DDL - Defining a Table

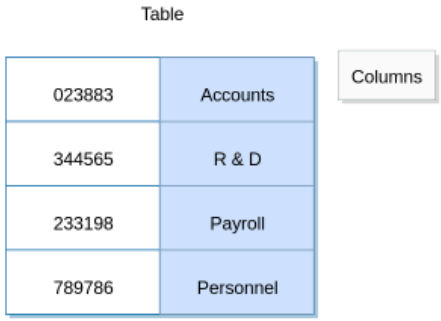
Objectives

DDL - Defining a Table

In this module you will examine the structure of the most important object in a Db2 system, which is a table. You will also look at how a table is defined.

After completing this module you will be able to:

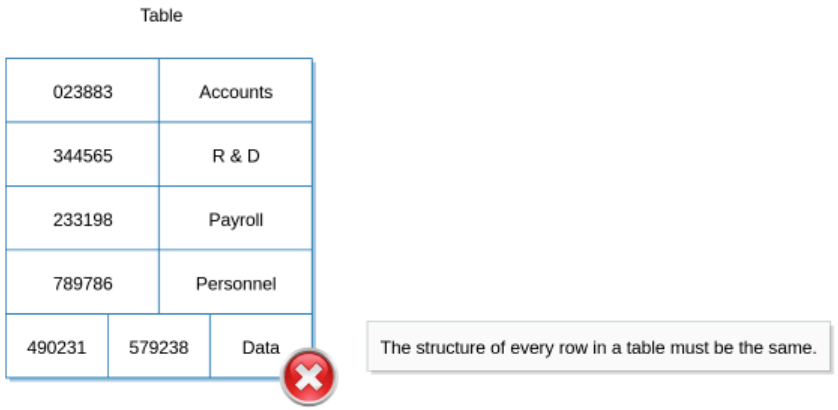
- Describe the Structure of a Table
- List Db2 Data Types
- Create a Table
- Create an Index
- Drop a Table
- Alter a Table



A table can be thought of as a flat file with rows instead of records and columns instead of fields.

Each column has a data type associated with it. The length of the column and the data type are part of the table definition.

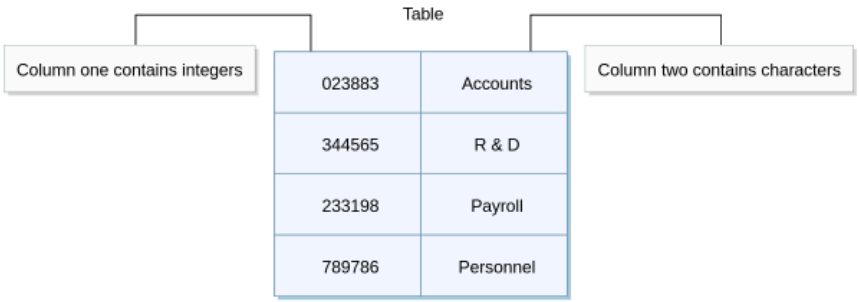
Click Play to see how a table is structured.



Every row in a table has the same structure as every other row, that is, they all have the same number of columns and columns in the same position have the same data type.

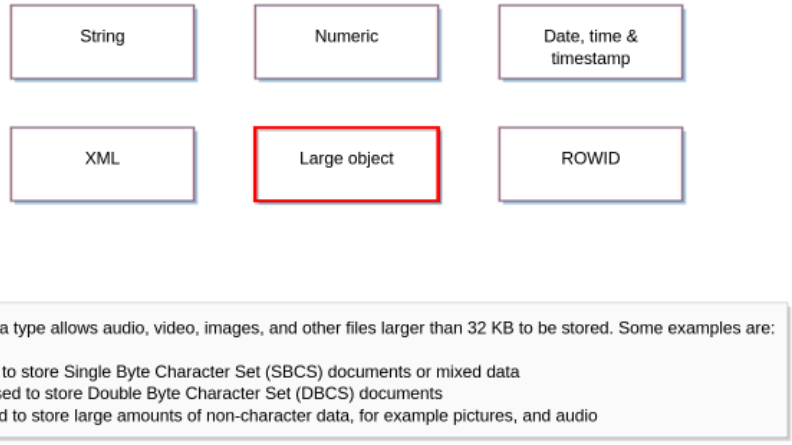
In fact, it is impossible to put rows with different columns in the same table because the column definitions, or data types, and the positions of the columns in the table are part of the table definition.

