SQL DML, DDL & DATA TYPES

- Press `Space` to navigate through the slides
- Use `Shift+Space` to go back

Quick Recap

- `ORDER BY`
- Using **aggregate functions**
    - `COUNT` returns number of values in specified column.
    - `SUM` returns sum of values in specified column.
    - `AVG` returns average of values in specified column.
    - `MIN` returns smallest value in specified column.
    - `MAX` returns largest value in specified column.
- Group data using `GROUP BY` and `HAVING`
- `JOIN` tables together

Continuing with SQL

- Understand how to `create` tables
- How to update database using **DML** statements:
  - `INSERT`
  - `UPDATE`
  - and `DELETE` statements
- Understand **data types**

SQL

- **SQL DATA QUAERY LANGUAGE (DQL)**
  - **`SELECT` – we have focused on this to date**
- **SQL DATA MANIPULATION LANGUAGE (DML)**
- We would have to understand also the following **DML** verbs:
  - `INSERT` – adds new rows of data to the table
  - `UPDATE` – modifies existing data in the table
  - `DELETE` – removes rows of data from the table
- **DATA DEFINITION LANGUAGE (DDL)**
  - **`CREATE` – we will focus on this now**
  - `DROP`
  - `ALTER`

`CREATE TABLE`

- **Creates a table with one or more columns:**
  - Of the specified `dataType`.
  - With `NOT NULL`, system rejects any attempt to insert a null in the column.
  - Can specify a `DEFAULT` value for the column.
  - **Primary keys** should always be specified as `NOT NULL`.
  - `FOREIGN KEY` clause specifies **Foreign keys**

CREATE TABLE – BASIC SYNTAX

```
CREATE TABLE table_name(
   column1 datatype(size) NOT NULL,
   column2 datatype(size) DEFAULT 'ABC',
   column3 datatype(size),
   ...
   columnN datatype(size),
   PRIMARY KEY(one or more columns),
   FOREIGN KEY(columnN) REFERENCES table_nameX(foreign key of table_nameX)
);
```

`CREATE TABLE` CONSTRUCT – SIMPLE

- **A SQL relation is defined using the create table command:**

  `CREATE TABLE` R (A$_1$ D$_1$, A$_2$ D$_2$, ... , A$_N$ D$_N$ (integrity-constraint$_1$), ... ,(integrity-constraint$_k$))

  - **R** is the **name of the relation**
  - each **A$_i$** is an **attribute name** in the schema of relation **R**
  - **D$_i$** is the **data type** of values in the domain of attribute **A$_i$**

Example:

```
CREATE TABLE instructor (
ID char(5),
name varchar(20) not null,
dept_name varchar(20),
salary numeric(8,2),
PRIMARY KEY (ID));
```

- Select the cell below and press `Shift` + `Enter`:

In [3]:

```
%%sql sqlite://
--Drop the table if exists
DROP TABLE IF EXISTS instructor;

--Create the new table
CREATE TABLE instructor
    (ID_new             varchar(5),
     name           varchar(20) not null,
     dept_name      varchar(20),
     salary         numeric(8,2),
     primary        key (ID_new));

--Show all the columns of the new table
PRAGMA table_info(instructor);
```

```
Done.
Done.
Done.
```

Out[3]:

cid	name	type	notnull	dflt_value	pk
0	ID_new	varchar(5)	0	None	1
1	name	varchar(20)	1	None	0
2	dept_name	varchar(20)	0	None	0
3	salary	numeric(8,2)	0	None	0

INTEGRITY CONSTRAINTS IN CREATE TABLE

- **Constraints** let you define **rules for allowed values** in columns.
- Your **DBMS** uses these rules to enforce the integrity of information in the database automatically:
  - `NOT NULL`
  - `PRIMARY KEY` $(A_1, \ldots, A_n)$
  - `FOREIGN KEY` $(A_m, \ldots, A_n)$ references **R**
- **Example:** declare **ID** as the **primary key** for instructor

```sql
CREATE TABLE instructor (
ID char(5) NOT NULL,
name varchar(20) not null,
dept_name varchar(20),
salary numeric(8,2),
PRIMARY KEY (ID),
FOREIGN KEY (dept_name) references department(dept_name))
);
```

- Select the cell below and press `Shift` + `Enter`:

In [4]:

```
%%sql sqlite://
--Drop the tables if exist
DROP TABLE IF EXISTS instructor; DROP TABLE IF EXISTS department;

--New department table
CREATE TABLE department
    (dept_name      varchar(20),
     building       varchar(15),
     budget         numeric(12,2) check (budget > 0),
     primary key    (dept_name));

--New instructor table
CREATE TABLE instructor
    (ID             varchar(5),
     name           varchar(20) not null,
     dept_name      varchar(20),
     salary         numeric(8,2) CHECK (salary > 29000),
     primary        key (ID),
     foreign key    (dept_name) references department (dept_name)
        ON DELETE SET NULL
    );

PRAGMA foreign_key_list(instructor);
```

```
Done.
Done.
Done.
Done.
Done.
```

Out[4]:

id	seq	table	from	to	on_update	on_delete	match
0	0	department	dept_name	dept_name	NO ACTION	SET NULL	NONE

INTEGRITY CONSTRAINTS IN CREATE TABLE

- You can specify some **constraints** as either **column** or **table constraints**, depending on the context in which they are used.
- If a **primary key** contains one column, for example, you can define it as either a column constraint or a table constraint.
- If the **primary key** has two or more columns, you must use a table constraint.

DATA TYPES INTRODUCTION

- A **domain** is the set of valid values allowed in a column.
- To define a **domain**, you use a column's data type (and constraints).
- A **data type**, or **column type**, has these characteristics:
    - **Each column in a table has a single data type.**
    - The **data type determines a column's allowable values** and the operations it supports.
    - An **integer** data type can represent any whole number between certain DBMS-defined limits and supports the usual arithmetic operations: addition, subtraction, multiplication, and division (among others).
    - An **integer** can't represent a non-numeric value such as 'jack'
- The **SQL** standard leaves many data-type implementation details up to the DBMS vendor.

DOMAIN TYPES SUPPORTED IN MOST RDBMS

- **char(n)**
    - **Fixed length** character string, with user-specified length ***n***.
- **varchar(n)**
    - Variable length **character strings**, with **user-specified maximum length *n***.
- **int**
    - **Integer** (a finite subset of the integers that is machine dependent).
- **smallint**
    - Small **integer** (a machine-dependent subset of the integer domain type).

DOMAIN TYPES SUPPORTED IN MOST RDBMS

- **bigint**
    - The **bigint** data type is intended for use when integer values might exceed the range that is supported by the int data type.
- **numeric(p,d)**
    - Fixed point number, with user-specified precision of p digits, with n digits to the right of decimal point.
- **real, double precision**
    - Floating point and double-precision floating point numbers, with machine-dependent precision.

FURTHER NOTES ON DOMAIN TYPES

- **float(n)**
    - Floating point number, with user-specified precision of at least n digits.

- **char(n)**
    - Fixed length character string, with user-specified length ***n***.

- When you store a string with fewer than length characters in a `CHARACTER(length)` column, the **DBMS pads the end of the string with spaces to create a string that has exactly length characters**.
- A `CHARACTER(6)` string `'Jack'` is stored as `'Jack  '`, for example.
- `CHARACTER` and `CHAR` are synonyms.

FURTHER NOTES ON DOMAIN TYPES (2)

- **varchar(n)**
    - **Variable length character strings**, with user specified maximum length ***n***.

- Unlike `CHARACTER` , when you store a string with fewer than length characters in a `CHARACTER VARYING(length)` column, the **DBMS stores the string as is and doesn't pad it with spaces.**
- A `CHARACTER VARYING(6)` string `'Jack'` is stored as `'Jack'` , for example.
- `CHARACTER VARYING` , `CHAR VARYING` and `VARCHAR` are synonyms.

- More information on `CHAR` and `VARCHAR` can be found here (https://dev.mysql.com/doc/refman/5.7/en/char.html)

FURTHER NOTES ON DOMAIN TYPES (3)

EXACT NUMERIC DATA TYPES

- **Integer**
  - The minimum and maximum values that can be stored in an `INTEGER` are column-depend in any DBMS.
  - `INTEGER` takes no arguments.
  - `INTEGER` and `INT` are synonyms.
- **Integer** is an **exact numeric data types** to represent numerical values that:
  - Can be a **negative**, **zero** or **positive number**.
  - Is a **whole number** expressed without a decimal point: **–42**, **0**, **62262**.

FURTHER NOTES ON DOMAIN TYPES (4)

- **Integer** can be:
  - A normal-sized integer that can be **signed** or **unsigned**.
  - If signed, the allowable range is from **-2147483648** to **2147483647**.
  - If unsigned, the allowable range is from **0** to **4294967295**.
  - You can specify a width of up to **11** digits.
- **tinyint**
  - A very small integer that can be signed or unsigned.
  - If signed, the allowable range is from **-128** to **127**.
  - If unsigned, the allowable range is from **0** to **255**.
  - You can specify a width of up to **4** digits.
- Others include: **smallint**, **mediumint**, **bigint**, etc.

