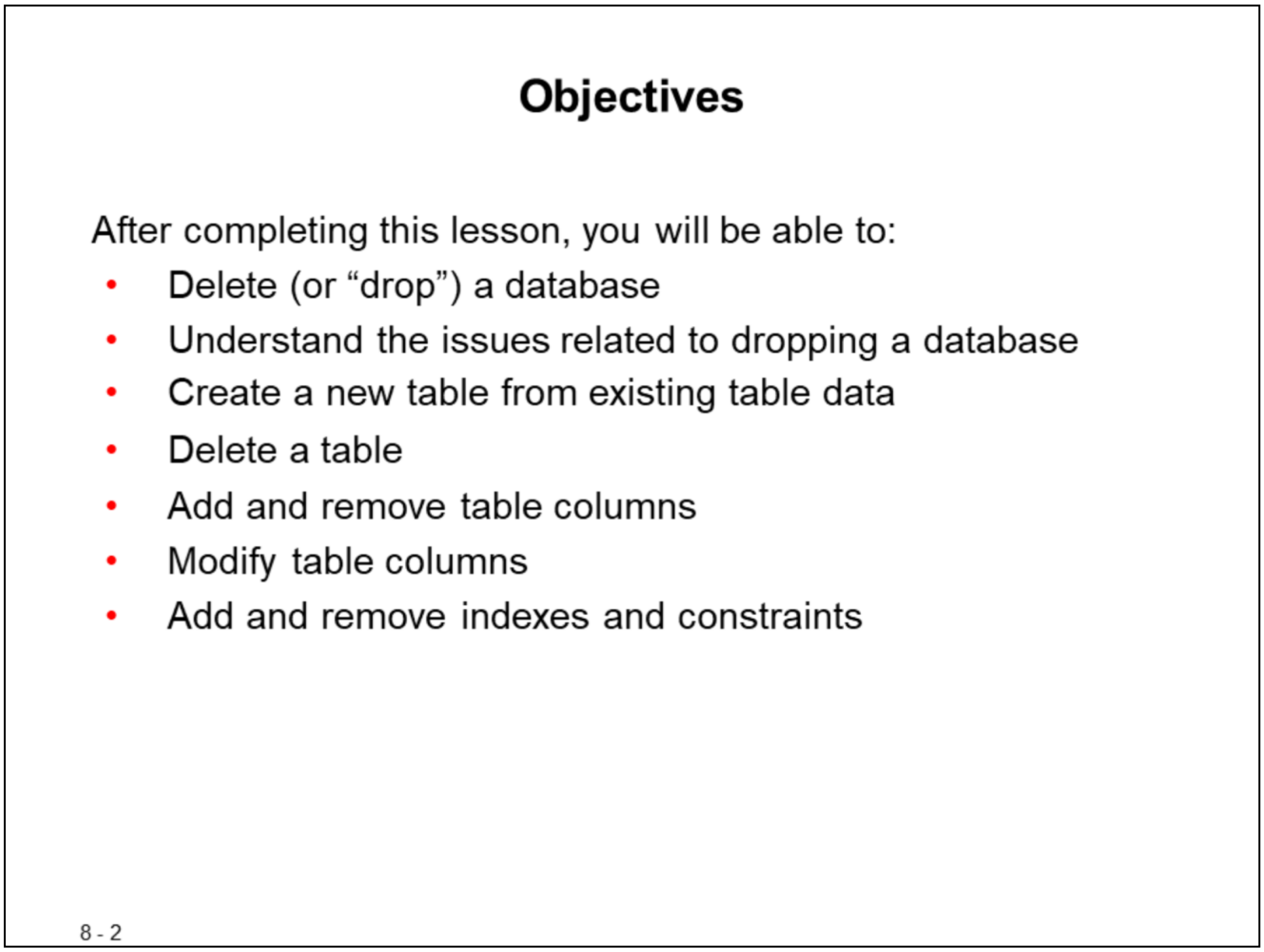
8 Database and Table Maintenance

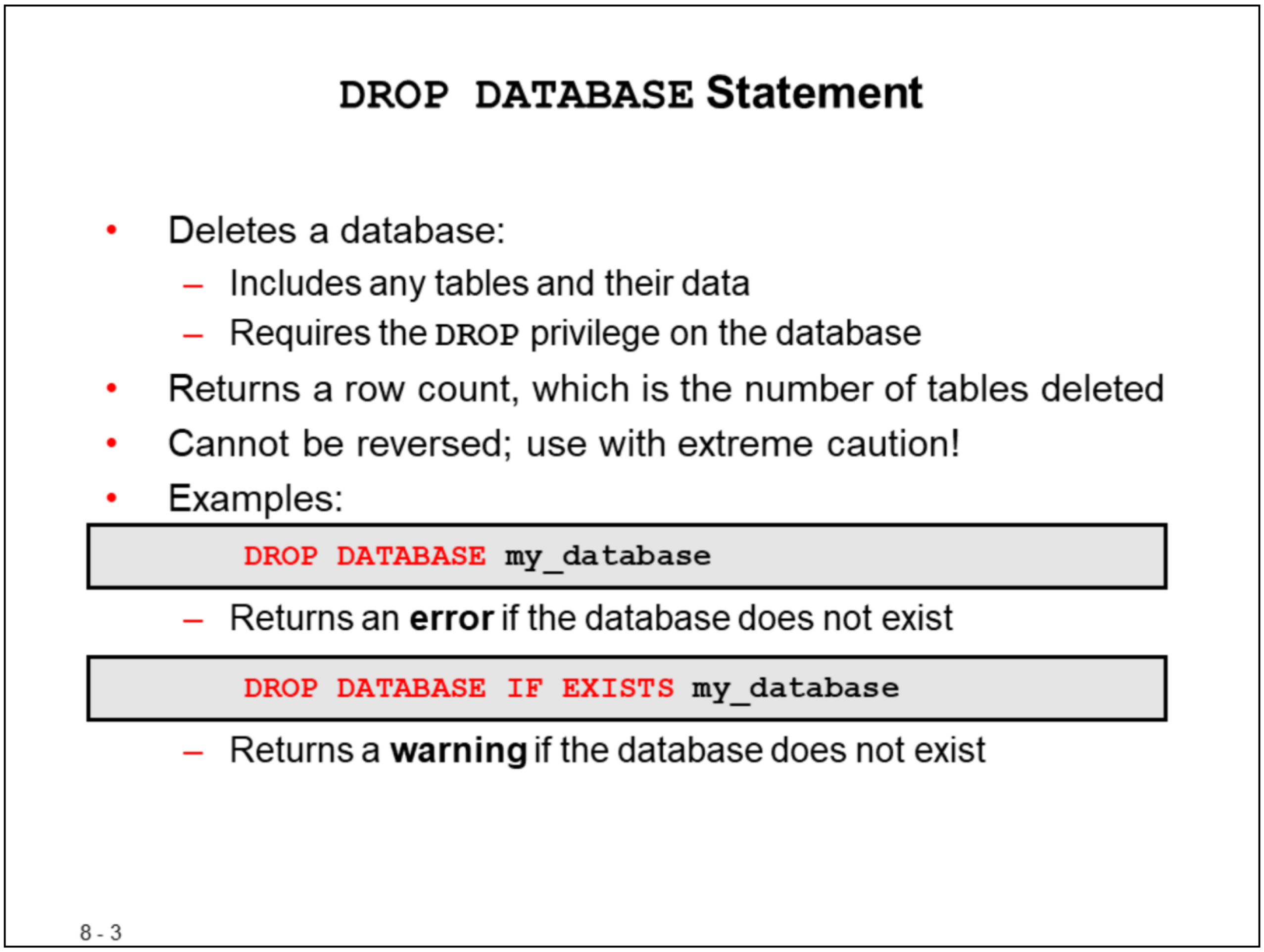


Objectives

After completing this lesson, you will be able to:

- Delete (or “drop”) a database
- Understand the issues related to dropping a database
- Create a new table from existing table data
- Delete a table
- Add and remove table columns
- Modify table columns
- Add and remove indexes and constraints

