Using SQL*Loader

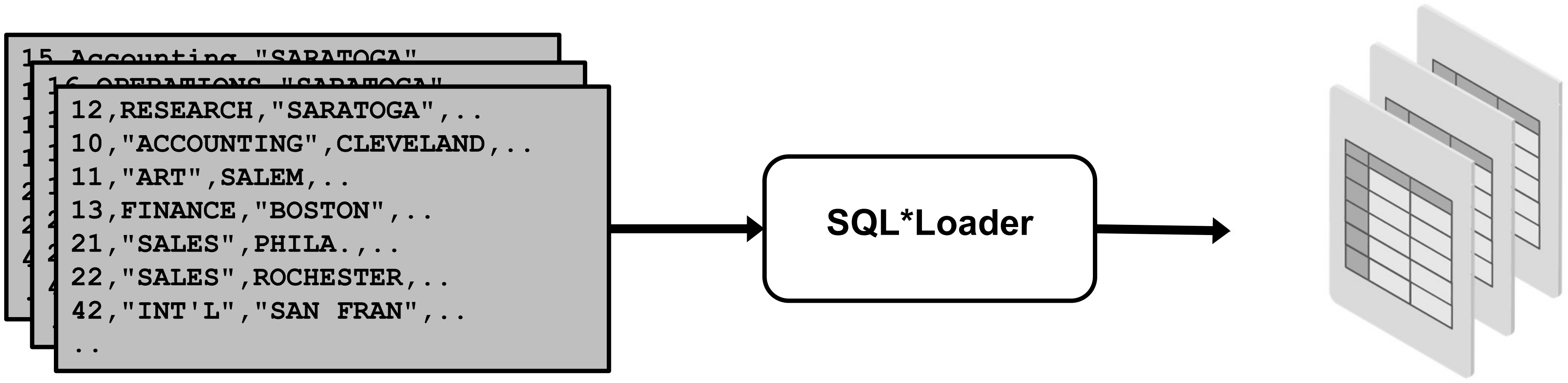
By Ahmed Baraka

Objectives

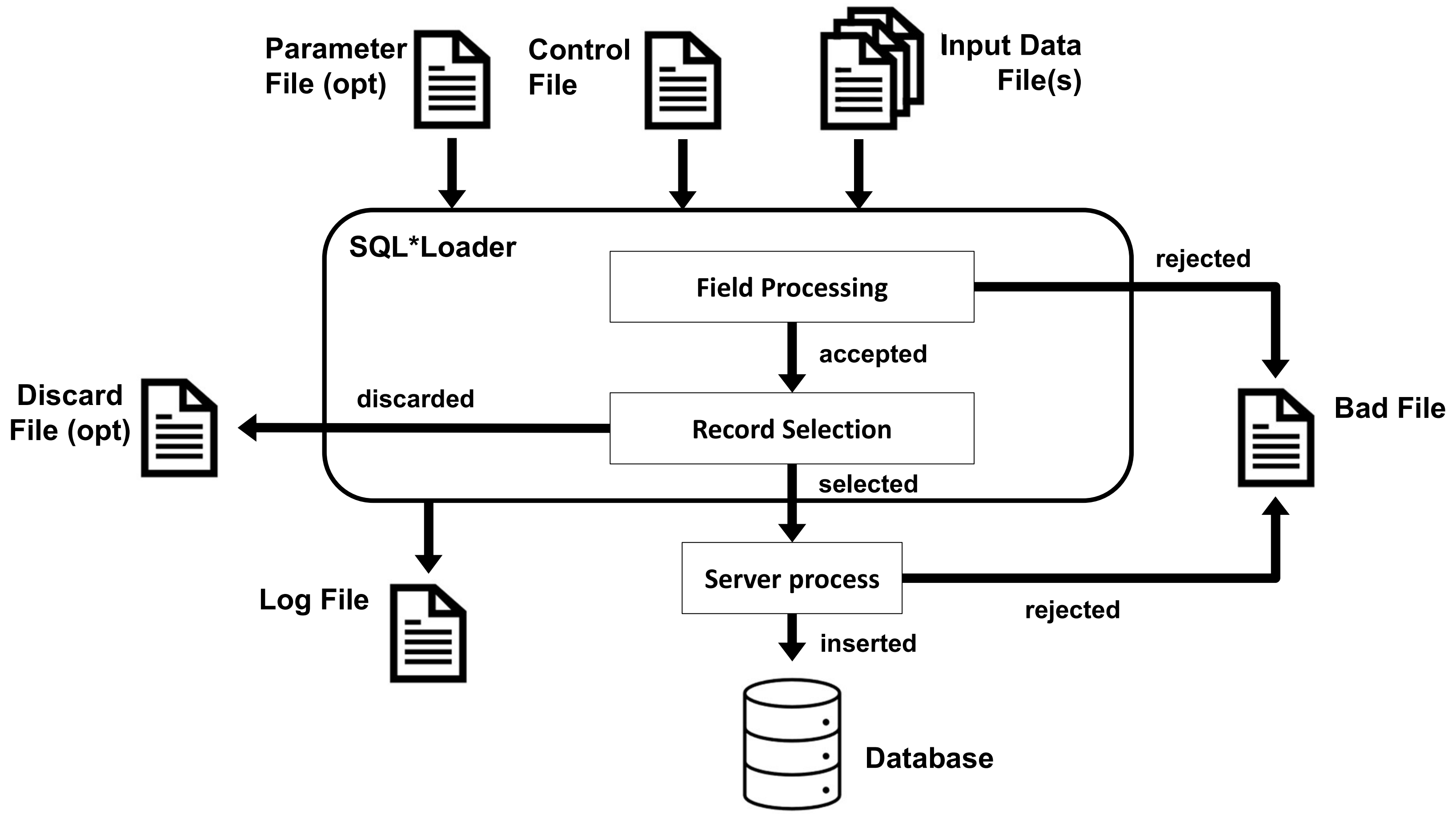
In this lecture, you will learn how to perform the following:

- Describe SQL*Loader target, components, and features
- Start SQL*Loader and use command-line parameters
- Create SQL*Loader control files
- Configure control files for different loading scenarios
- Set more control file configuration options
- Use multiple **`INTO TABLE`** clauses
- Specify the field list contents
- Use **`POSITION`** keyword
- Describe the differences between the SQL*Loader loading methods
- Install SQL*Loader Case Studies

SQL*Loader Target



SQL*Loader Components



SQL*Loader Framework Components

Component	Description
Control File	A text file edited by the administrator to instruct SQL*Loader on how to move the data
Data Files	External files containing the data to be uploaded
Parameter File	A text file to group SQL*Loader command line parameter values
Bad File	Contains records that were rejected, either by SQL*Loader or by Oracle Database
Discard File	Contains records that are filter out by the SQL*Loader
Log file	Contains a summary of the load, including a description of any errors that occurred during the load.

About SQL*Loader

- Loads data from external files to database tables
- Features:
  - Provides control on load operation:
    - Selectively load data
    - Basic data manipulation using SQL Functions
    - Setting loading methods: conventional path, direct path, or external table loads
  - Works with different platforms
  - Generates unique sequential key values in specified columns.
  - Capable of loading into complex data types and objects
  - Can be run from OS script files
  - Direct Path Load API is available for developers
- Not a sophisticated ETL product

Supported Destination Data Types

- Scalar types
- The four LOB data types: **`BLOB`**, **`CLOB`**, **`NCLOB`**, **`BFILE`**
- **`VARRAYS`**
- Nested tables
- Column and row objects
- XMLType

Starting SQL*Loader

- A common option to start SQL*Loader:

```
sqlldr CONTROL=<control-file> LOG=<log-file> [parameter=value]
```

- Username/password and connection name can be provided after the **`sqlldr`** or by setting the parameter **`USERID`**

- Command line parameters can also be provided by:
  - **`OPTIONS`** clause in the control file
  - **`PARFILE`** in the command line parameter

SQL*Loader Command-Line Parameters

Column	Description
`CONTROL`	Specifies the name of the SQL*Loader control file
`DATA`	Specifies the names of the data files (default extension is **`dat`**). If also specified in the control file, the first data file in the control file is ignored.
`BAD`	Specifies the name of the bad file (default: data file name with **`bad`** ext)
`LOG`	Specifies the name of the log file (default: control file name with **`log`** ext)
`DISCARD`	Specifies the name of the discard file
`DIRECT`	**`TRUE`**: SQL*Loader uses direct load method **`FALSE`**: SQL*Loader uses conventional load method
`EXTERNAL_TABLE`	Specifies whether to load data using the external tables option
`SILENT`	Used to suppress some of the content that is written to the screen
`SKIP`	Skip a specific number of logical records from the beginning of the data file
`TRIM`	Whether to trim spaces from the beginning or the end of a text field

About SQL*Loader Control File

- Where SQL*Loader will find the data to load?
- How SQL*Loader expects that data to be formatted?
- How SQL*Loader will be configured (memory management, rejecting records, interrupted load handling, and so on) as it loads the data?
- How SQL*Loader will manipulate the data being loaded?