Click Play to see a demonstration of this.



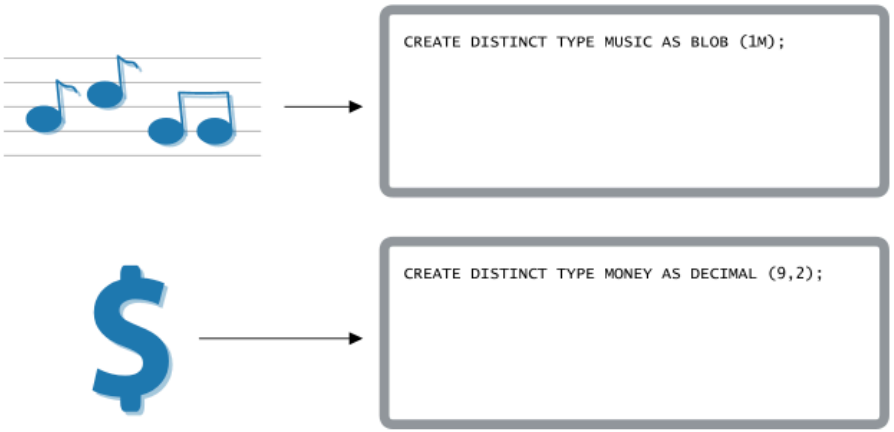
The data type of each column of every row is the same. Therefore, if the first column is defined as containing integers, the first column of every row must contain an integer.

As you will see, not all column data types are as exact as integer types. By using user-defined or binary object types it is possible to store very different data in the one column, but the data type defines how the column's data is constantly handled by Db2 and what Db2 operations can be performed on it.



Each column will have a built-in data type assigned to it, which dictates the data values that can be stored in it. The data type is defined at table creation but can be modified at a later stage.

Mouse-over the groups for examples of the data-types that can be applied to a column.



You are able to create your own data type, which is based on one of the existing Db2 built-in data types. The examples here show how user-defined distinct types, MUSIC and MONEY are created. Once this task is performed, this data type can be specified when creating a table.

Data type	COBOL	PL/I
CHAR(n)	PIC X(n)	CHAR(n)
SMALLINT	PIC S9(4) COMP	FIXED BIN (15)
INTEGER	PIC S9(9) COMP	FIXED BIN (31)
VARCHAR(n)	01 VAR-NAME. 49 VAR-LEN PIC S9(4) USAGE BINARY. 49 VAR-TEXT PIC X(n).	CHAR(n) VAR
DECIMAL(p,s)	PIC S9(p-s)V9(s) COMP-3	FIXED DEC(p,s)
DATE	PIC X(10)	CHAR (10)
TIME	PIC X(8)	CHAR(8)
TIMESTAMP	PIC X(26)	CHAR(26)

The Db2 data types are internal and when using Db2 data in a program you must use the language equivalent. Where there is no exact data type, a structure or character is used, for example, dates and times.

Name_ID	Official_Name	Last_Update

Customer

```
CREATE TABLE CUSTOMER
 (
 NAME_ID CHAR(8) NOT NULL,
 OFFICIAL_NAME CHAR(35) NOT NULL,
 LAST_UPDATE TIMESTAMP NOT NULL WITH DEFAULT );
```

You are now going to take a closer look at the CREATE TABLE statement.

This command will create the table shown above. It includes the CREATE statement, the object type of TABLE, and the object name of CUSTOMER.

Next you will look at the other components of the CREATE TABLE statement.