FURTHER NOTES ON DOMAIN TYPES (5)

EXACT NUMERIC DATA TYPES

- A **decimal number** has digits to the right of the decimal point:
    - **–22.06**, **0.0**, **0.0003**, **12.34**.
    - It has a fixed **precision** and **scale**.
    - **The precision is the number of significant digits used to express the number; it's the total number of digits both to the right and to the left of the decimal point.**
    - The **scale** is the number of **digits to the right of the decimal point.**
    - Obviously, the scale can't exceed the precision.
    - To represent a whole number, set the scale equal to **zero**.

INSERT, UPDATE, DELETE

- **SQL is a complete Data Manipulation Language that can be used for modifying the data in the database as well as querying the database**
- The commands for modifying the database are not as complex as the `SELECT` statement
- In this section, we describe the three **SQL** statements that are available to modify the contents of the tables in the database:
  - `INSERT` – adds new rows of data to a table
  - `UPDATE` – modifies existing data in a table
  - `DELETE` – removes rows of data from a table

INSERT

```
INSERT INTO TableName [(columnList)]
VALUES (dataValueList)
```

- `columnList` **is optional**
- If `columnList` omitted, **SQL** assumes a list of all columns in their original `CREATE TABLE` order.
- Any columns omitted must have been declared as `NULL` when table was created, unless `DEFAULT` was specified when creating column.
- **`dataValueList` must match `columnList` as follows:**
  - number of items in each list must be same;
  - must be direct correspondence in position of items in two lists;
  - data type of each item in `dataValueList` must be compatible with data type of corresponding column.

`INSERT` (EXAMPLE)

- Insert a new row into Staff table supplying data for all columns.

```
INSERT INTO department
VALUES ('Biology', 'Watson', '90000');
```

- Select the cell below and press `Shift` + `Enter`:

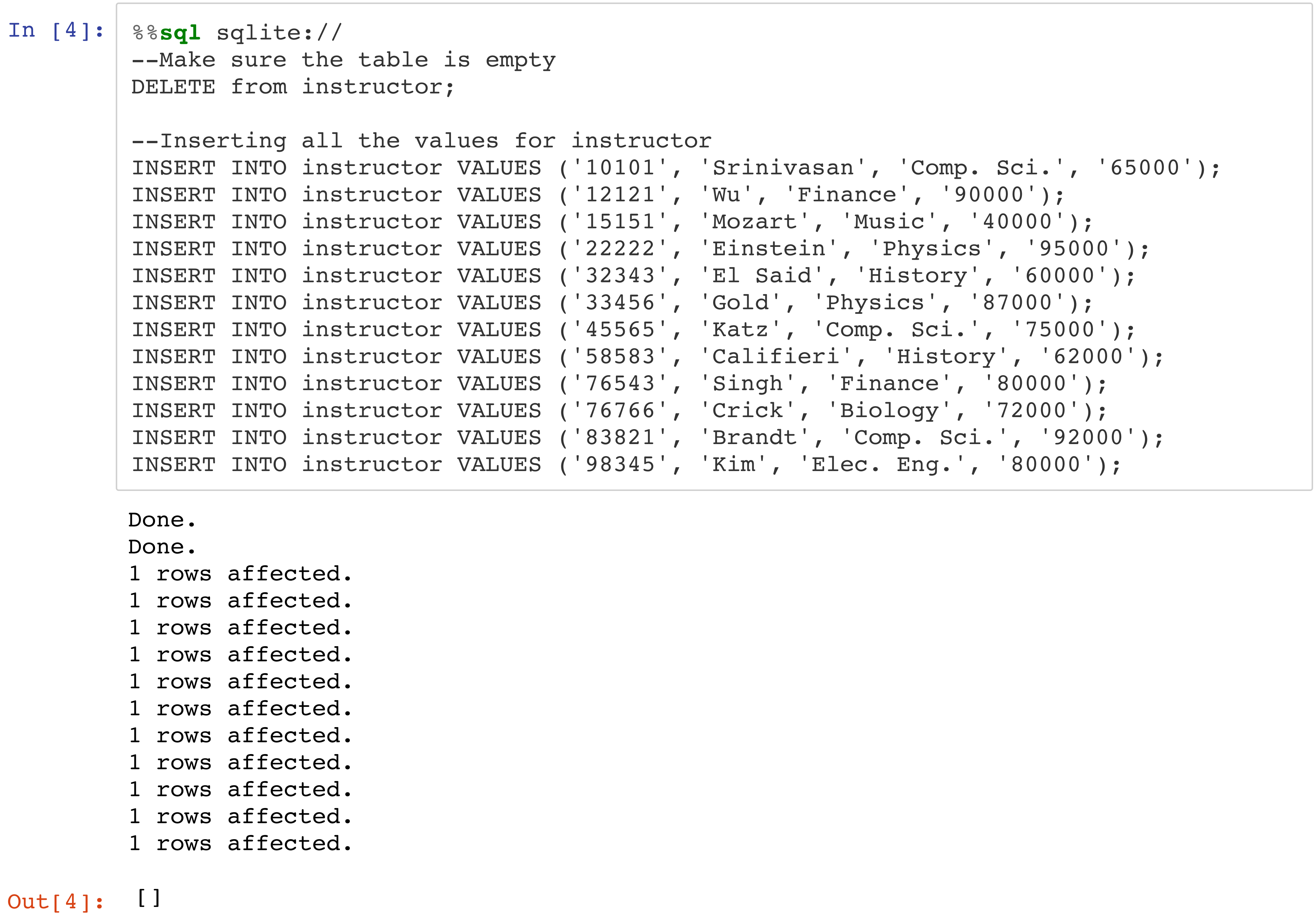
```
In [3]: %%sql sqlite://
        --Make sure the table is empty
        DELETE from department;

        --Inserting all the values for department and instructor
        INSERT INTO department VALUES ('Biology', 'Watson', '90000');
        INSERT INTO department VALUES ('Comp. Sci.', 'Taylor', '100000');
        INSERT INTO department VALUES ('Elec. Eng.', 'Taylor', '85000');
        INSERT INTO department VALUES ('Finance', 'Painter', '120000');
        INSERT INTO department VALUES ('History', 'Painter', '50000');
        INSERT INTO department VALUES ('Music', 'Packard', '80000');
        INSERT INTO department VALUES ('Physics', 'Watson', '70000');
```

```
Done.
Done.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
```

```
Out[3]: []
```