Control File Parameters

Component	Description
`INFILE`	Specifies the name of the data file that is to be loaded.
`INTO TABLE`	Specifies the name of the table where the data will be loaded.
`FIELDS` or `FILLER`	These parameters specify the layout of the data in the input file, such as the position of each field, its length, and its type.
`TRAILING NULLCOLS` or `TRAILING NULLS`	They specify that any columns that are not specified in the control file should be set to NULL in the target table.
`BADFILE`	Specifies the name of the file where records that cannot be loaded will be stored.
`DISCARDFILE`	Specifies the name of the file where records that are discarded during the loading process will be stored.
`APPEND` or `REPLACE`	These parameters specify whether to append to or replace existing data in the target table.
`CHARACTERSET`	Specifies the character set of the input file.
`SKIP`	Specifies the number of records to skip at the beginning of the input file.
`WHEN`	Specifies a condition that must be met for a record to be loaded into the target table.

Specifying Bad and Discard Files

- At the command line:

```
sqlldr ... BAD=mydatafile.bad DISCARD=mydatafile.dsc
```

- In the Control File:

```
...
BADFILE 'mydatafile.bad'
DISCARDFILE 'mydatafile.dsc'
...
```

SQL*Loader Control File Example 1

- The Control file sample:

```
LOAD DATA
INFILE 'dept.dat'
INTO TABLE DEPT
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(DEPTNO, DNAME, LOC)
```

- Data file sample: relatively positioned columns based on delimiters

```
12,RESEARCH,"SARATOGA"
10,"ACCOUNTING",CLEVELAND
11,"ART",SALEM
13,FINANCE,"BOSTON"
21,"SALES",PHILA.
22,"SALES",ROCHESTER
42,"INT'L","SAN FRAN"
```

- The destination Table:

```
Name       Type
---------- ----------
DEPTNO     NUMBER(2)
DNAME      VARCHAR2(14)
LOC        VARCHAR2(13)
ZIP_CODE   CHAR(6)
```

SQL*Loader Control File Example 2

- Sample Data: fixed positioned

```
7782 CLARK      MANAGER   7839  2572.50          10
7839 KING       PRESIDENT       5500.00          10
7934 MILLER     CLERK     7782   920.00          10
7566 JONES      MANAGER   7839  3123.75          20
7499 ALLEN      SALESMAN  7698  1600.00   300.00 30
7654 MARTIN     SALESMAN  7698  1312.50  1400.00 30
7658 CHAN       ANALYST   7566  3450.00          20
```

SQL*Loader Control File Example 2

- The Control file sample:

```
LOAD DATA
INFILE 'ulcase2.dat'
INTO TABLE EMP
( EMPNO    POSITION(01:04) INTEGER EXTERNAL,
  ENAME    POSITION(06:15) CHAR,
  JOB      POSITION(17:25) CHAR,
  MGR      POSITION(27:30) INTEGER EXTERNAL,
  SAL      POSITION(32:39) DECIMAL EXTERNAL,
  COMM     POSITION(41:48) DECIMAL EXTERNAL,
  DEPTNO   POSITION(50:51) INTEGER EXTERNAL)
```

SQL*Loader Common Data Types

- INTEGER
- FLOAT
- DECIMAL
- CHAR
- DATE
- TIMESTAMP
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

Numeric EXTERNAL

- The numeric **`EXTERNAL`** datatypes are the numeric datatypes (**`INTEGER`**, **`FLOAT`**, **`DECIMAL`**, and **`ZONED`**) specified as **`EXTERNAL`**, with optional length and delimiter specifications
- **`FLOAT EXTERNAL`** data can be given in either scientific or regular notation. Both "5.33" and "533E-2" are valid representations of the same value.

