LAB 02

DATA TYPES. CREATE AND ALTER TABLE STRUCTURE

❖ **Main contents:**

- Data types of MySQL
- Create Table
- Alter Table Structure
- Drop Table

1. Data types

MySQL supports database tables that contain columns with different data types. The following tables list the supported MySQL data types.

In MySQL there are three main data types: numeric, string, date and time.

Numeric Data Types

The following table describes the numeric data types in MySQL:

Type	**Capacity**
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT/INTEGER	4 bytes
BIGINT	8 bytes

Notes: Type *BOOLEAN corresponding to TINYINT(1)*

Type	**Capacity**
FLOAT	4 bytes
DOUBLE	8 bytes

DECIMAL	Depends on when defining the column

String Data Types

In MySQL, string can store everything from text to binary data like images and files. String can be compared and searched based on patterns using LIKE clauses or regular expressions.

The table below shows the string data types in MySQL:

String Data Type	**Description**
CHAR	A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY	Equal to CHAR(), but stores binary byte strings. The size parameter specifies the column length in bytes. Default is 1
VARBINARY	Equal to VARCHAR(), but stores binary byte strings. The *size* parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large Objects). Max length: 255 bytes
BLOB	For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data
MEDIUMBLOB	Holds a string with a maximum length of 16,777,215 characters
LONGBLOB	For BLOBs (Binary Large Objects). Holds up to 4,294,967,295 bytes of data
TINYTEXT	Holds a string with a maximum length of 255 characters

TEXT	Holds a string with a maximum length of 65,535 bytes
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters

Date and Time Data Types

MySQL provides Date, Time, and Date and Time data types. In addition, MySQL also provides the *timestamp* data type to save the change time of the record.

The following table shows the Date and Time data types in MySQL:

Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
TIME	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
DATETIME	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

The TIMESTAMP column data type plays a special rule due to automatically updating the latest changed time values when a record is added or updated.

2. Create Data table

The CREATE TABLE statement is used to create a new table in a database.

Syntax:

```
CREATE TABLE [IF NOT EXISTS] table_name(

       <column name><type> [<default value>] [column constraints],

      ...

      <column name><type> [<default value>] [column constraints],

      <table constraint>,

      ...

      <table constraint>

) type=table_type;
```

MySQL supports the IF NOT EXISTS option to avoid the error when creating a *table_name* table that already exists in the database.

DEFAULT value: MySQL allows default values to be assigned to a column. If the value of that column is not specified when adding data to the table, the column value will be assigned a default value. The default value of a column is NULL.

Table_type: Determine the type of data table when storing (note this attribute is a unique feature of MySQL). If not specified then MySQL will use the default table type. MySQL supports different types of storage tables, allowing for optimal database usage. Some types of tables in MySQL such as MyISAM, InnoDB, BerkeleyDB (BDB), MERGE, HEAP ...
MyISAM: MyISAM tables work very fast, but do not support transactions. Often used in Web applications, which is the default table type in MySQL versions prior to 5.5.
InnoDB: InnoDB tables support secure transactions, foreign keys are supported. InnoDB is the default storage engine from MySQL 5.5.

SQL constraints include: **Primary Key, Foreign Key, Not Null, Unique, Check**. If the updated data violates the declared constraint, it will be rejected.

Constraints can be defined in two ways:

1) Column constraint: constraints are applied to a specific column

2) Table constraint: is declared separately, can apply constraints for one or more columns.

PRIMARY KEY (primary key constraint): This constraint defines a column or combination of columns that uniquely identifies each row in the table.

NOT NULL: This constraint requires the value of an unallowable column to be NULL.

UNIQUE: Constraints that require the values of the column are distinguished. Note that with this constraint the value of the column can be NULL if the NOT NULL constraint is not applied on the column.

CHECK: MYSQL does not support this constraint. We can declare this constraint but they have no effect.