8 - 2



DROP DATABASE Statement

- Deletes a database:
  - Includes any tables and their data
  - Requires the DROP privilege on the database
- Returns a row count, which is the number of tables deleted
- Cannot be reversed; use with extreme caution!
- Examples:

```
DROP DATABASE my_database
```

  - Returns an **error** if the database does not exist

```
DROP DATABASE IF EXISTS my_database
```

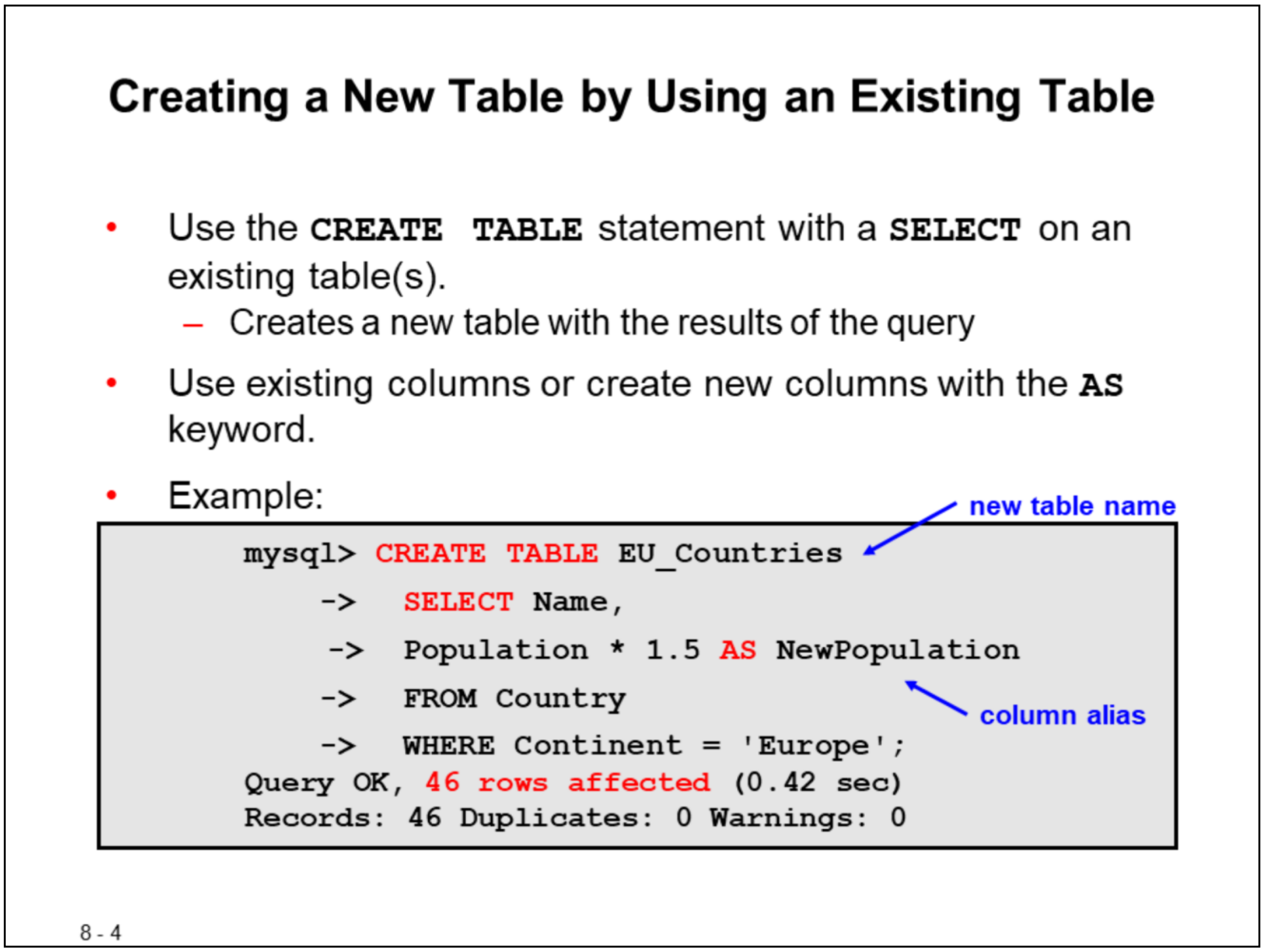
  - Returns a **warning** if the database does not exist

Before dropping the database, MySQL removes any objects that it contains, such as tables, stored routines, and triggers.

A successful DROP DATABASE returns a row count that indicates the number of tables dropped. (This is actually the number of .frm files removed, which amounts to the same thing.) Issue a SHOW DATABASES statement after the drop to confirm the deletion.

The host file system stores the database in its own directory under the data directory. When you drop the database, the server deletes only the files and directories it created and leaves others intact. If you or another process created files in the database directory, the server cannot delete those files and they prevent the server from deleting the directory. As such, the database still shows up in the results of a SHOW DATABASES statement. You need to remove the database directory and any remaining files manually.

Use the SHOW WARNINGS statement to display warnings generated by DROP DATABASE.



Creating a New Table by Using an Existing Table

- Use the **CREATE TABLE** statement with a **SELECT** on an existing table(s).
  - Creates a new table with the results of the query
- Use existing columns or create new columns with the **AS** keyword.
- Example:

new table name

```
mysql> CREATE TABLE EU_Countries
    ->   SELECT Name,
    ->   Population * 1.5 AS NewPopulation
    ->   FROM Country
    ->   WHERE Continent = 'Europe';
Query OK, 46 rows affected (0.42 sec)
Records: 46 Duplicates: 0 Warnings: 0
```