Specifying Data Files

- Get the data from specific data file:

```
INFILE 'mydatafile.dat'
```

- Get the data from the Control File itself:

```
LOAD DATA
INFILE *
INTO TABLE DEPT
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(DEPTNO, DNAME, LOC)
BEGINDATA
12,RESEARCH,"SARATOGA"
10,"ACCOUNTING",CLEVELAND
11,"ART",SALEM
```

Defining Record Terminator

- In Linux, it defaults to line feed character '**`\n`**'
- In Windows, it defaults to line feed character '**`\n`**' or carriage return and line feed '**`\r\n`**'
- Causes an issue when loading a file into a Linux system while the file is created in Windows
- Solutions: use **`str`** option with the **`INFILE`** parameter or **`FIELDS CSV`**

```
INFILE 'sales1123.dat' "str '\r\n'"
```

```
INFILE "sales1123.dat"
INTO TABLE sales23
FIELDS CSV WITH EMBEDDED
TRAILING NULLCOLS
(..
```

Field Termination Specification

- Is set by **`TERMINATED`** clause at the table level or column level:

```
TERMINATED BY WHITESPACE | X'<hexa> | '<string>' | EOF
```

- Examples:

```
..
DEPTNO INTEGER EXTERNAL TERMINATED BY X'9',
..
```

```
INTO TABLE persons
REPLACE
WHEN typid = 'P' TREAT AS person_t
FIELDS TERMINATED BY ","
 (TYPID   FILLER  POSITION(1) CHAR,
  NAME            CHAR,
  AGE             CHAR)
```

Specifying Table-Specific Loading Method

- Defined at the control file level (global) or at the table level:

Column	Description
INSERT	Load into empty table, return error if not empty
REPLACE	Delete all the rows from the table before loading the new data
TRUNCATE	Issue the statement **TRUNCATE TABLE <table_name> REUSE STORAGE** before loading the data
APPEND	Load the data even if the table is not empty

- Example

```
LOAD DATA
INFILE 'mydatafile.dat'
APPEND
INTO TABLE EMP
...
```

Replacing Specific Characters with NULL

- To load a table character field as **`NULL`** when it contains certain character

```
NULLIF {=|!=}{"char_string"|x'hex_string'|BLANKS}
```

- **`NULLIF`** can be specified at the table level and at the column level
- Example:

```
NULLIF = "NULL"
```

Loading Data Based on a Condition

- Use **WHEN** clause after the table name as follows:

```
WHEN <col-pos>|<field-name> <operator> {'string'|X'hex_str'|BLANKS}
```

- Only **AND** operand can be used with multiple conditions
- Examples:
  - Load the record only if the value of the **COURSE_CODE** is not **'NULL'**

```
INTO TABLE STUDENT_COURSES
WHEN COURSE_CODE != 'NULL'
( ...
```

  - Load the record if the value of the fifth column in the record is **Y:**

```
INTO TABLE STUDENT_COURSES
WHEN (5) = 'Y'
( ...
```

Handling Records with Missing Trailing Data

- The **`TRAILING NULLCOLS`** clause tells SQL*Loader to treat any relatively positioned columns that are not present in the record as null columns.
- Example:

```
INTO TABLE dept
TRAILING NULLCOLS
( deptno CHAR TERMINATED BY " ",
  dname  CHAR TERMINATED BY WHITESPACE,
  loc    CHAR TERMINATED BY WHITESPACE
)
```

```
10 Accounting
```

Using Multiple `INTO TABLE` Clauses

- Extracting Multiple Logical Records: fixed positioning

```
1319 Salim     1120 Yvonne
1121 Albert    1130 Thomas
```

```
INTO TABLE emp
(empno POSITION(1:4) INTEGER EXTERNAL,
 ename POSITION(6:15) CHAR)

INTO TABLE emp
(empno POSITION(17:20) INTEGER EXTERNAL,
 ename POSITION(21:30) CHAR)
```

Using Multiple `INTO TABLE` Clauses

- Extracting Multiple Logical Records: relative positioning

```
1319 Salim 1120 Yvonne
1121 Albert 1130 Thomas
```

```
INTO TABLE emp
(empno INTEGER EXTERNAL TERMINATED BY " ",
 ename CHAR TERMINATED BY " ")

INTO TABLE emp
(empno INTEGER EXTERNAL TERMINATED BY " ",
 ename CHAR TERMINATED BY WHITESPACE)
```

Field List Contents

- You can specify position, data type, conditions, and delimiters. For example:

```
HIREDATE SYSDATE,
DEPTNO POSITION(4:5) INTEGER EXTERNAL(2),
JOB_CODE CHAR TERMINATED BY WHITESPACE NULLIF JOB_CODE=BLANKS
"UPPER(:JOB_CODE)",
SALARY POSITION(51) CHAR TERMINATED BY WHITESPACE
"TO_NUMBER(:SAL,'$99,999.99')"
```