The primary key constraint is declared as a column-level constraint

```
Column_name datatype [CONSTRAINT constraint_name] PRIMARY
KEY
```

The primary key constraint is declared as a table-level constraint

```
[CONSTRAINT constraint_name] PRIMARY KEY
 (column_name1, column_name2, …)
```

Example: Create ***employees*** table with the primary key constraint is declared as a column-level constraint

```
CREATE TABLE employees (
              employeeNumber int(11) NOT NULL PRIMARY KEY ,
              lastName varchar(50) NOT NULL,
              firstName varchar(50) NOT NULL,
              extension varchar(10) NOT NULL,
              email varchar(100) NOT NULL,
              officeCode varchar(10) NOT NULL,
              reportsTo int(11) default NULL,
              jobTitle varchar(50) NOT NULL
            ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Or use the above method and name the constraint

```
CREATE TABLE employees (
              employeeNumber int(11) NOT NULL CONSTRAINT
emp_id_pk   PRIMARY KEY,
              lastName varchar(50) NOT NULL,
              firstName varchar(50) NOT NULL,
              extension varchar(10) NOT NULL,
              email varchar(100) NOT NULL,
              officeCode varchar(10) NOT NULL,
              reportsTo int(11) default NULL,
              jobTitle varchar(50) NOT NULL
            ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Name the constraint

The declaration `CONSTRAINT <name> <constraint>` is used to name the constraint. The purpose of naming constraints is that when updating constraint violation data, a database management system usually includes the binding name in an error message. You can also use binding names when modifying or deleting constraints. As in the example above, the primary key constraint is named `emp_id_pk.`

Example: Create an *employees* table with the primary key constraint is declared as a table-level constraint instead of `column-level constraint`.

```
 CREATE TABLE employees (
             employeeNumber int(11) NOT NULL,
             lastName varchar(50) NOT NULL,
             firstName varchar(50) NOT NULL,
             extension varchar(10) NOT NULL,
             email varchar(100) NOT NULL,
             officeCode varchar(10) NOT NULL,
             reportsTo int(11) default NULL,
             jobTitle varchar(50) NOT NULL,
             PRIMARY KEY (employeeNumber)
           ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

FOREIGN KEY

The keyword FOREIGN KEY is used to identify the foreign key. In the example below, specify the *country_id* column as the foreign key, referring to the primary key of the *country* table.

```
CREATE TABLE city (
    city_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,

    city VARCHAR(50) NOT NULL,

    country_id SMALLINT UNSIGNED NOT NULL,

    last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
    ON UPDATE CURRENT_TIMESTAMP,

    PRIMARY KEY(city_id),

    CONSTRAINT fk_city_country FOREIGN KEY (country_id)
    REFERENCES country (country_id) ON DELETE RESTRICT ON
    UPDATE CASCADE

);
```

Meaning of accompanying options when declaring foreign key constraints:

- `ON DELETE RESTRICT`: It is not allowed to delete the row of the referenced table while the data is referencing. In the above example, it is not allowed to delete the data line of the *country* table if there exists a row of data from the table of *city* referenced.

- `ON UPDATE CASCADE`: It means when updating the data in the reference table, the reference table data will be automatically updated. In the above example, when changing the data of the *country_id* column of the *country* table, the *country_id* column of the *city* table will be automatically updated.

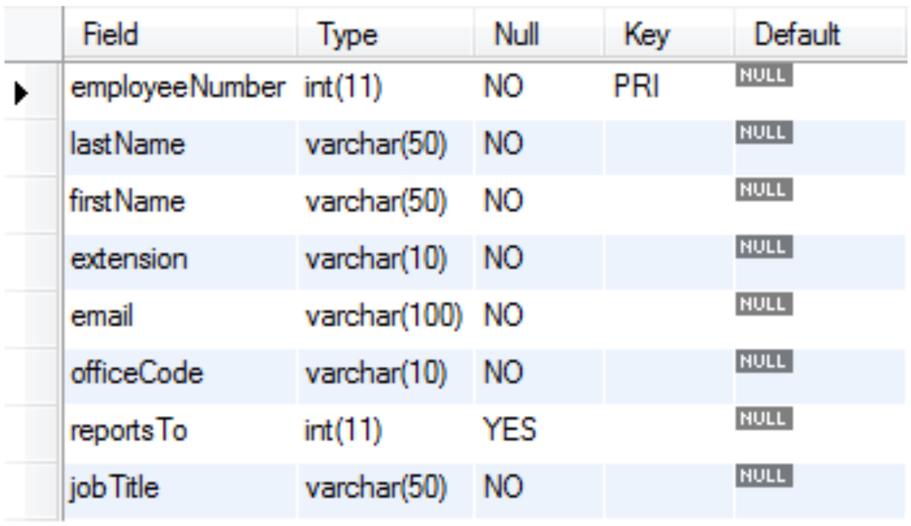
- When these options are not used, the `RESTRICT` default will be used for `DELETE` and `UPDATE` events.

After creating the data tables, you can check the structure of the data columns in the table.

Example: Displaying information of the *employees* table