column alias

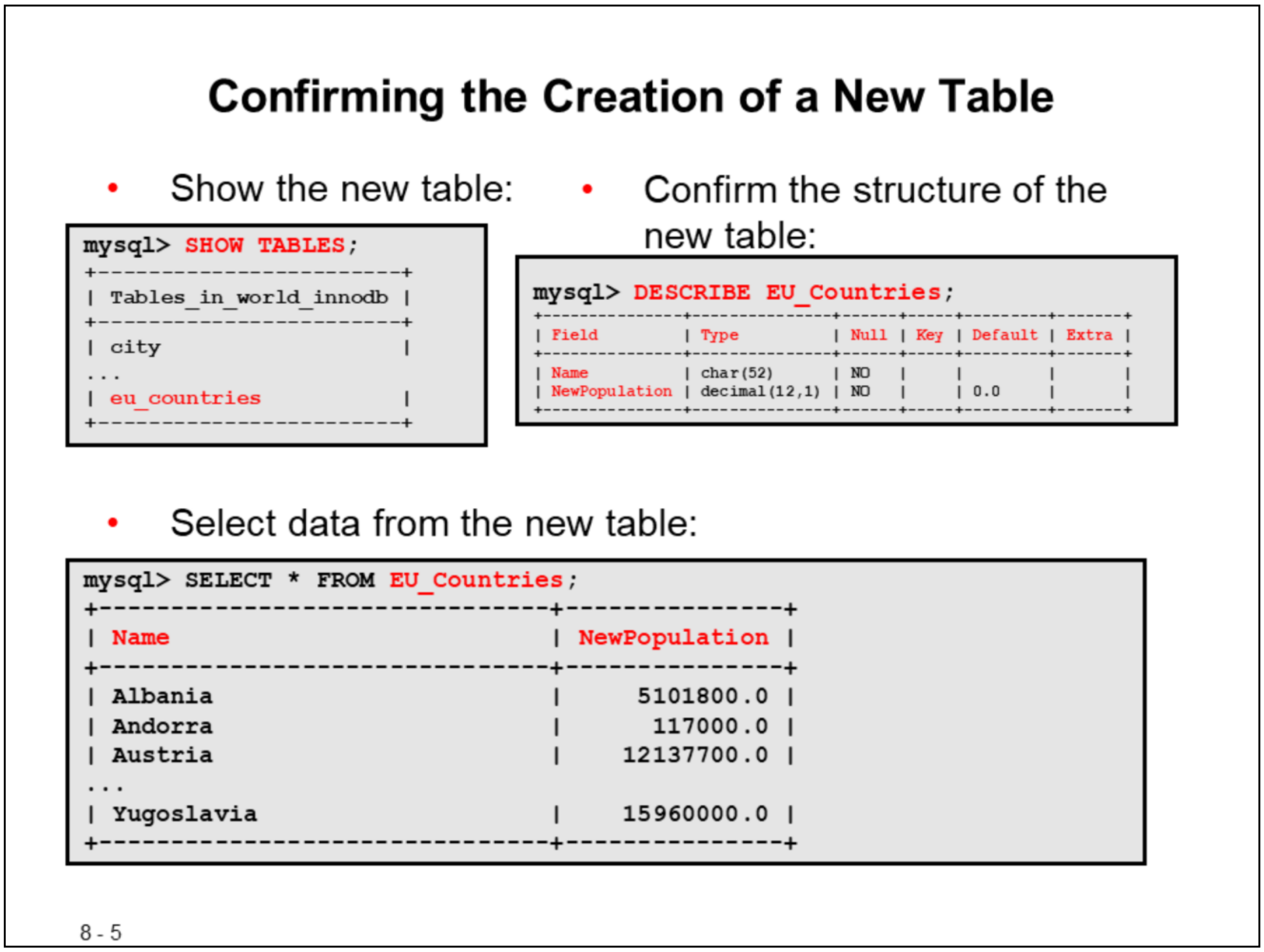
8 - 4

The SELECT statement can reference multiple tables with joins, subqueries, or unions. Or you can use a SELECT statement without any table.

The example shown creates a new table called EU_Countries from a SELECT query on the Country table. The new table has two columns: The Name column data from the Country table and a new column called NewPopulation. The NewPopulation column derives its values from an expression that uses the Country table's Population column data. The WHERE clause limits the source data to European countries only.

Note: The tables created with CREATE TABLE...SELECT are based solely on the output of the SELECT. They do not include table options, such as indexes and constraints. Also, the newly created tables do not always contain the data types you expect, particularly if the column is created based on an expression.

For more information about using CREATE TABLE with SELECT, see the MySQL Reference Manual at: http://dev.mysql.com/doc/refman/5.6/en/create-table-select.html.



Confirming the Creation of a New Table

- Show the new table:

```
mysql> SHOW TABLES;
+------------------------+
| Tables_in_world_innodb |
+------------------------+
| city                   |
...
| eu_countries           |
+------------------------+
```

- Confirm the structure of the new table:

```
mysql> DESCRIBE EU_Countries;
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| Name          | char(52)      | NO   |     |         |       |
| NewPopulation | decimal(12,1) | NO   |     | 0.0     |       |
+---------------+---------------+------+-----+---------+-------+
```

- Select data from the new table:

```
mysql> SELECT * FROM EU_Countries;
+--------------------------------+---------------+
| Name                           | NewPopulation |
+--------------------------------+---------------+
| Albania                        |     5101800.0 |
| Andorra                        |      117000.0 |
| Austria                        |    12137700.0 |
...
| Yugoslavia                     |    15960000.0 |
+--------------------------------+---------------+
```

8 - 5

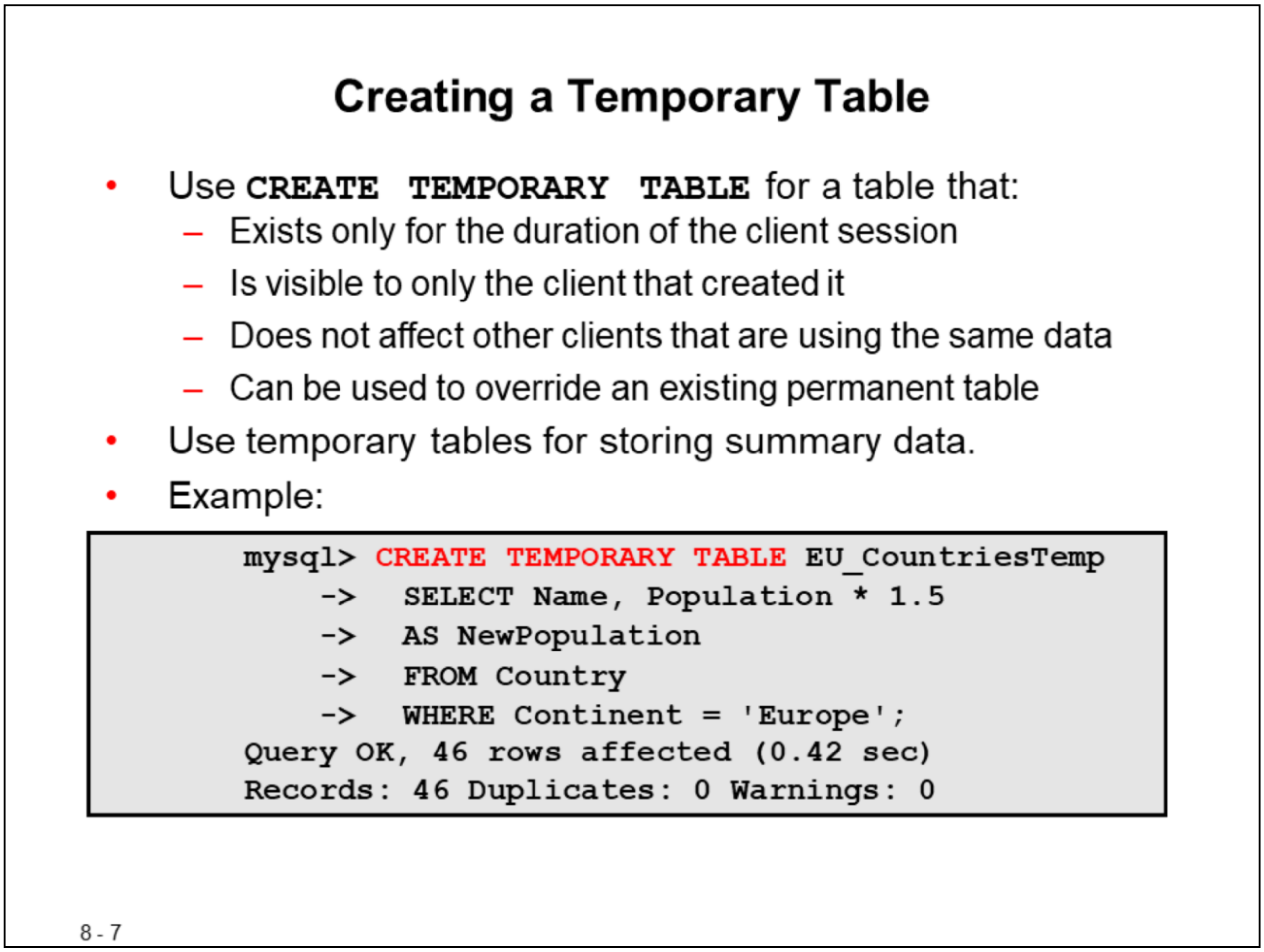
In the slide example, the `SHOW TABLES`, `DESCRIBE`, and `SELECT` statements confirm the creation of the new `EU_Countries` table and its contents.