Name_ID	Official_Name	Last_Update

Customer

```
CREATE TABLE CUSTOMER
 (
 NAME_ID CHAR(8) NOT NULL,
 OFFICIAL_NAME CHAR(35) NOT NULL,
 LAST_UPDATE TIMESTAMP NOT NULL WITH DEFAULT );
```

When you create a table you must also define the columns that make up a table.

In this example there are column definitions for NAME_ID, OFFICIAL_NAME, and LAST_UPDATE.

Customer

```
CREATE TABLE CUSTOMER
 (
 NAME_ID CHAR(8) NOT NULL,
 OFFICIAL_NAME CHAR(35) NOT NULL,
 LAST_UPDATE TIMESTAMP NOT NULL WITH DEFAULT );
```

Each column name in a table contains attributes. In the example here, the NAME_ID column has a data type indicating a fixed length character string of 8 characters.

If you do not want the column to contain any null values, then the NOT NULL attribute must be coded.

Name_ID	Official_Name	Last_Update

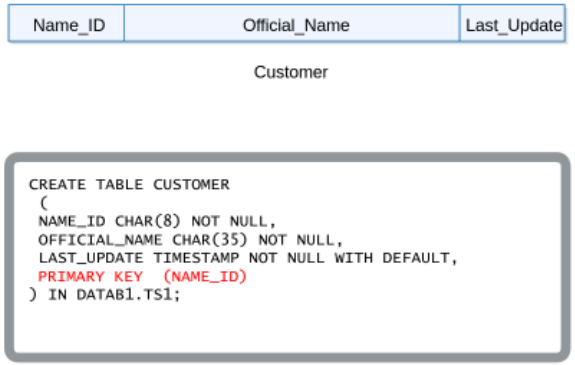
Customer

```
CREATE TABLE CUSTOMER
 (
 NAME_ID CHAR(8) NOT NULL
 OFFICIAL_NAME CHAR(35) NOT NULL
 LAST_UPDATE TIMESTAMP NOT NULL WITH DEFAULT );
```

There will be some situations during a load, insert or update of data into a table where a column value is not supplied. In this scenario, Db2 needs to know whether a default value should be inserted, and if so, what that value should be. Each data type has its own pre-defined default values.

With the LAST_UPDATE column shown here, a WITH DEFAULT attribute has been included. This means that if a value is not supplied for this column that the default for data type TIMESTAMP will be invoked (this is the current timestamp to a precision of 6 digits).

Note: You can specify an alternate default value for a table's column by adding that value after the WITH DEFAULT code. For example, WITH DEFAULT CURRENT TIMESTAMP(12), will result in a timestamp with a precision of 12 digits being used as the default.



Having defined all the columns, you can optionally define constraints such as a primary key for the table and the tablespace it will be stored in. The primary key is a unique key for the table which must have a corresponding index created (this is covered in the next section of this module).

Here you specify NAME_ID as your primary key with the statement:

```
PRIMARY KEY (NAME_ID)
```

Customer

```
CREATE TABLE CUSTOMER
 (
 NAME_ID CHAR(8) NOT NULL,
 OFFICIAL_NAME CHAR(35) NOT NULL,
 LAST_UPDATE TIMESTAMP NOT NULL WITH DEFAULT,
 PRIMARY KEY  (NAME_ID)
) IN DATAB1.TS1;
```