```
 DESCRIBE employees;
```

Result:



Field	Type	Null	Key	Default
employeeNumber	int(11)	NO	PRI	NULL
lastName	varchar(50)	NO		NULL
firstName	varchar(50)	NO		NULL
extension	varchar(10)	NO		NULL
email	varchar(100)	NO		NULL
officeCode	varchar(10)	NO		NULL
reportsTo	int(11)	YES		NULL
jobTitle	varchar(50)	NO		NULL

Besides the DESCRIBE statement, you can use the statement:

```
SHOW CREATE TABLE Table_Name;
```

will display the statement used to create the data table.

3. Alter Table Structure

Besides creating a table, to modify the table structure that already exists in the database, use the ALTER TABLE statement. The statement may be used to:

• Add, delete, edit columns of the table

• Add and remove constraints

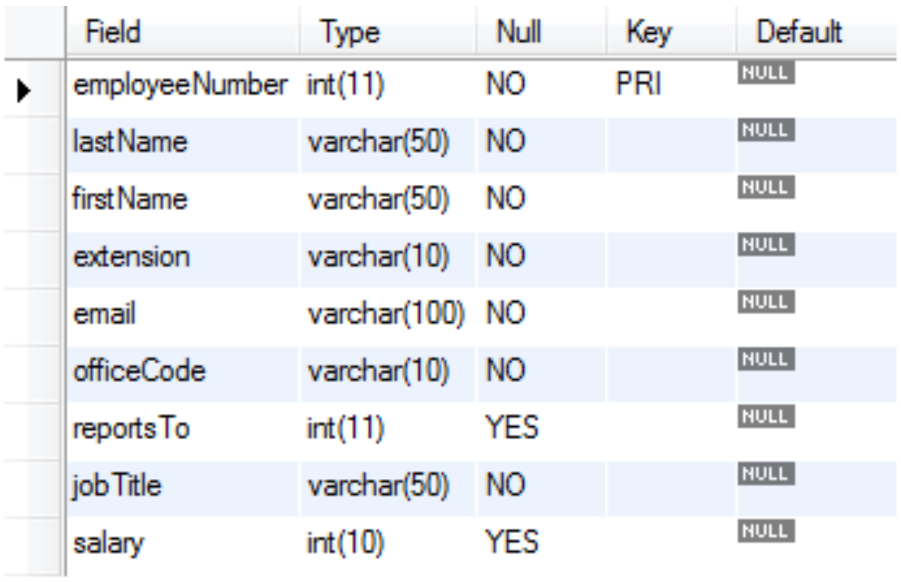
The syntax of the ALTER TABLE statement is as follows:

```
 ALTER TABLE table_name option[,option...]

 OPTIONs:

      ADD [COLUMN] <column_definition>

      MODIFY [COLUMN] <create_definition>

      DROP [COLUMN] <column_name>
```

```
        ADD <table_constraint>

        DROP <constraint_name>
```

Example: Add a *salary* column of the type INT, not exceeding 10 digits, the constraint must not be blank in the *employees* data table.

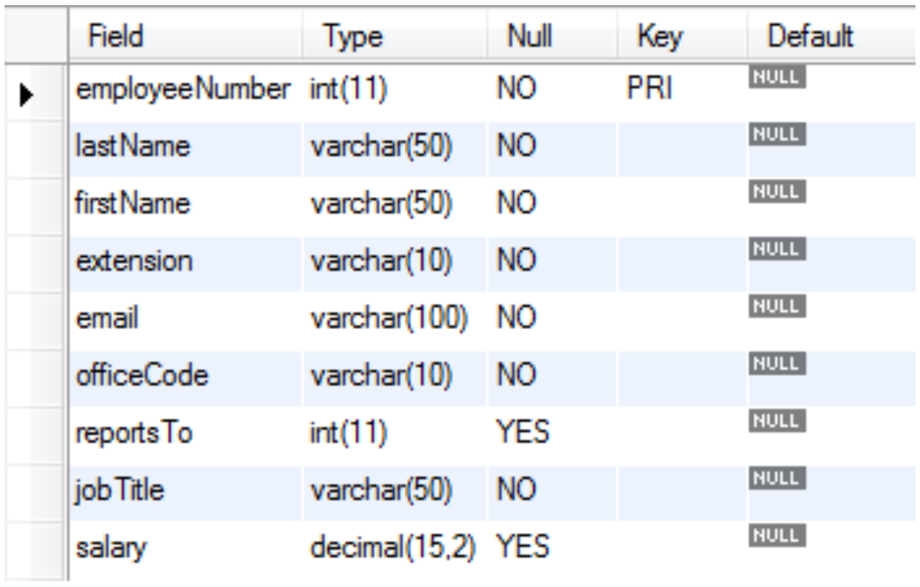
```
ALTER TABLE employees ADD salary INT(10) NOT NULL;
```



Field	Type	Null	Key	Default
employeeNumber	int(11)	NO	PRI	NULL
lastName	varchar(50)	NO		NULL
firstName	varchar(50)	NO		NULL
extension	varchar(10)	NO		NULL
email	varchar(100)	NO		NULL
officeCode	varchar(10)	NO		NULL
reportsTo	int(11)	YES		NULL
jobTitle	varchar(50)	NO		NULL
salary	int(10)	YES		NULL

Example: Modify the type of salary column to (15,2)

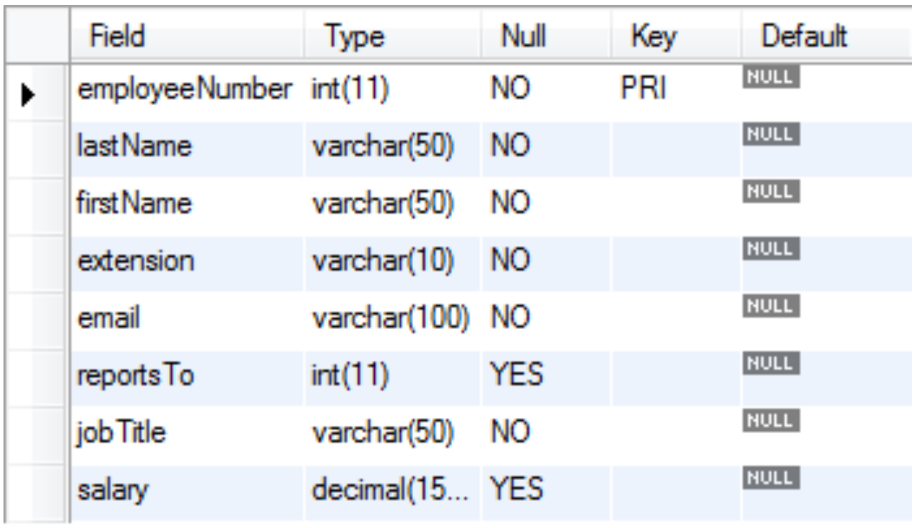
```
ALTER TABLE employees MODIFY salary decimal(15,2);
```



Field	Type	Null	Key	Default
employeeNumber	int(11)	NO	PRI	NULL
lastName	varchar(50)	NO		NULL
firstName	varchar(50)	NO		NULL
extension	varchar(10)	NO		NULL
email	varchar(100)	NO		NULL
officeCode	varchar(10)	NO		NULL
reportsTo	int(11)	YES		NULL
jobTitle	varchar(50)	NO		NULL
salary	decimal(15,2)	YES		NULL

Ví dụ: Drop the *officeCode* column from the *employees* table