Copying an Existing Table Structure

- Use **`CREATE TABLE`** with the **`LIKE`** keyword to create a table with the same structure as another table:
  - Indexes
  - Column options
- Only creates the table structure
  - Does not copy any data
- Example:

```
mysql> CREATE TABLE NewCity LIKE City;
Query OK, 0 rows affected (0.09 sec)
mysql> SELECT * FROM NewCity;
Empty set (0.00 sec)
```

8 - 6



Creating a Temporary Table

- Use **CREATE TEMPORARY TABLE** for a table that:
  - Exists only for the duration of the client session
  - Is visible to only the client that created it
  - Does not affect other clients that are using the same data
  - Can be used to override an existing permanent table
- Use temporary tables for storing summary data.
- Example:

```
mysql> CREATE TEMPORARY TABLE EU_CountriesTemp
    ->   SELECT Name, Population * 1.5
    ->   AS NewPopulation
    ->   FROM Country
    ->   WHERE Continent = 'Europe';
Query OK, 46 rows affected (0.42 sec)
Records: 46 Duplicates: 0 Warnings: 0
```

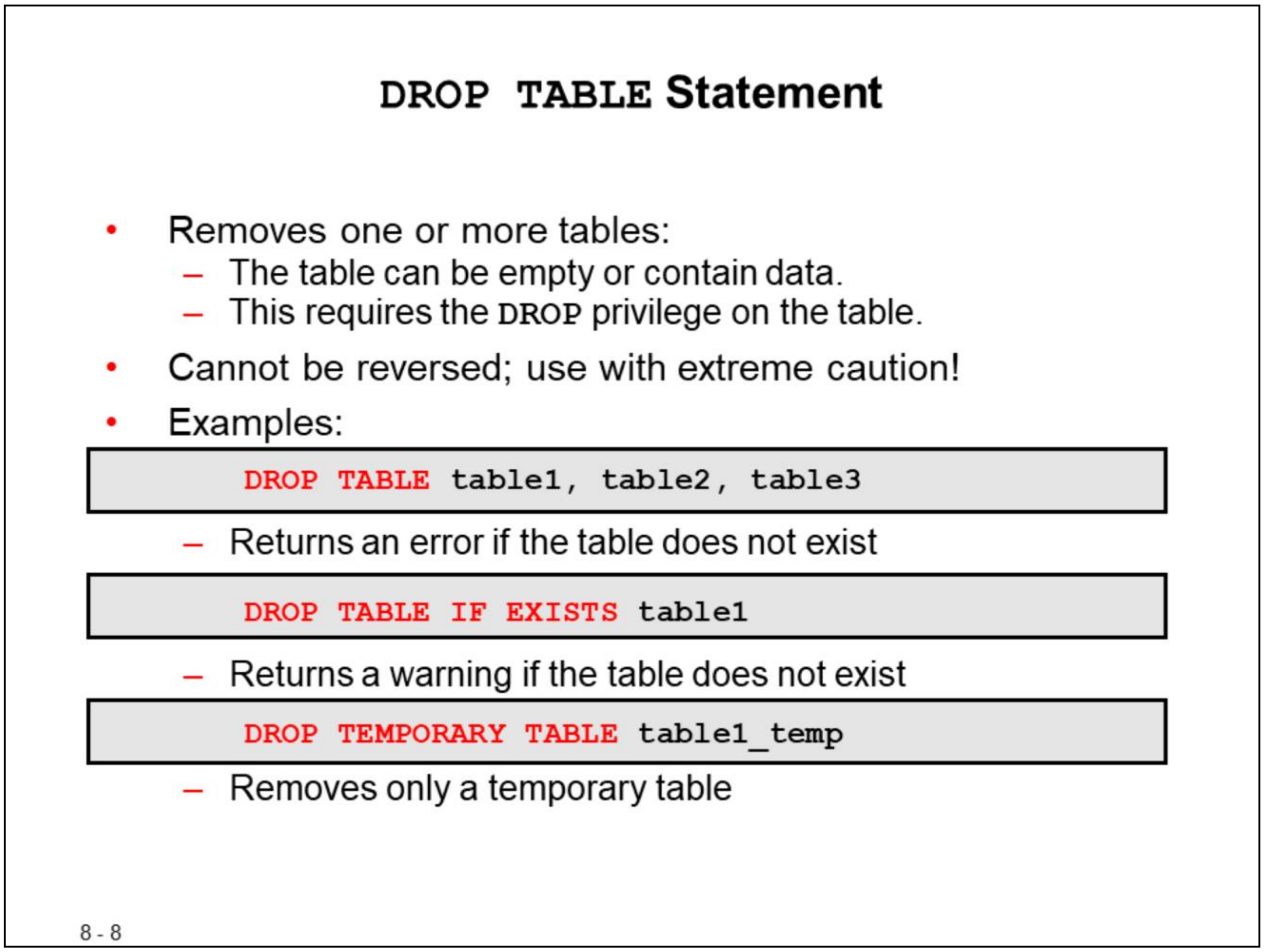
8 - 7

If you create a temporary table with the same name as an existing permanent table, the temporary table overrides the existing permanent table within the client session. This lasts until you drop the temporary table or disconnect the client session.

You can examine the table created by the statement in the slide example by using the `DESCRIBE` statement:

```
mysql> DESC EU_CountriesTemp;
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| Name          | char(52)      | NO   |     |         |       |
| NewPopulation | decimal(12,1) | NO   |     | 0.0     |       |
+---------------+---------------+------+-----+---------+-------+
2 rows in set (0.08 sec)
```

Use `CREATE TEMPORARY TABLE … LIKE` to create a temporary table with the same structure as an existing table.



DROP TABLE Statement

- Removes one or more tables:
  - The table can be empty or contain data.
  - This requires the DROP privilege on the table.
- Cannot be reversed; use with extreme caution!
- Examples:

```
DROP TABLE table1, table2, table3
```

  - Returns an error if the table does not exist

```
DROP TABLE IF EXISTS table1
```

  - Returns a warning if the table does not exist

```
DROP TEMPORARY TABLE table1_temp
```

  - Removes only a temporary table

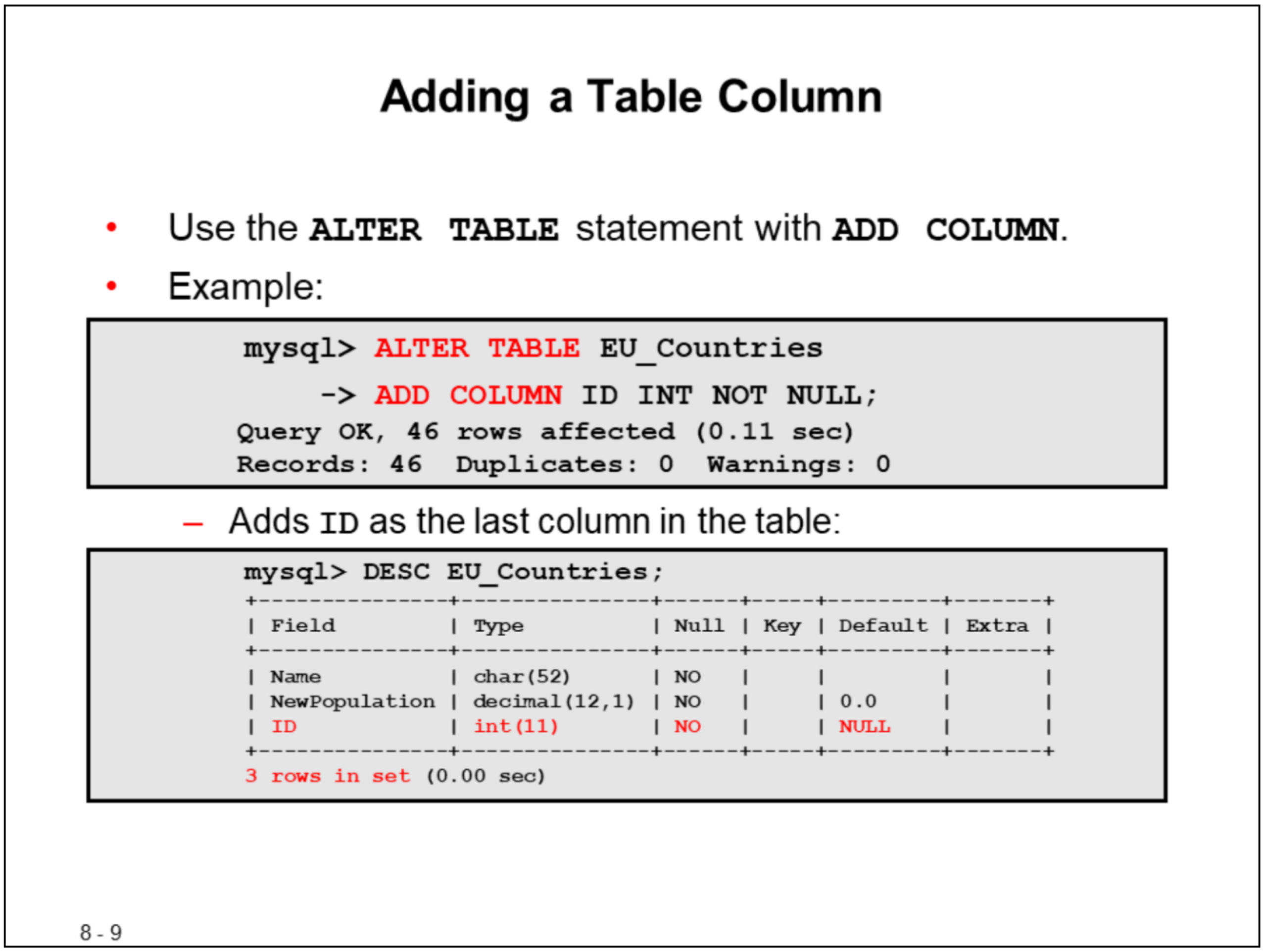
8 - 8

Use `IF EXISTS` to prevent an error if you drop tables that do not exist. This generates a warning instead which can be displayed with the `SHOW WARNINGS` statement.