- We can sett a column to the current date/time using the **`SYSDATE`** parameter
  - The column must be of **`DATE`** or character data type
  - The parameter value is processed for each loaded batch
- The **`WHITESPACE`** delimiter includes spaces, tabs, blanks, line feeds, form feeds, or carriage returns.
- SQL operators can be applied to field data with the SQL string
  - The column name is used as a bind variable in the SQL string

Setting a Column to a Unique Sequence Number

- The **`SEQUENCE`** parameter generates an incremented sequence number for a particular column:

```
<col-name> SEQUENCE ( { COUNT | MAX | n } [,m])
```

Keyword	Description
`<col-name>`	The name of the column in the database to which to assign the sequence.
`COUNT`	The sequence starts with the number of records already in the table plus the increment.
`MAX`	The sequence starts with the current maximum value for the column plus the increment.
`n`	Specifies the specific sequence number to begin with.
`m`	The interment value (defaults to 1)

Setting Sequence Number: Example

- The **`LOADSEQ`** column is incremented by 1 starting from the maximum column value in the table:

```
LOAD DATA
INFILE 'customers.dat'
APPEND
INTO TABLE CUSTOMERS
( CUSTOMER_ID POSITION(01:11) INTEGER EXTERNAL,
  CUST_NAME   POSITION(13:42) CHAR "UPPER(:CUST_NAME)",
  EMAIL       POSITION(44:73) CHAR,
  LOADSEQ     SEQUENCE(MAX,1)
)
```

Using `POSITION` Keyword

- For fixed positioning data, specify the range of columns (start with 1):

```
JOB_CODE POSITION(5:10)
```

- For relatively positioned columns based on delimiters, **`POSITION(*)`** is related to the current field. The second **`POSITION(*)`** is related to the next column, and so on. **`POSITION(n)`** corresponds to specific column

```
EMPNO POSITION (*) INTEGER
ENAME POSITION (*) CHAR
```

- Specific column order:

```
EMPNO POSITION (1) INTEGER
SALARY POSITION (*) CHAR ...
```

Specifying Filler Fields

- A Filler field is a field that does not correspond to a database column.

```
LOAD DATA
INFILE 'emp.dat'
INTO TABLE EMP
REPLACE
FIELDS TERMINATED BY ','
( EMPNO    INTEGER EXTERNAL,
  ENAME    CHAR,
  JOB      CHAR,
  HIRE_DATE DATE(21) 'DD-MM-YYYY HH24:MI:SS',
  SAL      DECIMAL EXTERNAL,
  RES_FILE FILLER CHAR,
  DEPTNO   INTEGER EXTERNAL,
)
```

SQL*Loader Loading Methods: Comparison

Conventional Load	**Direct Path Load**
Uses `COMMIT`	Uses data saves (faster operation)
Always generates redo entries	Generates redo only under specific conditions
Enforces all constraints	Enforces only `PRIMARY KEY`, `UNIQUE`, and `NOT NULL`
Fires `INSERT` triggers	Does not fire `INSERT` triggers
Can load into clustered tables	Does not load into clusters
Allows other users to modify tables during load operation	Prevents other users from making changes to tables during load operation
Maintains index entries on each insert	Merges new index entries at the end of the load

"External Tables" Loading Method

- The advantages of using **external table** loads over conventional path and direct path loads:
  - File can be loaded in parallel
  - Loaded data can be modified using SQL and PL/SQL functions
- An external table load creates an external table for data that is contained in an external data file.

SQL*Loader Case Studies

- Examples of using SQL*Loader in different scenarios
- You must install Oracle Database Examples (formerly Companion) media
- Case study files are installed in the directory **`$ORACLE_HOME/rdbms/demo`**
- File format names are **`ulcase`*`n`*`.sql`**, **`ulcase`*`n`*`.ctl`**, and **`ulcase`*`n`*`.dat`**
- SQL*Loader reference: "Oracle Database Utilities"

Summary

In this lecture, you should have learnt how to perform the following:

- Describe SQL*Loader target, components, and features
- Start SQL*Loader and use command-line parameters
- Create SQL*Loader control files
- Configure control files for different loading scenarios
- Set more control file configuration options
- Use multiple **`INTO TABLE`** clauses
- Specify the field list contents
- Use POSITION keyword
- Describe the differences between the SQL*Loader loading methods
- Install SQL*Loader Case Studies