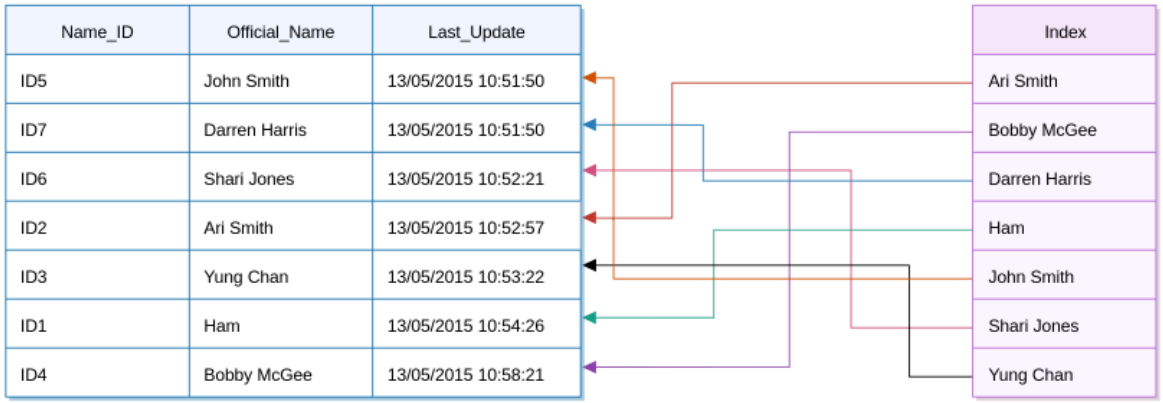
This example also specifies that this table will be stored in the TS1 tablespace of the DATAB1 database and not the default tablespace or container for the database.

Name_ID	Official_Name	Last_Update

Customer

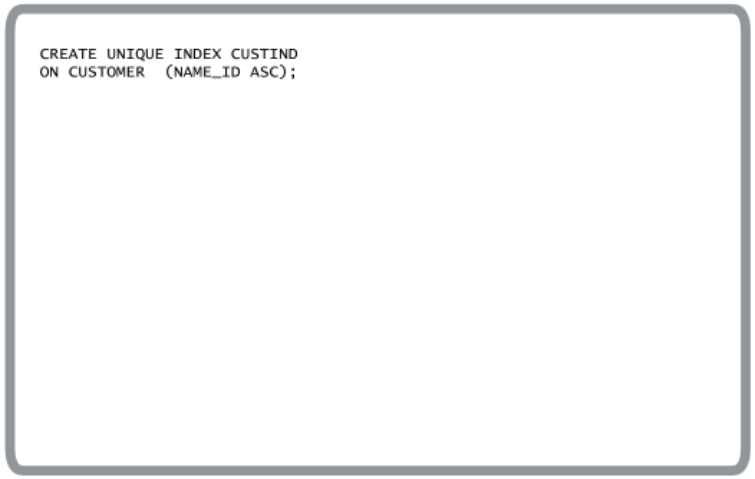
```
CREATE TABLE CUSTOMER
 (
 NAME_ID CHAR(8) NOT NULL,
 OFFICIAL_NAME CHAR(35) NOT NULL,
 LAST_UPDATE TIMESTAMP NOT NULL WITH DEFAULT,
 PRIMARY KEY  (NAME_ID)
) IN DATAB1.TS1;
```

You have now created a table called CUSTOMER. If this is defined on the mainframe the definition is incomplete, as it requires an index on the primary key to be created. While it is incomplete, the only operations allowed on the table are DROP or CREATE to create the primary index.



Having defined a primary key for a table, you must also create an index for that key. Additionally you could create indexes on other columns to help Db2 look up values in the columns. The index contains pointers to the location of that row of data within the table.

Generally indexes improve SELECT performance but degrade INSERT and UPDATE. You should consider table access and usage when deciding what indexes to build. In many cases this is a job for the DBA.



The basic syntax to create an index is shown here. This example is creating a unique index named CUSTIND (the attributes of a unique index are discussed on the following pages).

The second line of the code identifies the table that is being referenced (CUSTOMER) and the column within that table that will be used for indexing (NAME_ID). The last part of the code indicates how the index is sorted. ASC indicates ascending order, DESC is descending order, while RANDOM is used when index entries are put in random order by the column..