Using `DROP TABLE` with the `TEMPORARY` keyword:

- Drops only temporary tables
- Does not end an ongoing transaction. (Transactions are covered later in the course.)
- Does not check access rights. (A `temporary` table is visible only to the client that created it, so no check is necessary.)

Use `DROP TEMPORARY TABLE` to ensure that you do not accidentally drop a permanent table with the same name.



Adding a Table Column

- Use the **ALTER TABLE** statement with **ADD COLUMN**.
- Example:

```
mysql> ALTER TABLE EU_Countries
    -> ADD COLUMN ID INT NOT NULL;
Query OK, 46 rows affected (0.11 sec)
Records: 46  Duplicates: 0  Warnings: 0
```

– Adds ID as the last column in the table:

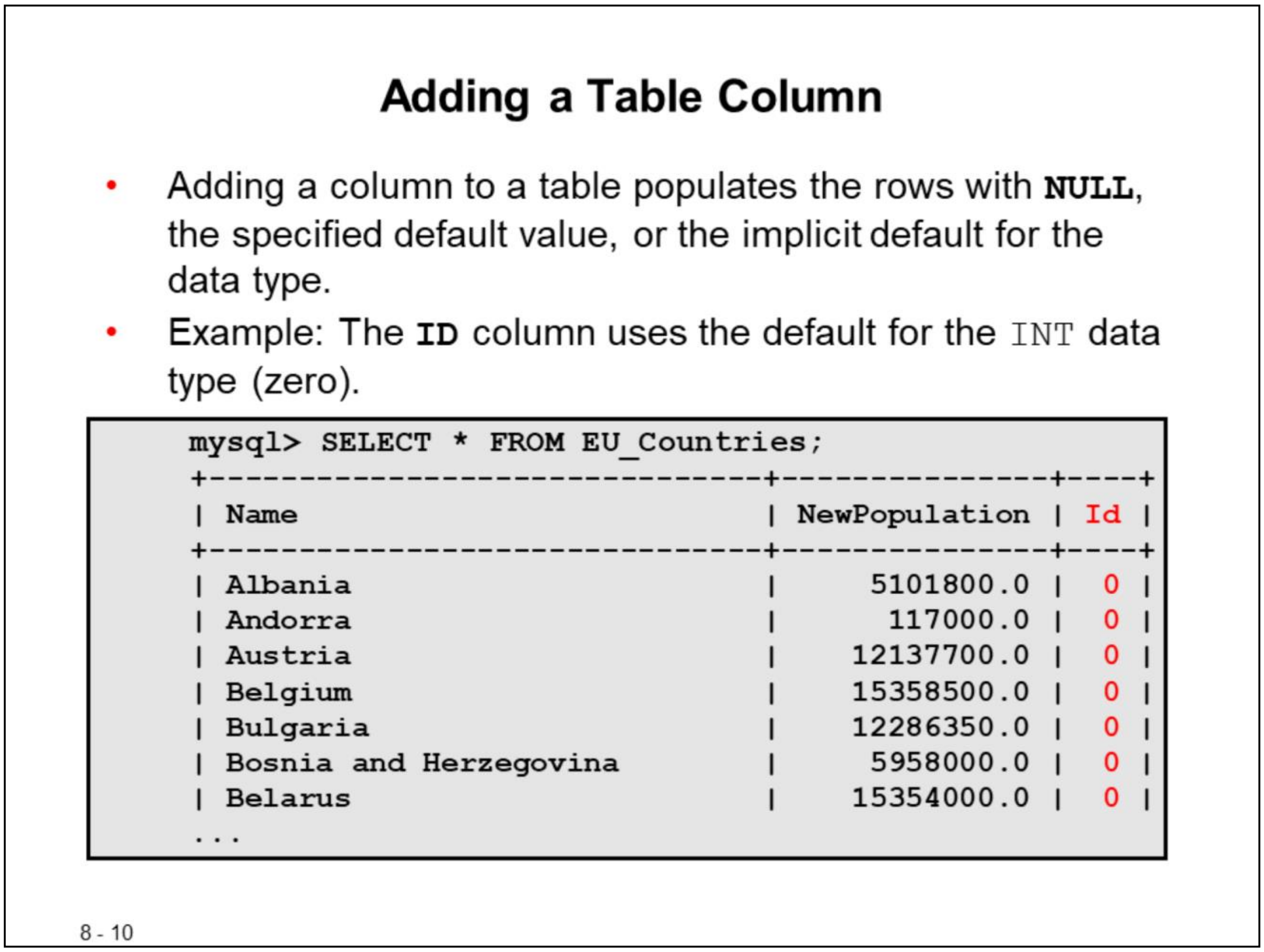
```
mysql> DESC EU_Countries;
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| Name          | char(52)      | NO   |     |         |       |
| NewPopulation | decimal(12,1) | NO   |     | 0.0     |       |
| ID            | int(11)       | NO   |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

8 - 9

Use ALTER TABLE … ADD COLUMN with the appropriate clause that specifies the column's definition. Use the same ADD COLUMN syntax as you would for CREATE TABLE. You can use ALTER TABLE to:

- Add or remove a column
- Add or remove an index
- Change an existing column's definition

Column names within a table must be unique, so you cannot add a column if one of the same name already exists in the table. Also, column names are not case-sensitive, so if the table already contains a column named ID, you cannot add a new column using any of these names: ID, id, Id, or iD. They are all considered to be the same.



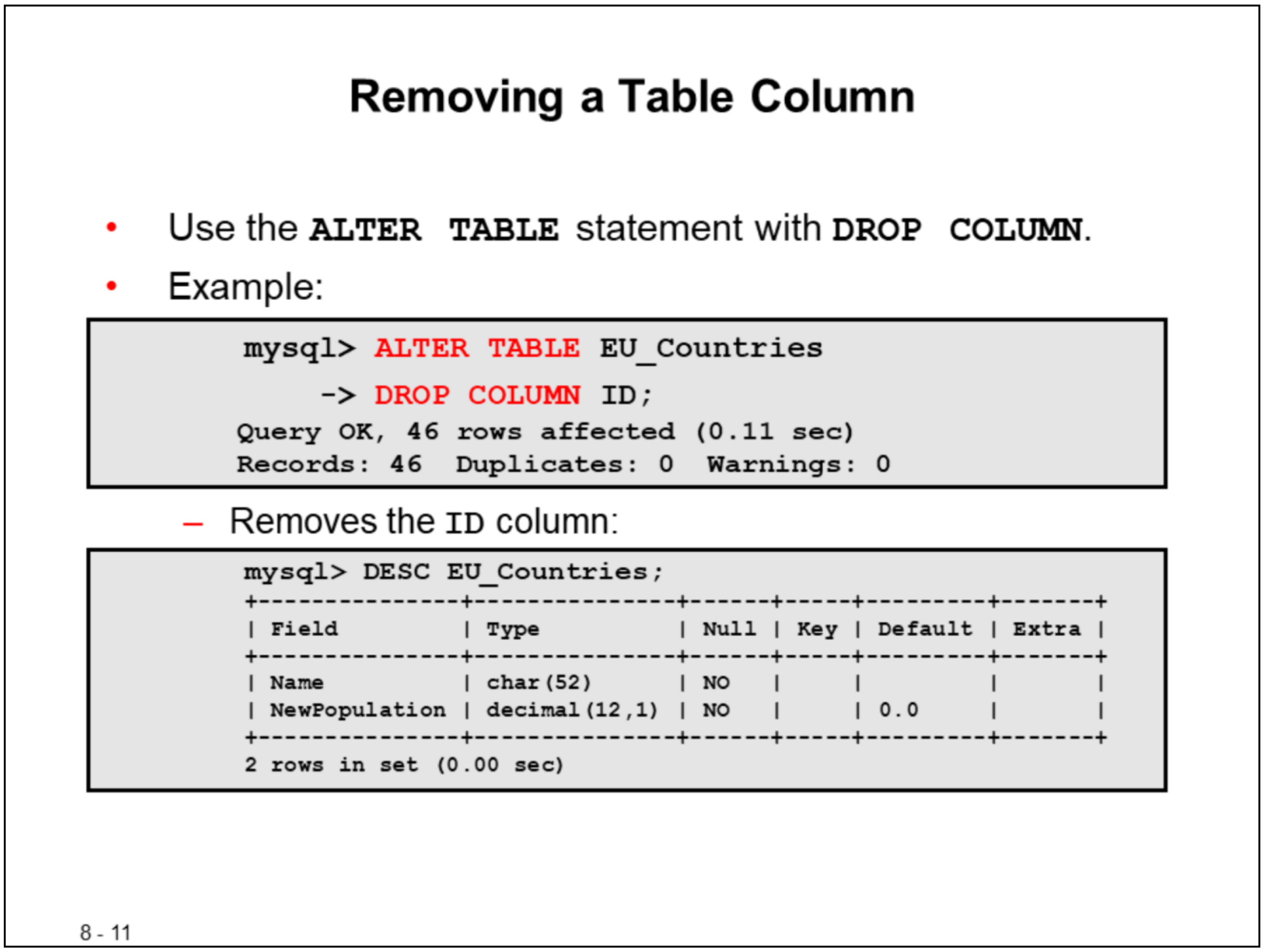
Adding a Table Column

- Adding a column to a table populates the rows with **NULL**, the specified default value, or the implicit default for the data type.
- Example: The **ID** column uses the default for the INT data type (zero).

