



# Acessando um ambiente de banco de dados e gerenciando tabelas

*Miriã Corrêa*

Análise e Desenvolvimento de Sistemas

[miriacoeelho@gmail.com](mailto:miriacoeelho@gmail.com)



# Sumário



SQL

Create Database

Tipos de Dados

Create Table



O comando DROP

# SQL

- **SQL** (Structured Query Language) Linguagem de consulta estrutura.
- A linguagem SQL pode ser considerada um dos principais motivos para o sucesso dos bancos de dados relacionais comerciais.
- Oferece uma interface de linguagem declarativa de nível mais alto, de modo que o usuário apenas especifica qual deve ser o resultado, deixando a otimização real e as decisões sobre como executar a consulta para o SGBD.



# SQL

- A SQL agora é uma linguagem padrão para SGBDs relacionais comerciais.
- SQL é uma linguagem de banco de dados abrangente: tem instruções para definição de dados, consultas e atualizações.
- É uma DDL e uma DML.



# SQL

- Além disso, ela tem facilidades para:
  - definir visões sobre o banco de dados;
  - especificar segurança e autorização;
  - definir restrições de integridade para especificar controles de transação.
- Usa os termos tabela, linha e coluna para os termos do modelo relacional formal relação, tupla e atributo, respectivamente.



# SQL

- O principal comando SQL para a definição de dados é o CREATE;
- Este comando pode ser usado para criar esquemas, tabelas (relações) e domínios.



# Banco de dados

- O comando CREATE DATABASE é usado para criar uma base de dados, dando-lhe um nome.
- A sintaxe SQL usada para criar uma base de dados é:

```
CREATE {DATABASE | SCHEMA} db_name
```

- A partir da versão 5.0.2 do MySQL CREATE SCHEMA é um sinônimo para CREATE DATABASE.

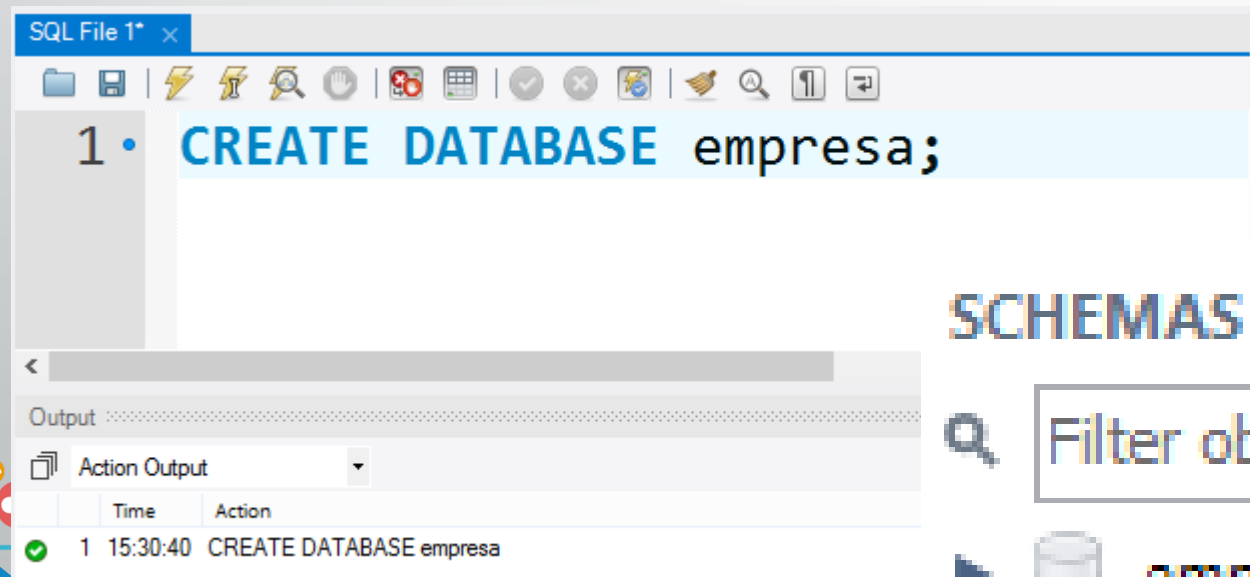


# Banco de dados

- Exemplo:

```
CREATE DATABASE empresa;
```

- Executando o comando acima no MySQL obtemos:



## SCHEMAS



empresa



# Tipos de dados

- Os **tipos de dados** básicos disponíveis para atributos são:
  - Numérico;
  - Cadeia ou sequência de caracteres;
  - Cadeia ou sequência de bits;
  - Booleano;
  - Data e hora.



# Numérico

- Números inteiros de vários tamanhos (INT, INTEGER, TINYINT, MEDIUMINT, SMALLINT e BIGINT).

Type	Storage	Minimum Value	Maximum Value
	(Bytes)	(Signed/Unsigned)	(Signed/Unsigned)
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607
		0	16777215
INT	4	-2147483648	2147483647
		0	4294967295
BIGINT	8	-9223372036854775808	9223372036854775807
		0	18446744073709551615

# Numérico

- Números de ponto flutuante (reais) de várias precisões (FLOAT ou REAL e DOUBLE).
- O formato dos números pode ser declarado usando `FLOAT(i,j)`, `REAL(i,j)`, `DOUBLE(i,j)`, `DECIMAL(i,j)` ou `NUMERIC(i,j)`
- *i* é a precisão, é o número total de dígitos decimais e *j*, a escala, é o número de dígitos após o ponto decimal.



- Exemplo:  
Salário DECIMAL (5,2)
- É capaz de armazenar qualquer valor com cinco dígitos e duas casas decimais.
- Os valores possíveis para salário são -999,99 a 999,99 .



# Numérico

- Decimal e numérico são tipos utilizados para armazenar valores exatos de dados numéricos.
- Esses tipos são usados quando é importante preservar a exatidão como, por exemplo, dados monetários.
- O máximo número de dígitos para decimal e numérico é 65.



# Numérico

- FLOAT permite números de  $-3.402823466E+38$  até  $-1.175494351E-38$ , 0, e  $1.175494351E-38$  to  $3.402823466E+38$ .
- DOUBLE permite números de  $1.7976931348623157E+308$  até  $-2.2250738585072014E-308$ , 0, e  $2.2250738585072014E-308$  até  $1.7976931348623157E+308$ .