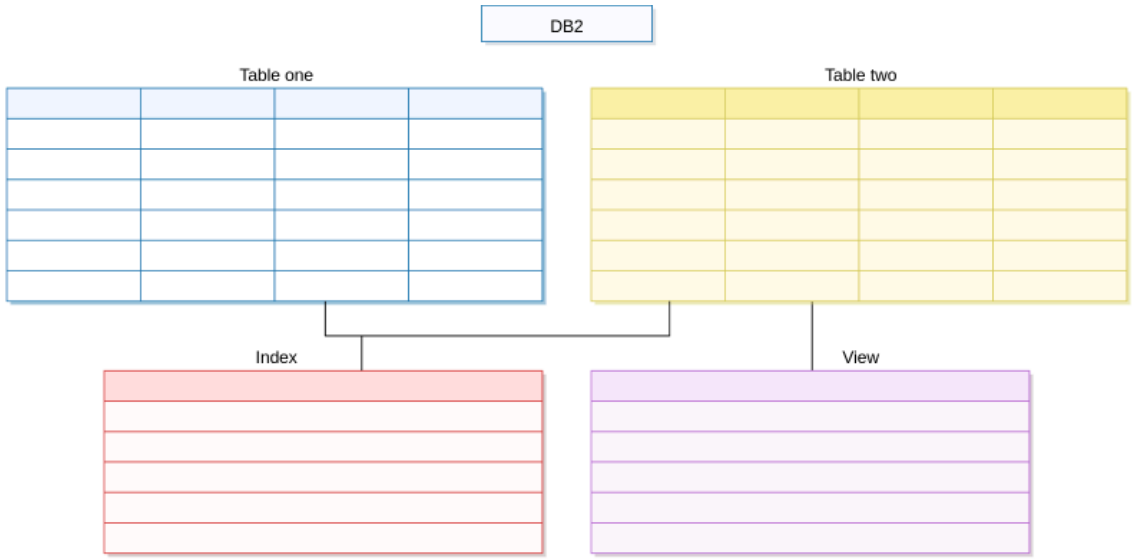
Name_ID	Official_Name	Last_Update
ID5	John Smith	13/05/2015 10:51:50
ID7	Darren Harris	13/05/2015 10:51:50
ID6	Shari Jones	13/05/2015 10:52:21
ID2	Ari Smith	13/05/2015 10:52:57
ID3	Yung Chan	13/05/2015 10:53:22
ID1	Ham	13/05/2015 10:54:26
ID4	Bobby McGee	13/05/2015 10:58:21

Unique index
ID1
ID2
ID3
ID4
ID5
ID6
ID7

Non-unique index
Ari Smith
Bobby McGee
Darren Harris
Ham
John Smith
John Smith
Shari Jones
Yung Chan

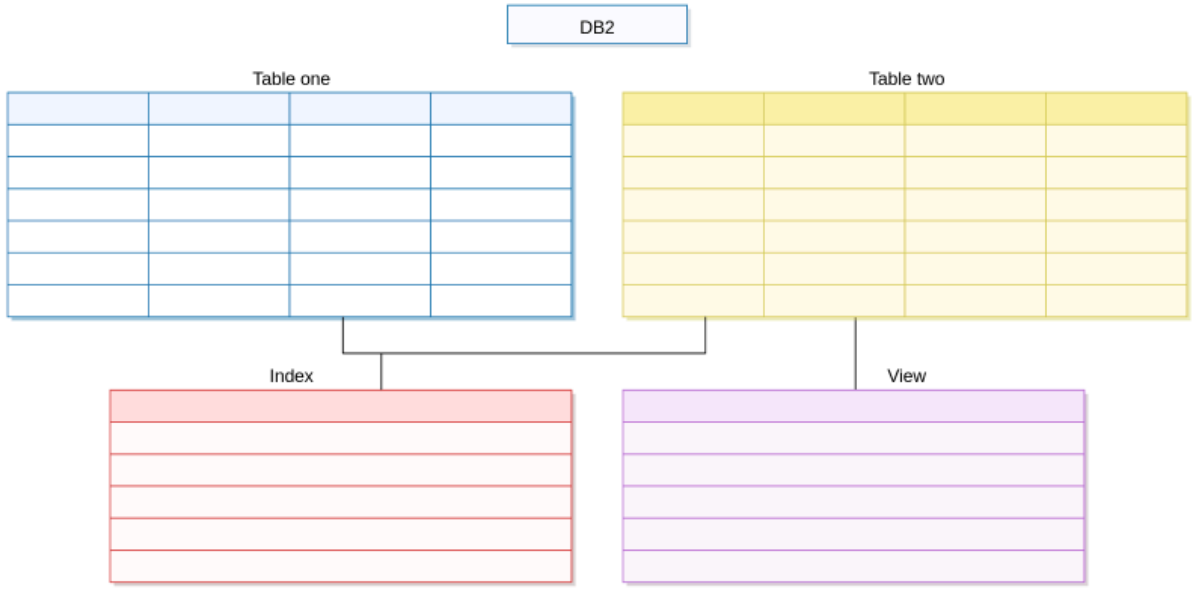
As the name suggests, a unique index ensures that there are no identical key values stored in the table for a specific column. In the example here, the Name_ID column is a good candidate for a unique index as no two employees should have the same ID. If an attempt is made to insert a record that contains an existing Name-ID value, the action will fail.