```
ALTER TABLE employees DROP officeCode;
```



Field	Type	Null	Key	Default
employeeNumber	int(11)	NO	PRI	NULL
lastName	varchar(50)	NO		NULL
firstName	varchar(50)	NO		NULL
extension	varchar(10)	NO		NULL
email	varchar(100)	NO		NULL
reportsTo	int(11)	YES		NULL
jobTitle	varchar(50)	NO		NULL
salary	decimal(15...	YES		NULL

4. Drop table

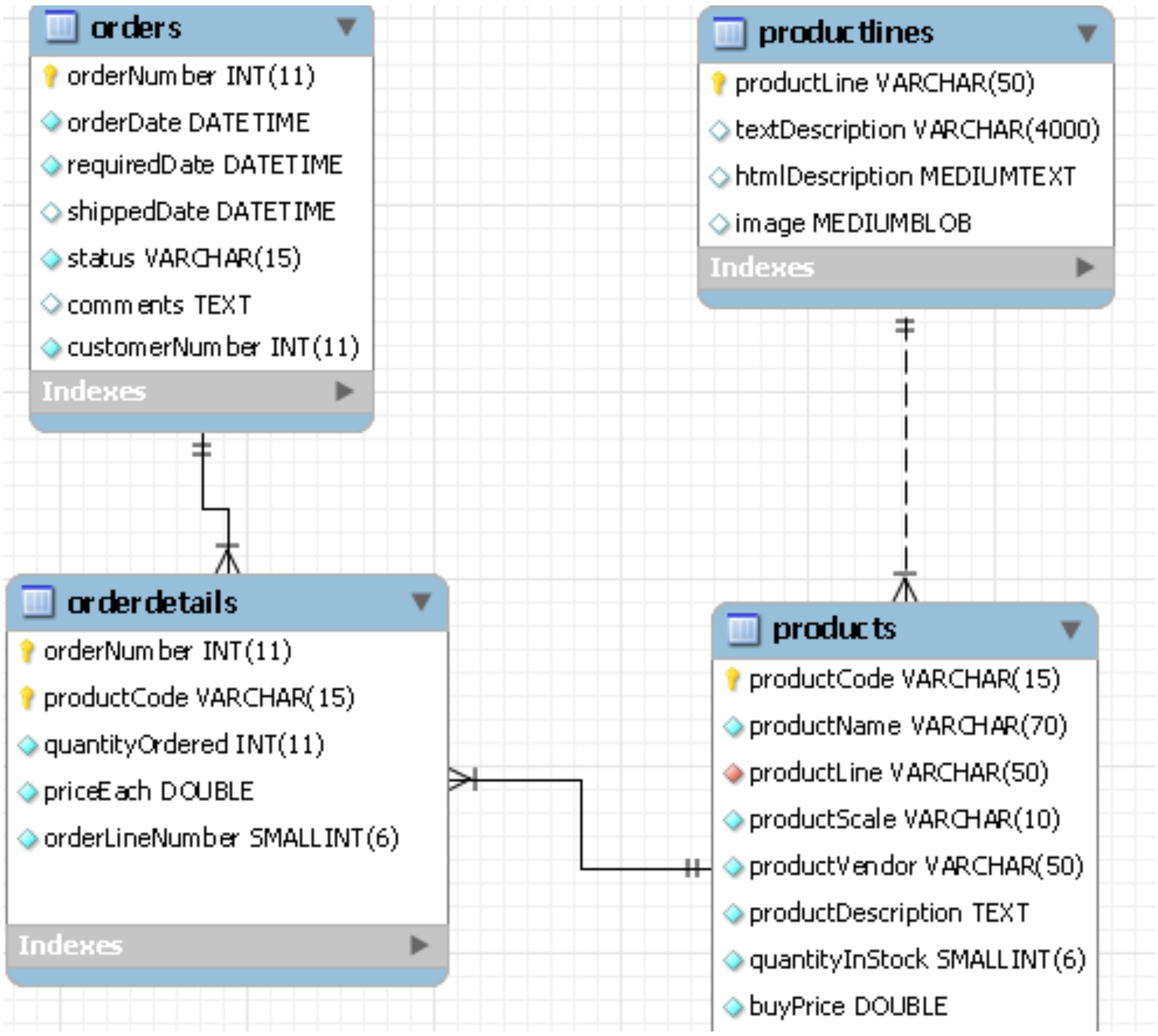
To drop a table from database, use the statement DROP TABLE:

```
    DROP TABLE [IF EXISTS] <table_name>;
```

MySQL allows deleting multiple tables at the same time by listing the names of the tables separated by commas. The IF EXISTS option is used to avoid deleting tables that do not exist in the database.

❖ Practical exercises

1. Create database *My_Classicmodels* includes 4 tables: *productlines, products, orders and orderdetails* with properties as shown in the figure below. Primary keys of type INT use auto-increment type `AUTO_INCREMENT`. *Hint*: The primary key is made up of more than one column combination that needs to be declared in table-level constraint.
2. After creating the above 4 data tables, add foreign key constraints between tables as shown in the figure. Foreign key constraints use the `ON UPDATE CASCADE` option.


orders
orderNumber INT(11)
orderDate DATETIME
requiredDate DATETIME
shippedDate DATETIME
status VARCHAR(15)
comments TEXT
customerNumber INT(11)
Indexes
productlines
productLine VARCHAR(50)
textDescription VARCHAR(4000)
htmlDescription MEDIUMTEXT
image MEDIUMBLOB
Indexes
orderdetails
orderNumber INT(11)
productCode VARCHAR(15)
quantityOrdered INT(11)
priceEach DOUBLE
orderLineNumber SMALLINT(6)
Indexes
products
productCode VARCHAR(15)
productName VARCHAR(70)
productLine VARCHAR(50)
productScale VARCHAR(10)
productVendor VARCHAR(50)
productDescription TEXT
quantityInStock SMALLINT(6)
buyPrice DOUBLE