# Cadeia de bits

- Outro tipo de dados é a cadeia de bits  $\text{BIT}(n)$ .
- $n$  é o número máximo de bits.



# Numérico

*data\_type:*

```
    BIT [ ( length ) ]  
| TINYINT [ ( length ) ] [ UNSIGNED ] [ ZEROFILL ]  
| SMALLINT [ ( length ) ] [ UNSIGNED ] [ ZEROFILL ]  
| MEDIUMINT [ ( length ) ] [ UNSIGNED ] [ ZEROFILL ]  
| INT [ ( length ) ] [ UNSIGNED ] [ ZEROFILL ]  
| INTEGER [ ( length ) ] [ UNSIGNED ] [ ZEROFILL ]  
| BIGINT [ ( length ) ] [ UNSIGNED ] [ ZEROFILL ]  
| REAL [ ( length, decimals ) ] [ UNSIGNED ] [ ZEROFILL ]  
| DOUBLE [ ( length, decimals ) ] [ UNSIGNED ] [ ZEROFILL ]  
| FLOAT [ ( length, decimals ) ] [ UNSIGNED ] [ ZEROFILL ]  
| DECIMAL [ ( length [, decimals ] ) ] [ UNSIGNED ] [ ZEROFILL ]  
| NUMERIC [ ( length [, decimals ] ) ] [ UNSIGNED ] [ ZEROFILL ]
```



# Cadeia de caracteres

Cadeia ou sequência de caracteres:

- CHAR(n) e VARCHAR(n).
- n é o número de caracteres
- Os tipos CHAR e VARCHAR são parecidos, mas diferem no modo como são armazenados e recuperados.



# CHAR

- O tamanho de um campo CHAR é fixado pelo tamanho declarado na criação da tabela.
- O tamanho pode ser qualquer valor entre 1 e 255.
- Quando valores CHAR são armazenados, eles são preenchidos a direita com espaços até o tamanho especificado.
- Quando valores CHAR são recuperados, espaços extras são removidos.



# CHAR

- Por exemplo:  
nome char(10)
- Se quisermos armazenar o nome 'Maria', teremos armazenado 'Maria \_ \_ \_ \_ \_'.

Valor	CHAR (4)	Exigência p/ armazenamento
' '	' '	4 bytes
'ab'	'ab '	4 bytes
'abcd'	'abcd'	4 bytes
'abcdefgh'	'abcd'	4 bytes

# VARCHAR

- Valores no campo VARCHAR são strings de tamanho variável.
- Um campo VARCHAR pode ter qualquer tamanho entre 1 e 255, assim como para campo CHAR.
- No entanto, diferente de CHAR, valores VARCHAR são armazenados usando apenas quantos caracteres forem necessários, mais 1 byte para gravar o tamanho.



# VARCHAR

- Ao contrário do CHAR, espaços extras são removidos quando valores são armazenados.
- Por exemplo:  
nome varchar(10)
- Se quisermos armazenar o nome 'Maria', teremos armazenado 'Maria'.

VARCHAR (4)	Exigência p/ armazenamento
' '	1 byte
'ab'	3 bytes
'abcd'	5 bytes
'abcd'	5 bytes

**n bytes para os caracteres necessários +1 byte para gravar o tamanho**



# Booleano

- Tem os valores tradicionais TRUE ou FALSE.
- Para declarar valores booleanos em MySQL usamos o TINYINT(1), em que os valores iguais a zero são considerados falsos e os valores diferentes de zero são considerados verdadeiros.



# DATE

- O tipo DATE é usado para armazenar valores que contém apenas informações sobre a data.
- MySQL recupera e exibe valores do tipo DATE no formato 'YYYY-MM-DD'.
- A faixa de valores suportada é '1000-01-01' para '9999-12-31'.



# YEAR

- O tipo YEAR é usado para armazenar valores que contém apenas informações sobre o ano.
- A faixa de valores suportada é '1901' até '2155'.





# TIME

- O tipo TIME é usado para armazenar valores que contém apenas informações sobre a hora.
- MySQL recupera e exibe valores do tipo TIME no formato 'HH:MM:SS' ou 'HHH:MM:SS' para armazenar longos valores de hora.
- A faixa de valores suportada é '-838:59:59' até '838:59:59'.



# DATETIME

- O tipo DATETIME é usado para armazenar valores que contêm partes de data e hora.
- MySQL recupera e mostra valores DATETIME no formato 'YYYY-MM-DD HH: MM: SS'.
- A faixa de valores suportada é '1000-01-01 00:00:00' para '9999-12-31 23:59:59'.



# TIMESTAMP

- O tipo DATETIME é usado para armazenar valores que contêm partes de data e hora.
- TIMESTAMP tem um alcance de '1970-01-01 00:00:01 ' UTC para '2038-01-19 03:14:07 ' UTC.
- **UTC** (sigla de Universal Time Coordinated), é o fuso horário de referência a partir do qual se calculam todas as outras zonas horárias do mundo.



# Data e Hora

- | DATE
- | TIME
- | TIMESTAMP
- | DATETIME
- | YEAR



# Tabelas

- A tabela é a forma básica de armazenamento da informação em um SGBD relacional.
- O comando `CREATE TABLE` é usado para especificar o objeto tabela, dando-lhe um nome e especificando seus atributos e restrições iniciais.