Non-unique indexes can also be useful depending on the information that needs to be referenced in the table. In this example, the Official_Name column has been used to create a non-unique index. It is non-unique because there could be two or more people with the same name. In this scenario it would allow another John Smith record to appear in the table.



The DROP statement can be used to remove an object from the Db2 system.

Except for storage groups, any other Db2 objects that are reliant on the dropped object will also be dropped. In this example, if Table two is dropped, then the Index associated with it will be dropped as well as the View that used part of its data.



When you drop a table or another object, all data contained in the object is deleted as well as the object definition.

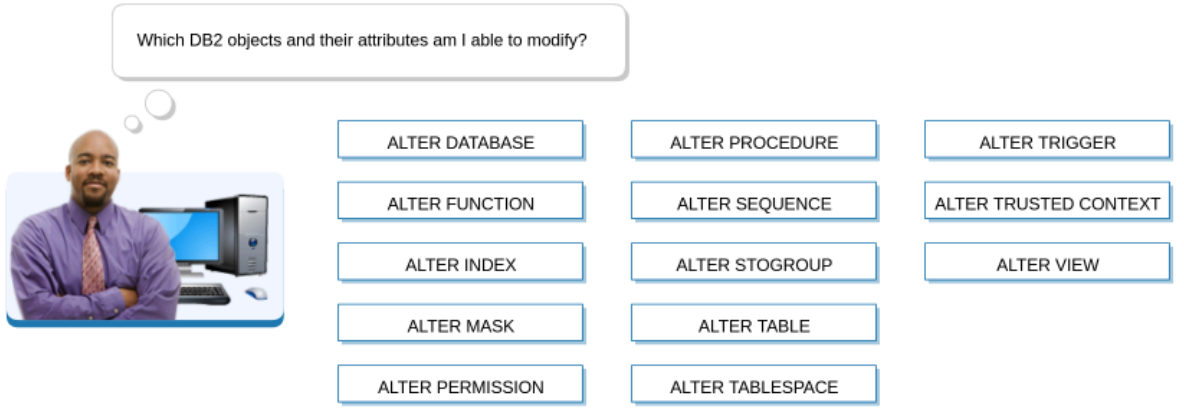
If your aim is to just remove the data from the table, then you could use the:
- DELETE FROM statement
- TRUNCATE statement

EMP_ID	LAST_NAME	LAST_UPDATE
ID5	John Smith	13/05/2015 10:51:50
ID7	Darren Harris	13/05/2015 10:51:50
ID6	Shari Jones	13/05/2015 10:52:21
ID2	Ari Smith	13/05/2015 10:52:57
ID3	Yung Chan	13/05/2015 10:53:22
ID1	Ham	13/05/2015 10:54:26
ID4	Bobby McGee	13/05/2015 10:58:21

```
CREATE TABLE EMPLOYEE
 (
  EMP_ID       VARCHAR(5) NOT NULL,
  LAST_NAME     VARCHAR(25) NOT NULL,
  LAST_UPDATE TIMESTAMP NOT NULL WITH DEFAULT
  )
  WITH RESTRICT ON DROP;
```

To avoid dropping important tables accidentally, you can specify a restriction on the table. This means that you cannot drop the table or the tablespace that contains that table.