```
mysql> SELECT * FROM EU_Countries;
+------------------------------+---------------+----+
| Name                         | NewPopulation | Id |
+------------------------------+---------------+----+
| Albania                      |     5101800.0 |  0 |
| Andorra                      |      117000.0 |  0 |
| Austria                      |    12137700.0 |  0 |
| Belgium                      |    15358500.0 |  0 |
| Bulgaria                     |    12286350.0 |  0 |
| Bosnia and Herzegovina       |     5958000.0 |  0 |
| Belarus                      |    15354000.0 |  0 |
...
```

8 - 10

Note: The statement for populating a table with data is covered in detail later in the course.



Removing a Table Column

- Use the **ALTER TABLE** statement with **DROP COLUMN**.
- Example:

```
     mysql> ALTER TABLE EU_Countries
         -> DROP COLUMN ID;
    Query OK, 46 rows affected (0.11 sec)
    Records: 46  Duplicates: 0  Warnings: 0
```

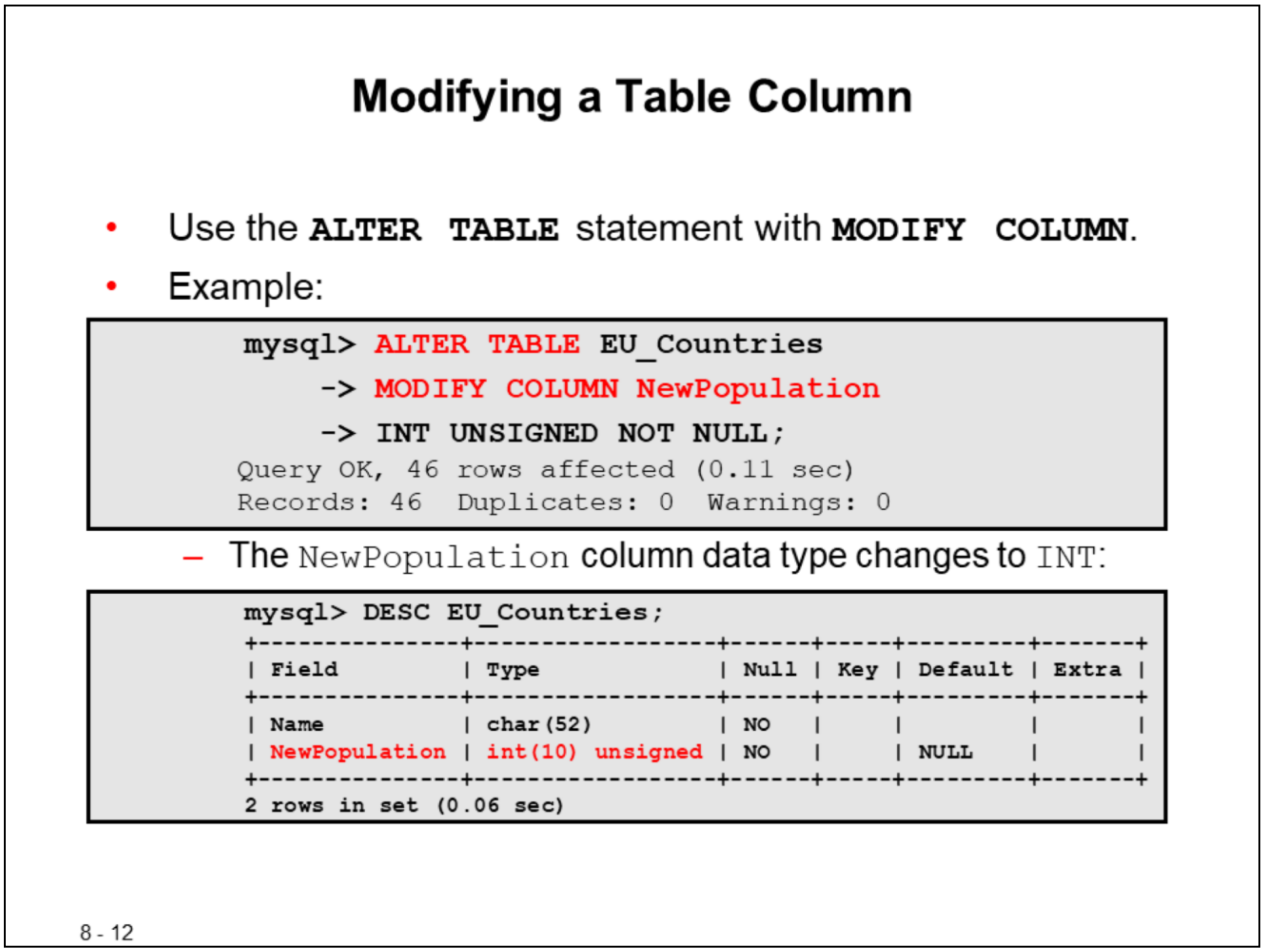
  – Removes the **ID** column:

```
mysql> DESC EU_Countries;
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| Name          | char(52)      | NO   |     |         |       |
| NewPopulation | decimal(12,1) | NO   |     | 0.0     |       |
+---------------+---------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

If a table contains only one column, you cannot drop the column. If you intend to remove the table, use `DROP TABLE` instead.

Do not remove a column from a table if it is a primary key. You cannot remove a column that is a foreign key referencing another table.

For more information about using `ALTER TABLE`, see the MySQL Reference Manual at: http://dev.mysql.com/doc/refman/5.6/en/alter-table.html.