# CREATE TABLE

- A sintaxe SQL básica usada para criar uma tabela é:

```
CREATE TABLE <nome tabela> (<nome coluna><tipo coluna> [<restrição atributo>]  
{, <nome coluna><tipo coluna> [<restrição atributo>] }  
[<restrição tabela>{,<restrição tabela>}] )
```

- Exemplo:

```
CREATE TABLE aluno (nome varchar (30), matricula decimal(4),  
tipo_aluno integer, curso varchar(3))
```



# CREATE TABLE

```
CREATE TABLE aluno (nome varchar (30), matricula decimal(4),  
                        tipo_aluno integer, curso varchar(3));
```

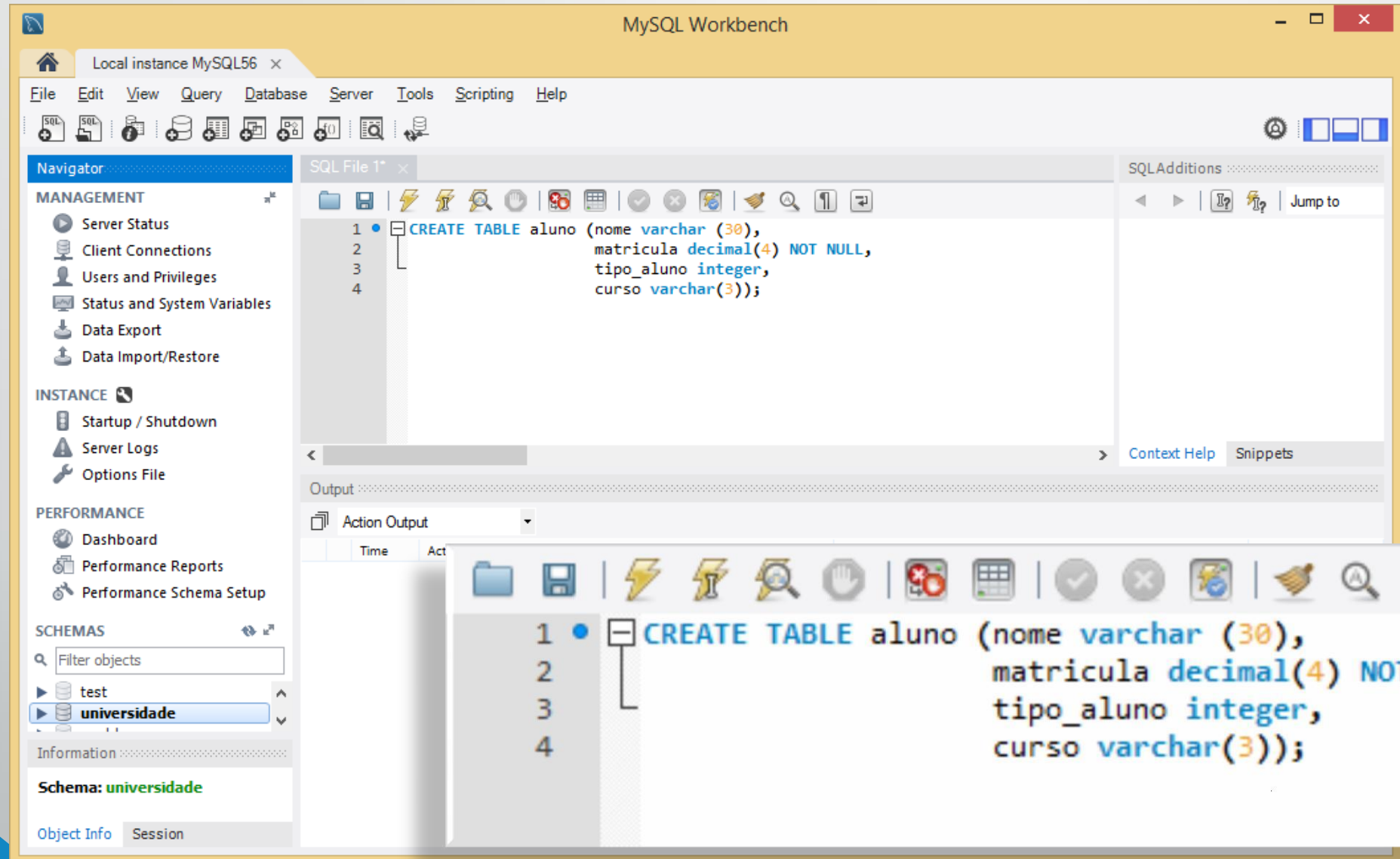
- Resultado:

✓ 3 10:54:33 CREATE TABLE aluno (nome varchar (30), matricula decimal(4), tipo\_aluno integer, curso varchar(3))