Even with this restriction the table can still be dropped:
- using the REPAIR DBD DROP utility
- Using the ALTER TABLE statement to remove the RESTRICT clause, which will then allow the table to be dropped



After using your new database for some time, you may have determined that certain structural aspects and attributes need to be improved or modified. In this scenario the ALTER statement can be used to change the definitions of Db2 objects.

In this section you will see how the ALTER statement is used to modify TABLE attributes.

Name_ID	Official_Name	Last_Update
ID5	John Smith	13/05/2015 10:51:50
ID7	Darren Harris	13/05/2015 10:51:50
ID6	Shari Jones	13/05/2015 10:52:21
ID2	Ari Smith	13/05/2015 10:52:57
ID3	Yung Chan	13/05/2015 10:53:22
ID1	Ham	13/05/2015 10:54:26
ID4	Bobby McGee	13/05/2015 10:58:21

Columns

Can be added, renamed or dropped. Some data types can be changed.

Partitions

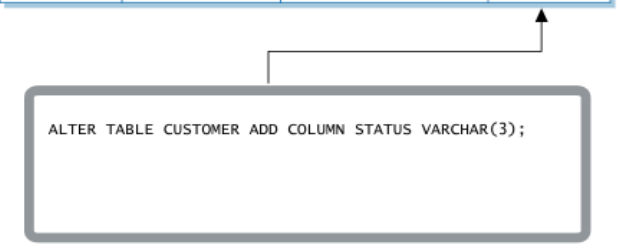
A new partition to a table space can be added, and boundaries between partitions can modified.

Keys

Primary and foreign keys can be added and dropped.

The ALTER TABLE statement can be used to add, modify and remove components and attributes associated with a table. Some of the items that can be altered appear above, and will be discussed on the following pages.

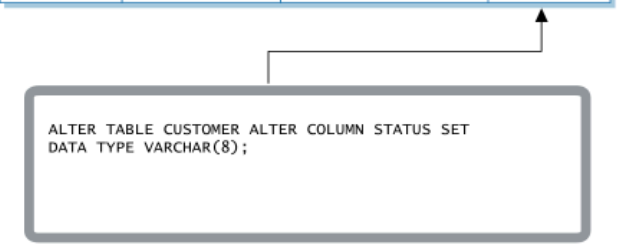
Name_ID	Official_Name	Last_Update	STATUS
ID5	John Smith	13/05/2015 10:51:50	
ID7	Darren Harris	13/05/2015 10:51:50	
ID6	Shari Jones	13/05/2015 10:52:21	
ID2	Ari Smith	13/05/2015 10:52:57	
ID3	Yung Chan	13/05/2015 10:53:22	
ID1	Ham	13/05/2015 10:54:26	
ID4	Bobby McGee	13/05/2015 10:58:21	



In the example shown here a new column is being added to the CUSTOMER table. The VARCHAR data type is specified for the new STATUS column, indicating that any values stored in it must be characters strings of variable length with a maximum length of three bytes.

Once created, the default value associated with that data type (VARCHAR) will be used to populate that column.