Modifying a Table Column

- Use the **ALTER TABLE** statement with **MODIFY COLUMN**.
- Example:

```
mysql> ALTER TABLE EU_Countries
    -> MODIFY COLUMN NewPopulation
    -> INT UNSIGNED NOT NULL;
Query OK, 46 rows affected (0.11 sec)
Records: 46  Duplicates: 0  Warnings: 0
```

  – The `NewPopulation` column data type changes to `INT`:

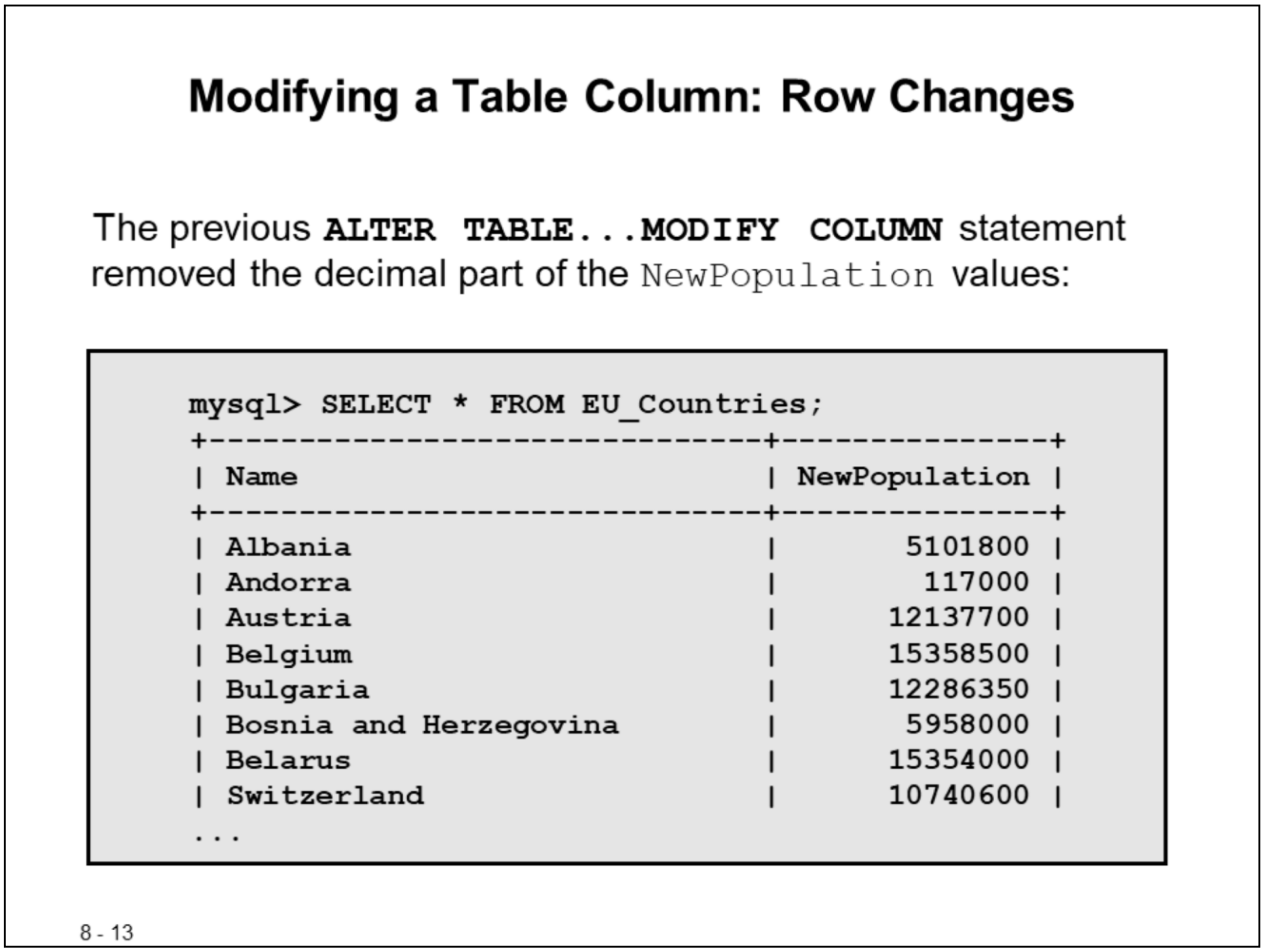
```
mysql> DESC EU_Countries;
+---------------+------------------+------+-----+---------+-------+
| Field         | Type             | Null | Key | Default | Extra |
+---------------+------------------+------+-----+---------+-------+
| Name          | char(52)         | NO   |     |         |       |
| NewPopulation | int(10) unsigned | NO   |     | NULL    |       |
+---------------+------------------+------+-----+---------+-------+
2 rows in set (0.06 sec)
```

8 - 12

The example in the slide shows how to change the `NewPopulation` column's data type from `DECIMAL` to `INT`, restricting column values to whole numbers, disallowing negative values with the `UNSIGNED` attribute and null values with `NOT NULL`.

When you modify a table column you have to reapply all the attributes you want to keep from the old column definition. For example, the old column definition did not permit null values. If you want to disallow `nulls` in the new column definition you need to specify `NOT NULL` again.

You cannot modify a column if it is a primary key and if a foreign key from another table references the column.



Modifying a Table Column: Row Changes

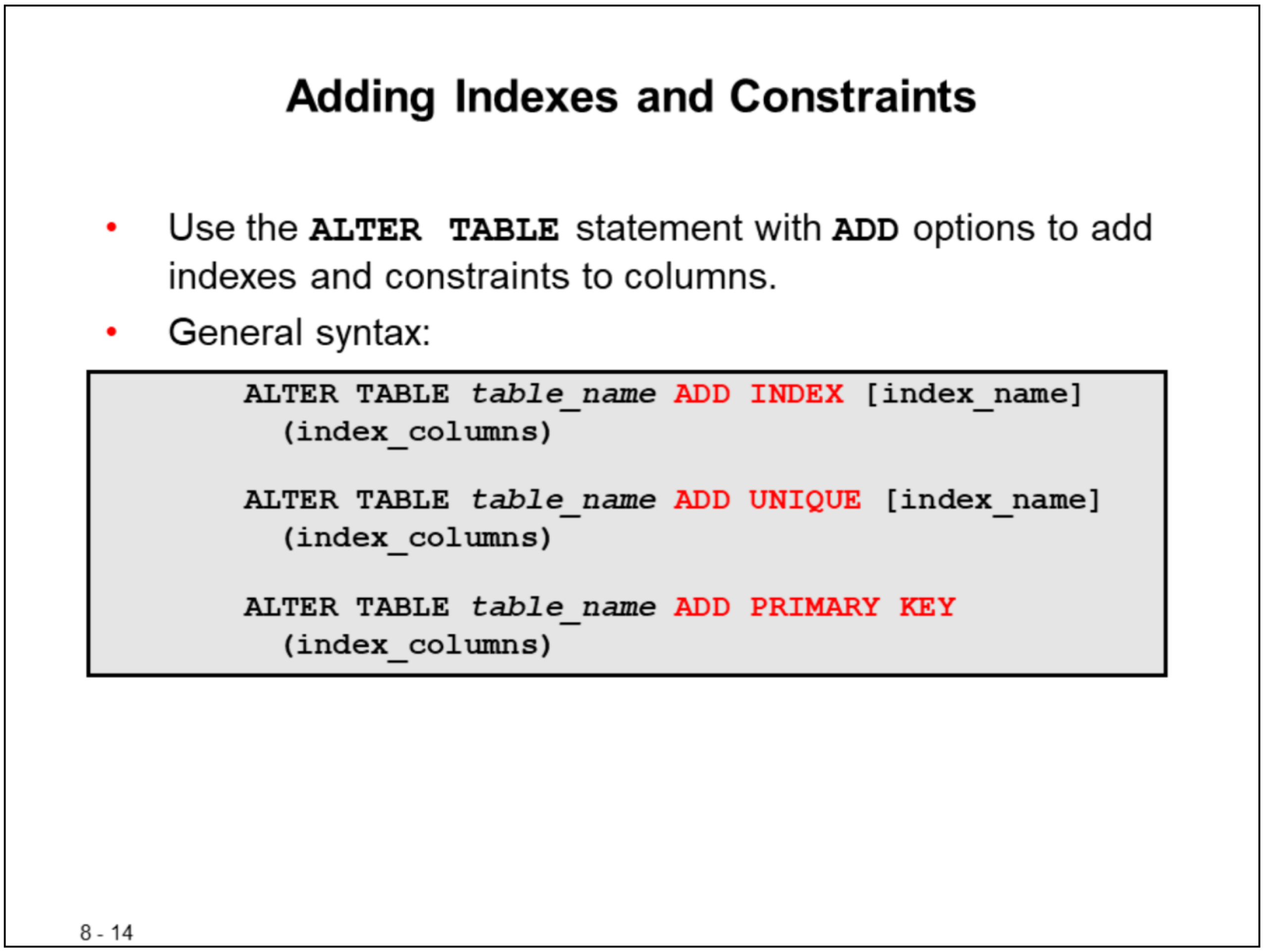
The previous **`ALTER TABLE...MODIFY COLUMN`** statement removed the decimal part of the `NewPopulation` values:

```
mysql> SELECT * FROM EU_Countries;
+------------------------------+---------------+
| Name                         | NewPopulation |
+------------------------------+---------------+
| Albania                      |       5101800 |
| Andorra                      |        117000 |
| Austria                      |      12137700 |
| Belgium                      |      15358500 |
| Bulgaria                     |      12286350 |
| Bosnia and Herzegovina       |       5958000 |
| Belarus                      |      15354000 |
| Switzerland                  |      10740600 |
...
```