	nome	matricula	tipo_aluno	curso
--	------	-----------	------------	-------

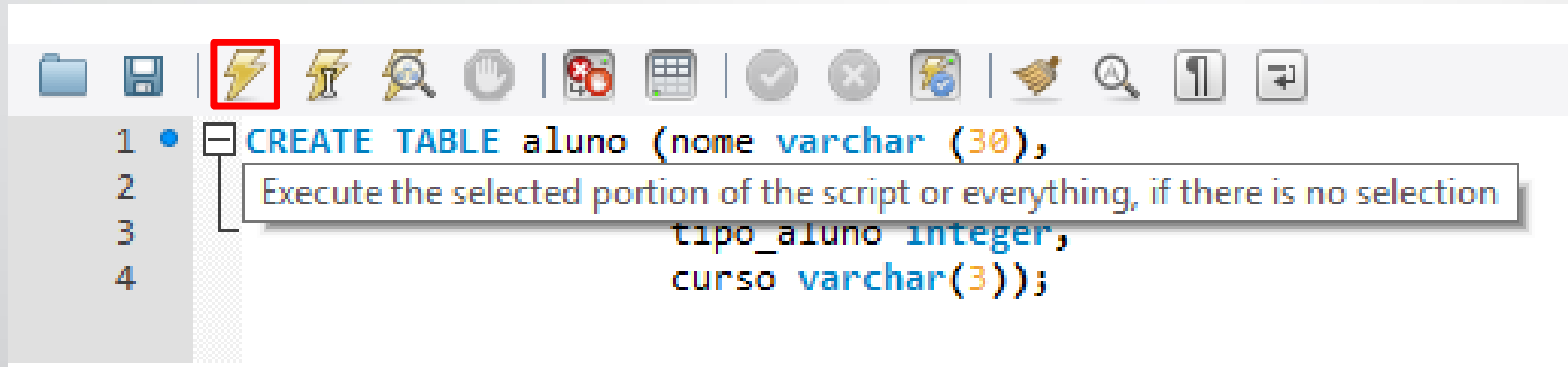


# Exemplo – CREATE TABLE

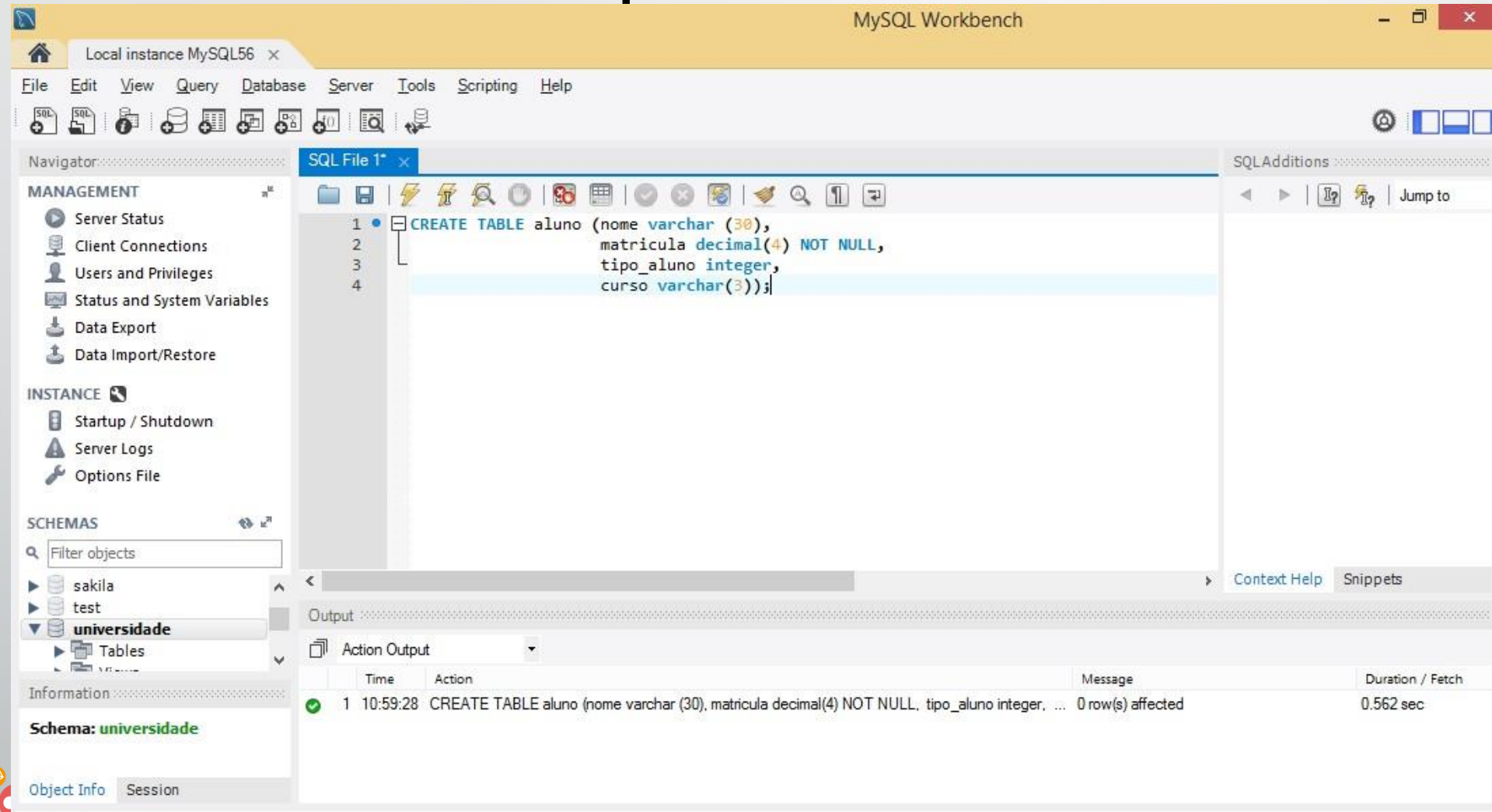




# Exemplo – CREATE TABLE



# Exemplo – CREATE TABLE



The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'MANAGEMENT' and 'SCHEMAS' panels. The 'SCHEMAS' panel shows the 'universidade' schema selected. The main editor window displays the SQL code for creating the 'aluno' table:

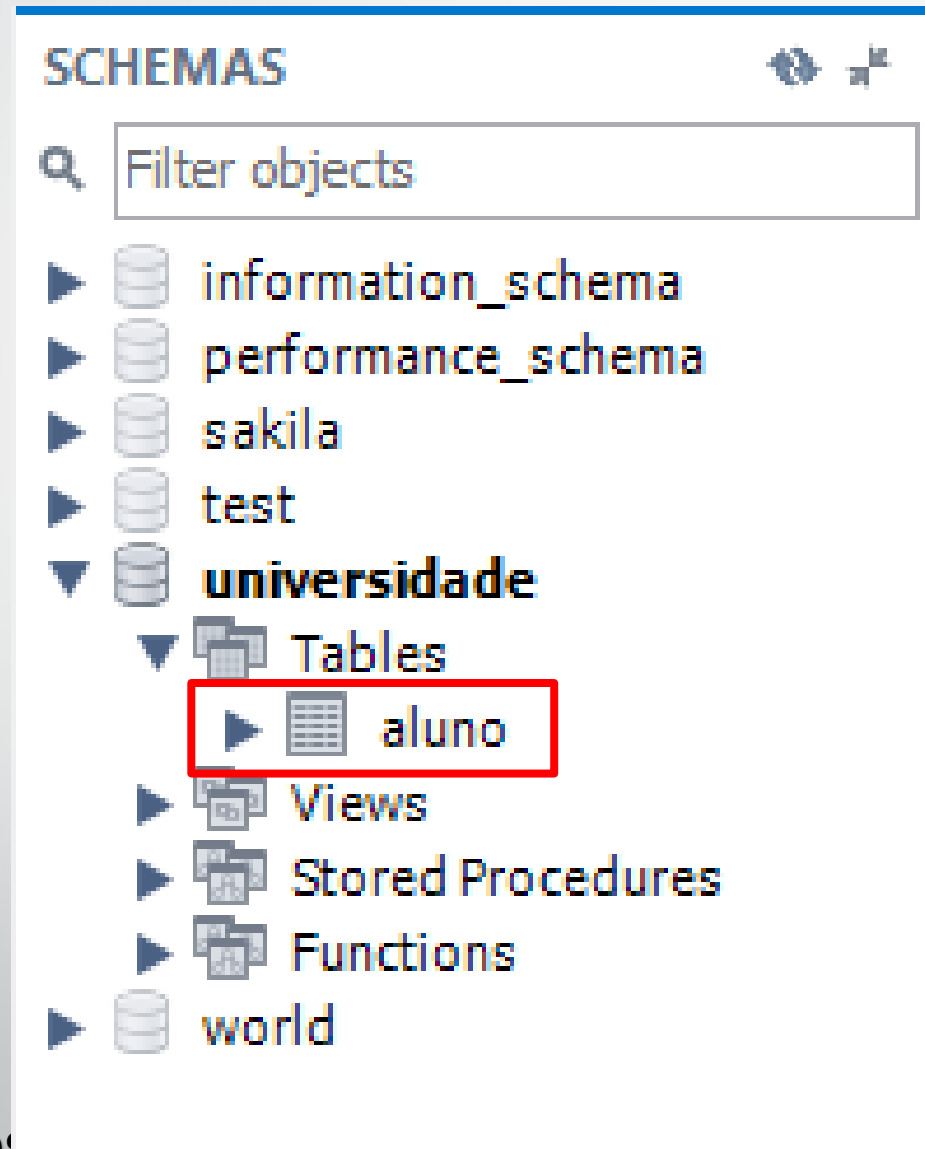
```
1 CREATE TABLE aluno (nome varchar (30),  
2 matricula decimal(4) NOT NULL,  
3 tipo_aluno integer,  
4 curso varchar(3));
```

