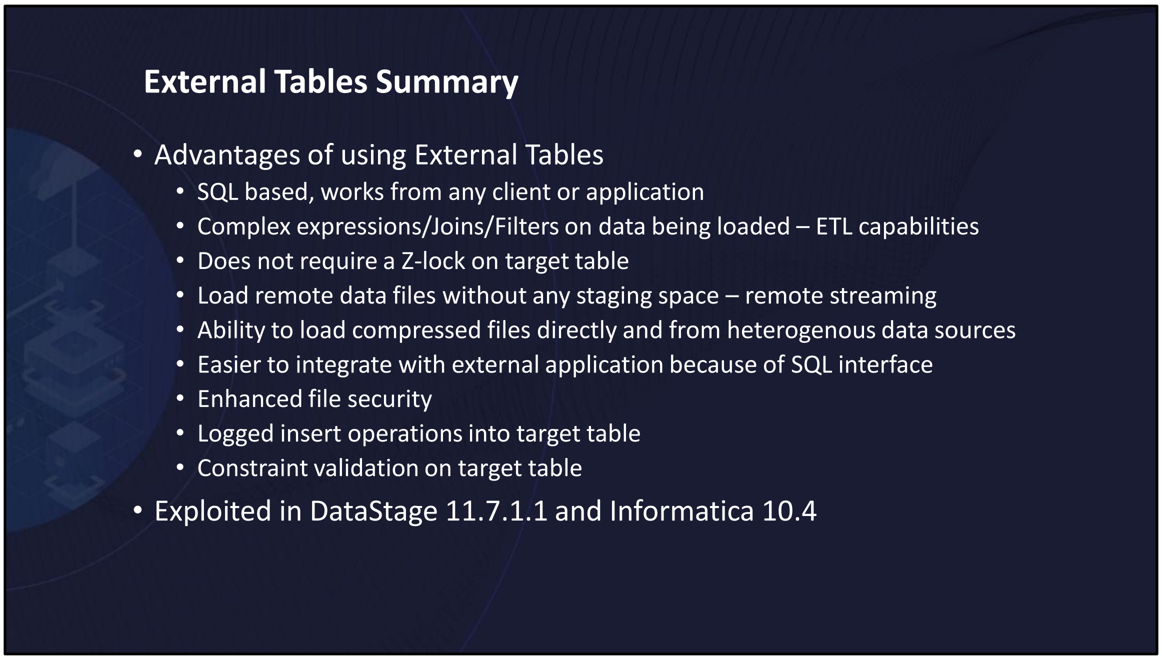
External Table compared to INGEST and LOAD

- INGEST
  - Continuous ingestion tool, formatting off loaded to INGEST tool location
  - Can inject “commit points”
  - Leverages “array insert” better then single row
    - but still not as efficient as INSERT FROM SELECT
- Load
  - Requires exclusive access to table
  - Reduce logging requirements (assuming full logging)
    - although COPY option still requires space
  - Less flexibly to transform input data during load operation
    - staging often required



Things to keep in mind

- PIPE need to be consumed or fed
  - ET may appear to hang when pipes are used if no consumer or producer present
  - By default, no timeout – will be unblocked by interrupt or force
    - client side must be an interrupt
- Unload
  - NO data NO file
  - PARTITION ALL - Data File created only on partitions containing data
- No Commit Interval
  - May need to self batch or use Not Logged Initially
    - Unless Db2 Warehouse with reduced logging enabled
- FORMAT BINARY / INTERNAL
  - No max errors, No bad file
  - Meant for near like to like transfer with out conversion errors



External Tables Summary

- Advantages of using External Tables
  - SQL based, works from any client or application
  - Complex expressions/Joins/Filters on data being loaded – ETL capabilities
  - Does not require a Z-lock on target table
  - Load remote data files without any staging space – remote streaming
  - Ability to load compressed files directly and from heterogenous data sources
  - Easier to integrate with external application because of SQL interface
  - Enhanced file security
  - Logged insert operations into target table
  - Constraint validation on target table
- Exploited in DataStage 11.7.1.1 and Informatica 10.4

Advantages of using External Tables
Fully SQL based, works from any client or application
Complex expressions/Joins/Filters on data being loaded – ETL capabilities
Does not require a Z-lock on target table, Can drive parallel inserts into a table to improve ingestion rates.
Load remote data files without any staging space – Remote streaming
Ability to load compressed files directly and from heterogenous data sources
Easier to integrate with external application because of SQL interface
Enhanced Security - EXTBL_LOCATION provides restriction on file access – File permissions definer based not instance owner
Logged insert operations into target table
Constraint validation on target table

Generally just a much better version of IMPORT or EXPORT. And in some cases an improvement over LOAD.

Acknowledgements and Disclaimers

Speaker: Mike Springgay
Company: IBM
Email Address: springga@ca.ibm.com

Don’t forget to fill out a session evaluation!

Mike Springgay is the Db2 Warehouse Architect focused on the common engine components. Prior to that he was responsible for extending Db2's SQL compatibility features, routine infrastructure and client server connectivity.