Anything that can be added by using `CREATE TABLE` can be changed by using `ALTER TABLE`.

For more information about using `ALTER TABLE`, see the MySQL Reference Manual at: http://dev.mysql.com/doc/refman/5.6/en/alter-table.html.

Note: The statement for populating a table with data is covered in detail later in the course.



Adding Indexes and Constraints

- Use the **ALTER TABLE** statement with **ADD** options to add indexes and constraints to columns.
- General syntax:

```
ALTER TABLE table_name ADD INDEX [index_name]
  (index_columns)

ALTER TABLE table_name ADD UNIQUE [index_name]
  (index_columns)

ALTER TABLE table_name ADD PRIMARY KEY
  (index_columns)
```

8 - 14

Use `ALTER TABLE` to add indexes and constraints to existing columns. The index name is optional.

Adding a Column Index

- Use the **ALTER TABLE** statement with **ADD INDEX**.
- Example:

```
mysql> ALTER TABLE NewCity
    -> ADD INDEX Pop (Population);
Query OK, 0 rows affected (0.22 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

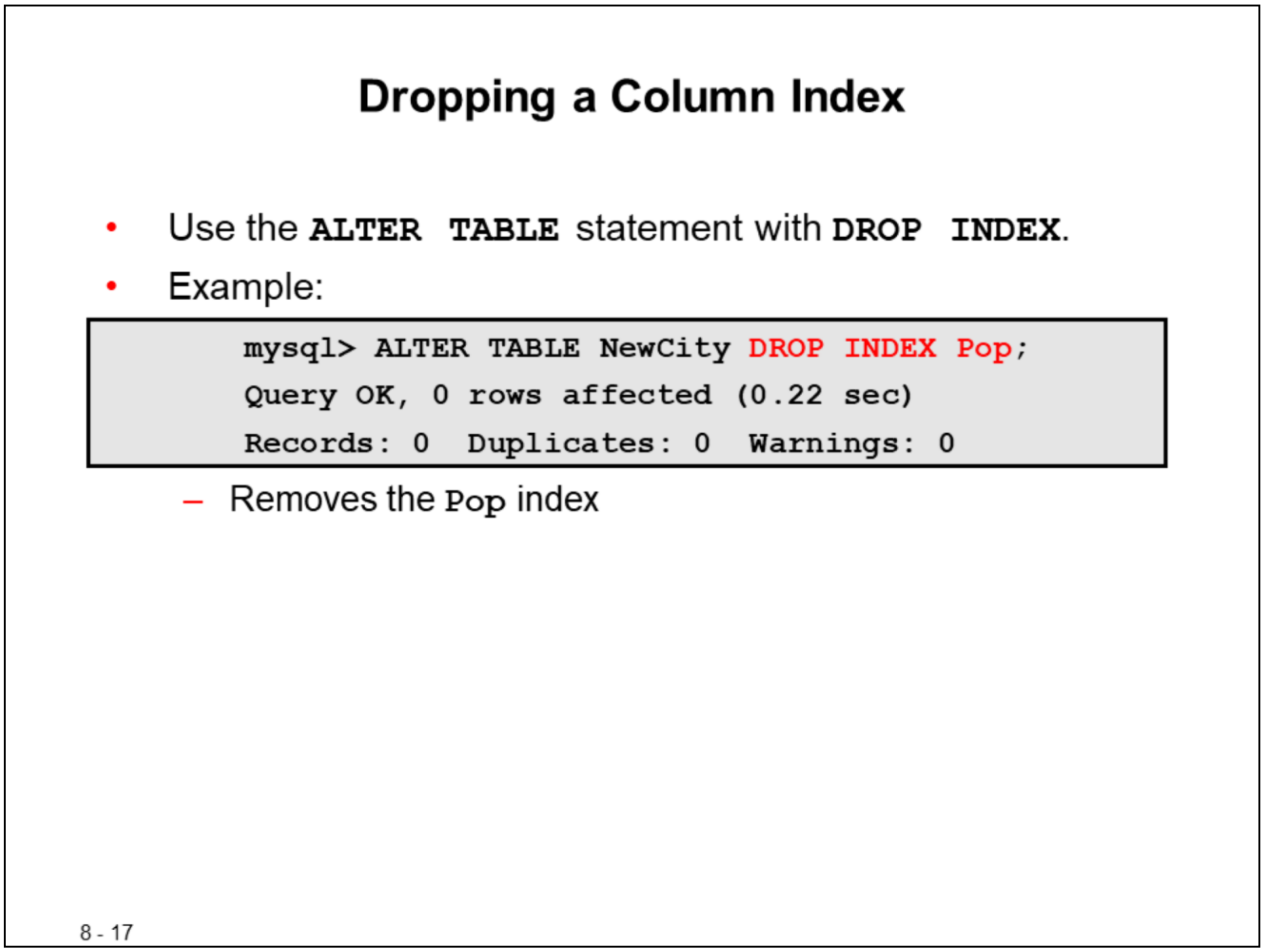
  – Adds the index `Pop` for the `Population` column

8 - 15

Adding a Column Index

Confirm the change by using **SHOW CREATE TABLE**:

```
mysql> SHOW CREATE TABLE NewCity\G
****************** 1. row ******************
Table: City
Create Table: CREATE TABLE 'city' (
'ID' int(11) NOT NULL auto_increment,
'Name' char(35) NOT NULL default '',
'CountryCode' char(3) NOT NULL default '',
'District' char(20) NOT NULL default '',
'Population' int(11) NOT NULL default '0',
PRIMARY KEY ('ID'),
KEY `CountryCode` (`CountryCode`),
KEY 'Pop' ('Population')
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```



Dropping a Column Index

- Use the **ALTER TABLE** statement with **DROP INDEX**.
- Example:

```
mysql> ALTER TABLE NewCity DROP INDEX Pop;
Query OK, 0 rows affected (0.22 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

  – Removes the **Pop** index

8 - 17

Confirm the change in the slide example with `SHOW CREATE TABLE`:

```
mysql> SHOW CREATE TABLE NewCity\G
*************************** 1. row ***************************
       Table: NewCity
Create Table: CREATE TABLE `newcity` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  `Name` char(35) NOT NULL DEFAULT '',
  `CountryCode` char(3) NOT NULL DEFAULT '',
  `District` char(20) NOT NULL DEFAULT '',
  `Population` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`ID`),
  KEY `CountryCode` (`CountryCode`),
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.08 sec)
```

You can remove constraints in the same way, for example:

```
ALTER TABLE City DROP FOREIGN KEY CountryCode;
```