The bottom output panel shows the execution results:

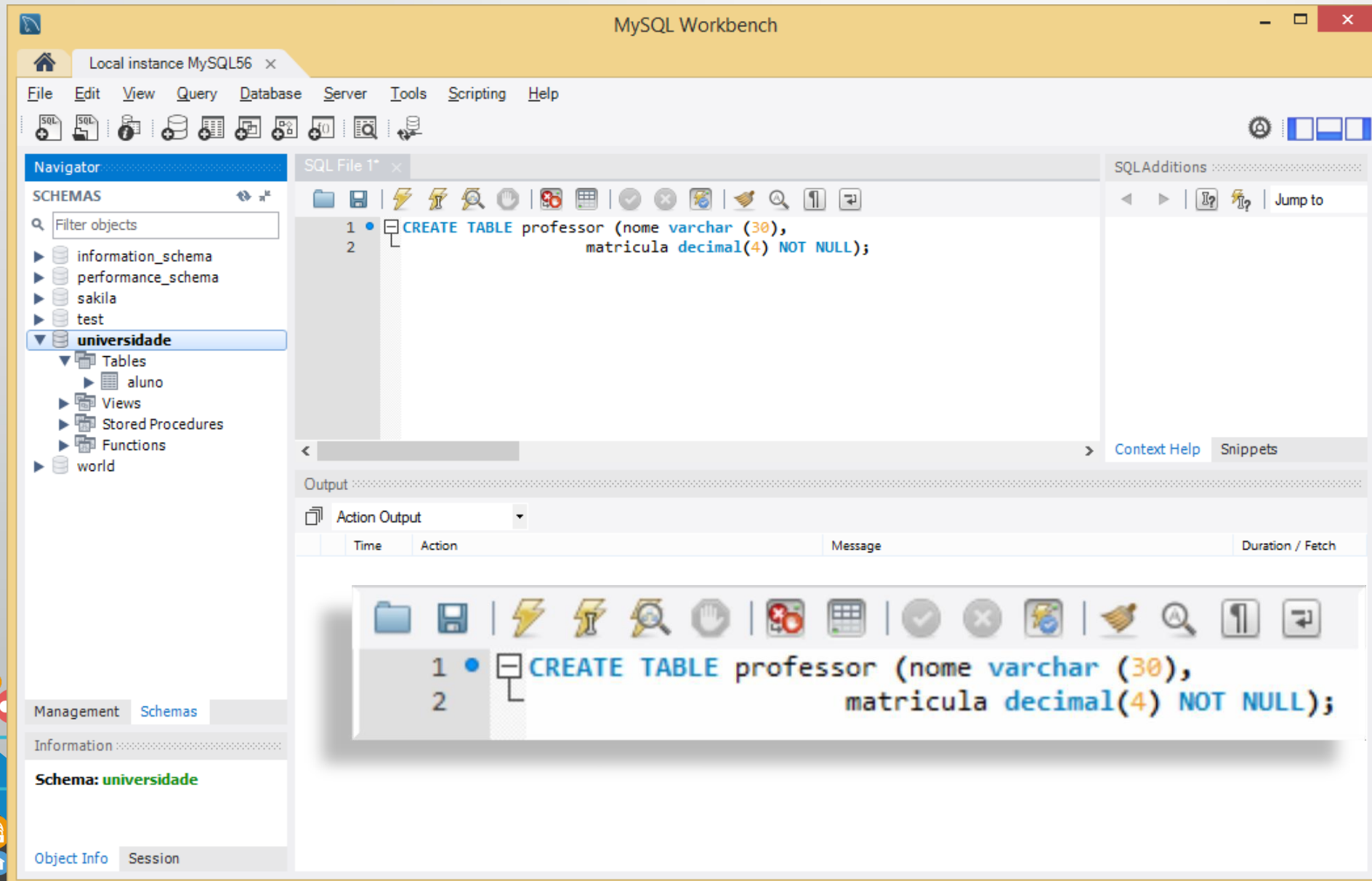
Time	Action	Message	Duration / Fetch
1 10:59:28	CREATE TABLE aluno (nome varchar (30), matricula decimal(4) NOT NULL, tipo_aluno integer, ...	0 row(s) affected	0.562 sec

1 10:59:28 CREATE TABLE aluno (nome varchar (30), matricula decimal(4) NOT NULL, tipo\_aluno integer,