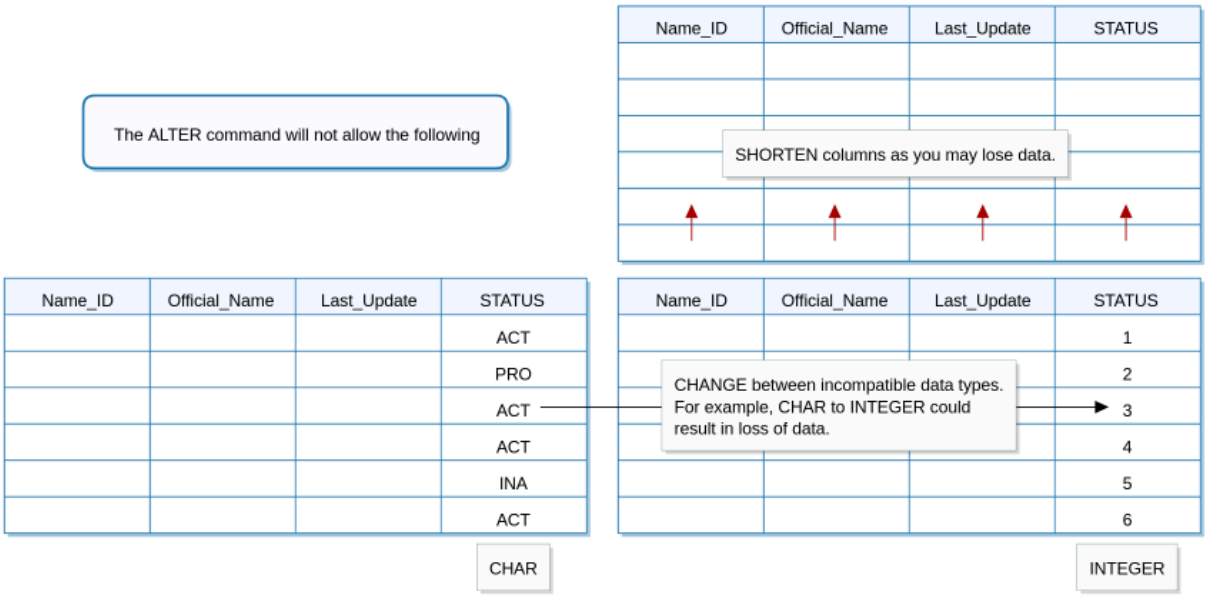
Name_ID	Official_Name	Last_Update	STATUS
ID5	John Smith	13/05/2015 10:51:50	ACT
ID7	Darren Harris	13/05/2015 10:51:50	PRO
ID6	Shari Jones	13/05/2015 10:52:21	ACT
ID2	Ari Smith	13/05/2015 10:52:57	ACT
ID3	Yung Chan	13/05/2015 10:53:22	INA
ID1	Ham	13/05/2015 10:54:26	ACT
ID4	Bobby McGee	13/05/2015 10:58:21	LOA



Attributes of existing table columns can also be modified using the ALTER TABLE statement.

In this example, the VARCHAR data type has been retained for the STATUS column, but the maximum amount of bytes of data that can be stored in this column has increased from three to eight.

Note that only future values of the column are affected by the changes made using this statement. So, after this statement is invoked, a new record with a STATUS value of ACTPART or ACTFULL could be inserted.



The ALTER TABLE statement is not able to perform some actions because they may result in errors or loss of integrity. An example of this is modifying a data type of a column from character to an integer. Shrinking or expanding the size of a column can also have dire consequences depending on the current values in that column and the overall maximum length allowed for the row.

Summary

DDL - Defining a Table

In this module you examined the structure of the most important object in a Db2 system - a table. You also looked at how to define it.

You should now be able to:

- Describe the Structure of a Table
- List Db2 Data Types
- Create a Table
- Create an Index
- Drop a Table
- Alter a Table

Module Complete

DDL - Defining a Table

You can **click** **Exit** to leave DDL - Defining a Table and record your results, or use the Table of Contents to go back and review the module.

To take the DDL - Defining a Table module test again you will need to exit the module and then retake from your LMS.

Note: To ensure your results are stored correctly, use the '**Exit**' button located in the top right corner of this training window. **Do NOT** use the '**X**' in the top right of your browser.