# Exemplo - CREATE TABLE



# Exemplo – CREATE TABLE

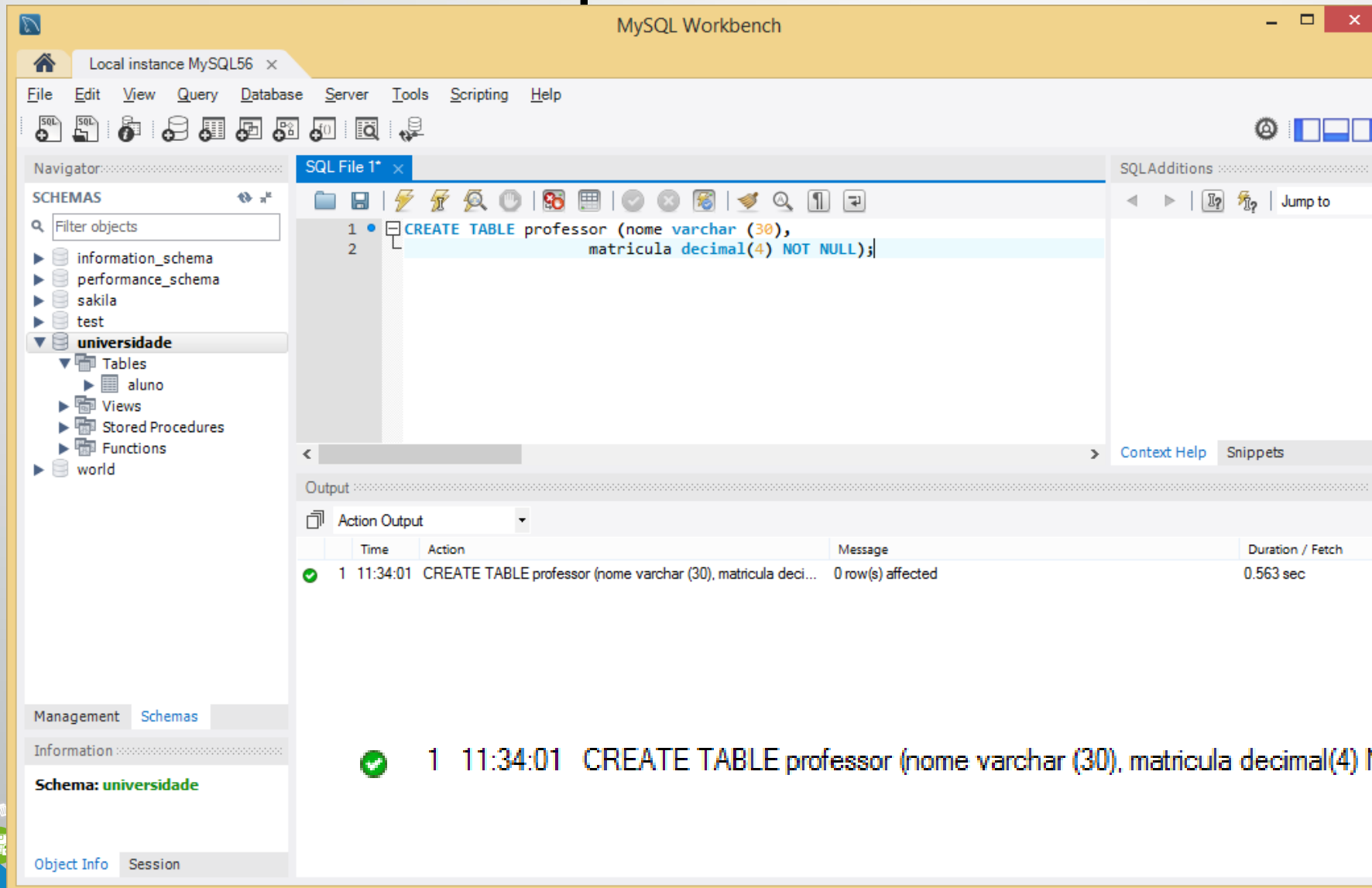


The screenshot displays the MySQL Workbench interface for a local instance of MySQL 5.6. The main window shows a SQL script for creating a table named 'professor' in the 'universidade' schema. The script is as follows:

```
1 CREATE TABLE professor (nome varchar (30),  
2                             matricula decimal(4) NOT NULL);
```

The interface includes a Navigator pane on the left showing the 'universidade' schema and its objects (Tables, Views, Stored Procedures, Functions). The bottom pane shows the 'Schema: universidade' information. The bottom status bar indicates 'Object Info' and 'Session'.

# Exemplo – CREATE TABLE



MySQL Workbench

Local instance MySQL56 x

File Edit View Query Database Server Tools Scripting Help

Navigator: SQL File 1\* x

SCHMAS

Filter objects

- information\_schema
- performance\_schema
- sakila
- test
- universidade**
  - Tables
    - aluno
  - Views
  - Stored Procedures
  - Functions
  - world

SQL File 1\* x

```
1 CREATE TABLE professor (nome varchar (30),  
2 matricula decimal(4) NOT NULL);
```

SQLAdditions

Context Help Snippets

Output

Action Output

	Time	Action	Message	Duration / Fetch
✓	1 11:34:01	CREATE TABLE professor (nome varchar (30), matricula deci...	0 row(s) affected	0.563 sec

Management Schemas

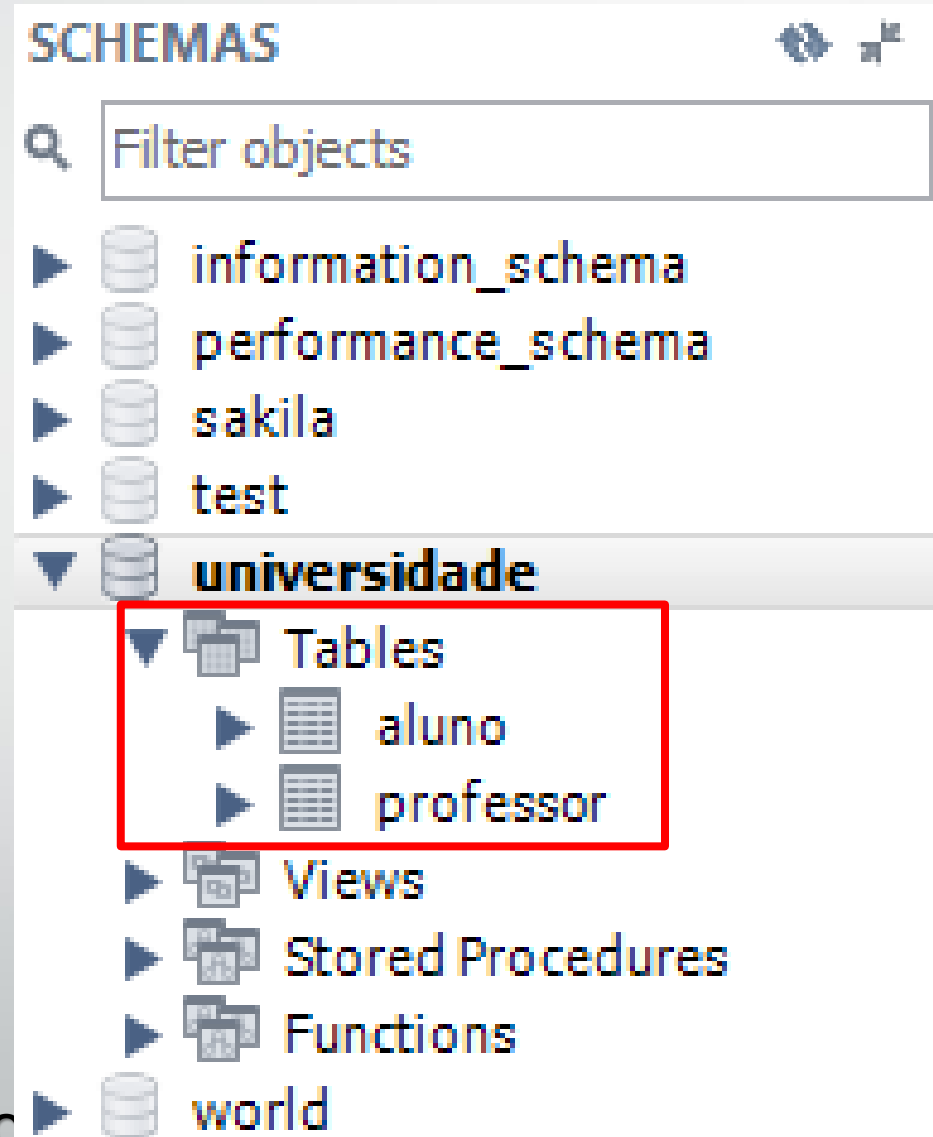
Information

Schema: **universidade**

Object Info Session

✓ 1 11:34:01 CREATE TABLE professor (nome varchar (30), matricula decimal(4) NOT NULL)

# Exemplo - CREATE TABLE



# O comando DROP

- O comando DROP pode ser usado para remover elementos nomeados do esquema, como tabelas, restrições.
- A sintaxe SQL usada é:

```
DROP TABLE tbl_name [, tbl_name] ...
```



# Exemplo – DROP TABLE

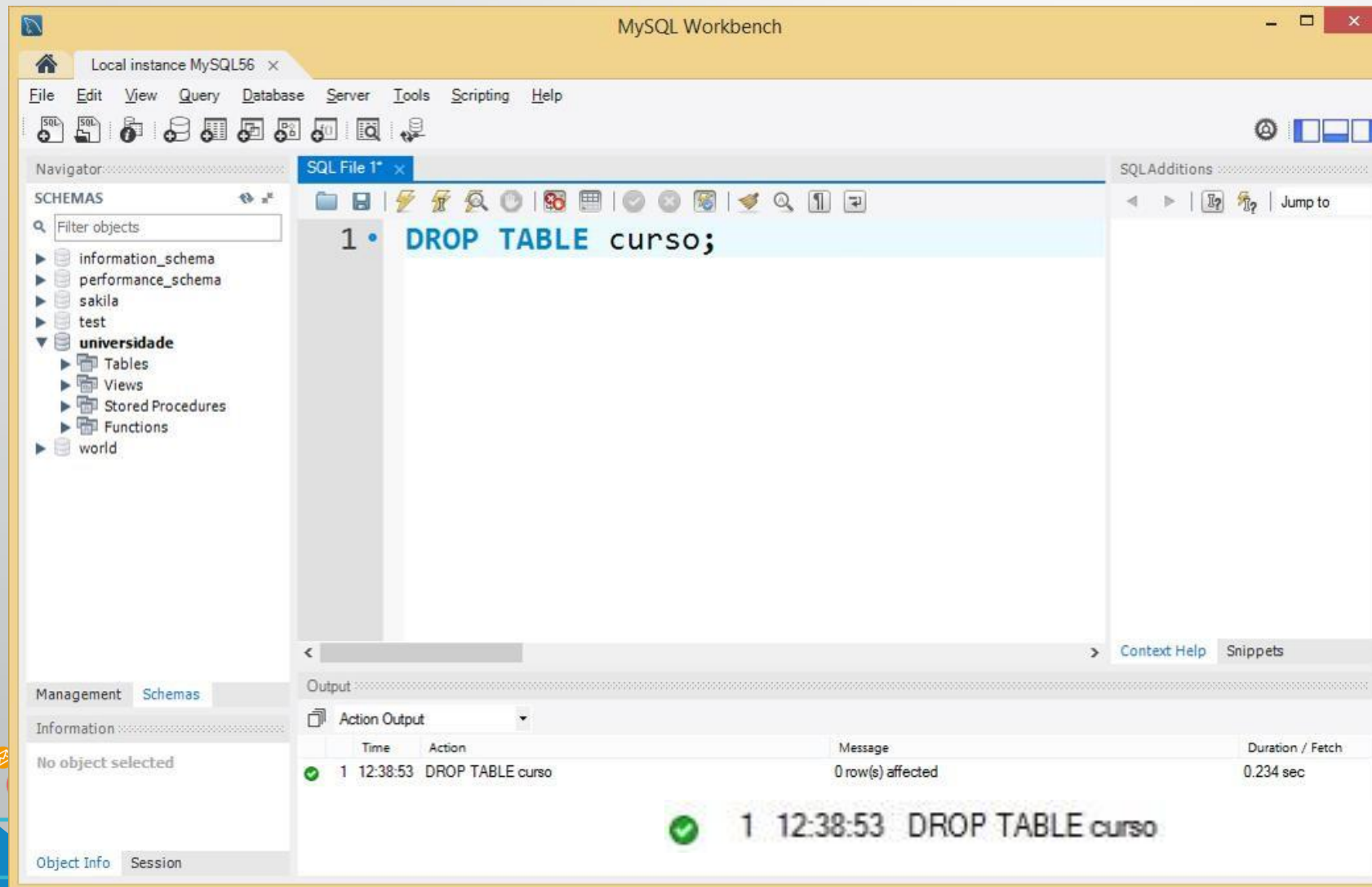
**DROP TABLE** professor;



	matricula	nome
	01	Gildo
	02	Miriã
	03	Tadeu
	04	Daves
	05	Hércules



# Exemplo – DROP TABLE

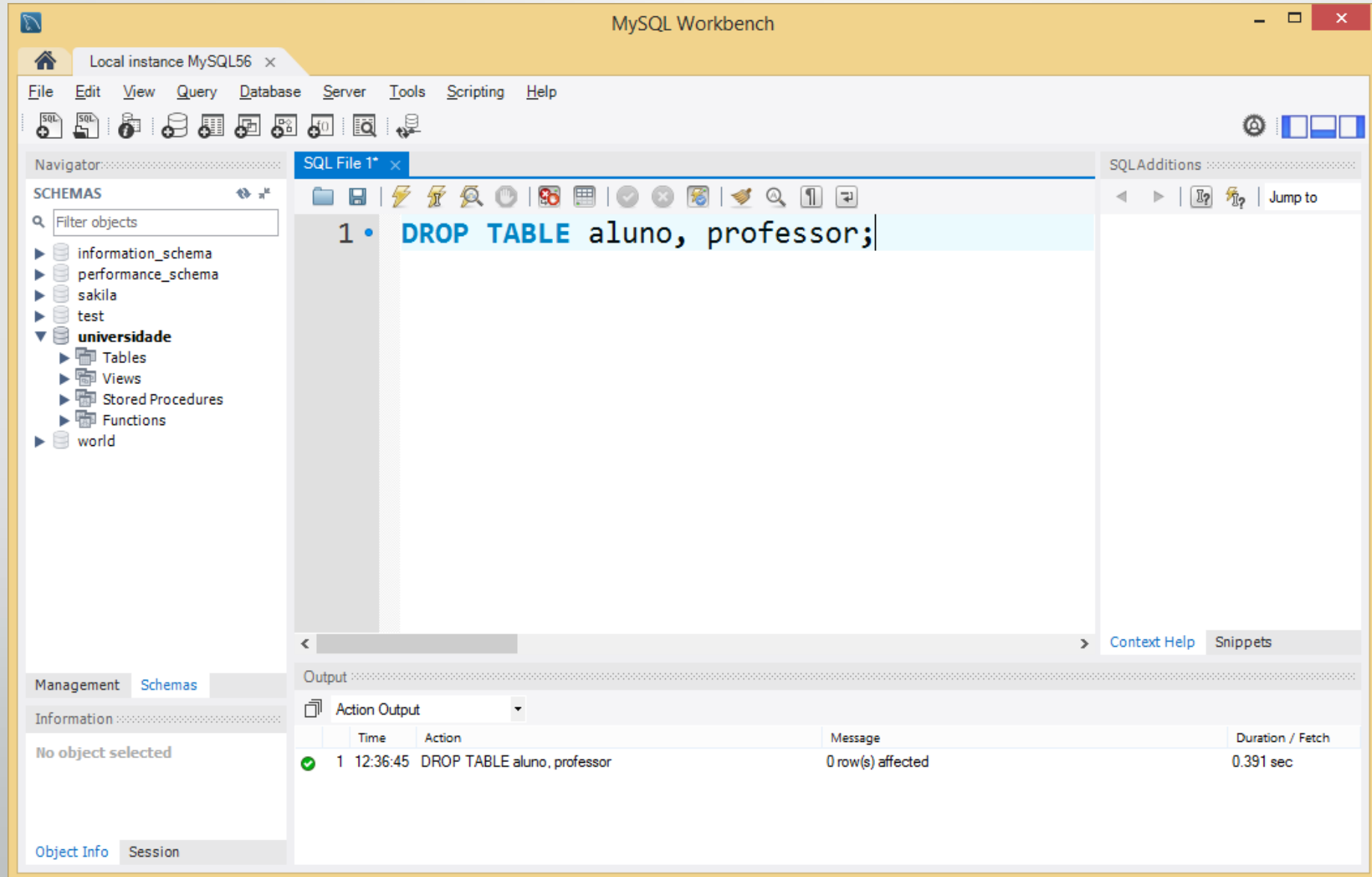


The screenshot shows the MySQL Workbench interface. The 'Navigator' pane on the left displays a tree of schemas, with 'universidade' expanded to show 'Tables', 'Views', 'Stored Procedures', and 'Functions'. The 'SQL File 1\*' editor in the center contains the query: `1 • DROP TABLE curso;`. The 'Output' pane at the bottom shows the execution results in a table format.

	Time	Action	Message	Duration / Fetch
✓	1 12:38:53	DROP TABLE curso	0 row(s) affected	0.234 sec

Below the screenshot, a green checkmark icon is followed by the text: `1 12:38:53 DROP TABLE curso` and `0 row(s) affected`.

# Exemplo – DROP TABLE



The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with the 'universidade' schema selected. The central editor window shows the SQL command: `DROP TABLE aluno, professor;`. The bottom output pane shows the execution results:

	Time	Action	Message	Duration / Fetch
✓	1 12:36:45	DROP TABLE aluno, professor	0 row(s) affected	0.391 sec

# O comando DROP

- Se for desejado excluir apenas os registros, mas deixar a definição de tabela para o uso futuro, então o comando DELETE deve ser usado no lugar de DROP TABLE.
- O comando DROP TABLE não apenas exclui todos os registros na tabela se tiver sucesso, mas também remove a definição de tabela do catálogo.



# Dúvidas?