- Select the cell below and press `Shift` + `Enter`:



```
In [4]: %%sql sqlite://
        --Make sure the table is empty
        DELETE from instructor;

        --Inserting all the values for instructor
        INSERT INTO instructor VALUES ('10101', 'Srinivasan', 'Comp. Sci.', '65000');
        INSERT INTO instructor VALUES ('12121', 'Wu', 'Finance', '90000');
        INSERT INTO instructor VALUES ('15151', 'Mozart', 'Music', '40000');
        INSERT INTO instructor VALUES ('22222', 'Einstein', 'Physics', '95000');
        INSERT INTO instructor VALUES ('32343', 'El Said', 'History', '60000');
        INSERT INTO instructor VALUES ('33456', 'Gold', 'Physics', '87000');
        INSERT INTO instructor VALUES ('45565', 'Katz', 'Comp. Sci.', '75000');
        INSERT INTO instructor VALUES ('58583', 'Califieri', 'History', '62000');
        INSERT INTO instructor VALUES ('76543', 'Singh', 'Finance', '80000');
        INSERT INTO instructor VALUES ('76766', 'Crick', 'Biology', '72000');
        INSERT INTO instructor VALUES ('83821', 'Brandt', 'Comp. Sci.', '92000');
        INSERT INTO instructor VALUES ('98345', 'Kim', 'Elec. Eng.', '80000');
```

```
Done.
Done.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
```

```
Out[4]: []
```

- Select the cell below and press `Shift` + `Enter`:

In [5]:
```
%%sql sqlite://
SELECT * FROM department;
```

```
Done.
```

Out[5]:

dept_name	building	budget
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

In [8]:
```
%%sql sqlite://
SELECT * FROM instructor
WHERE ID='98345';
```

```
Done.
```

Out[8]:

ID	name	dept_name	salary
98345	Kim	Elec. Eng.	80000

INSERT USING DEFAULTS (EXAMPLE)

- **Insert a new row into Staff table supplying data for all mandatory columns.**

```
INSERT INTO Staff (staffNo,fName,lName,position,salary,branchNo)
VALUES('SG44','Anne','Jones','Assistant',8100,'B003');
```

Or

```
INSERT INTO Staff
VALUES('SG44','Anne','Jones','Assistant',NULL,NULL,8100,'B003');
```

- The values for **sex** and **DOB** are set to `NULL` in this case

AUTOINCREMENT FIELD

- **Autoincrement** allows a unique number to be generated automatically when a new record is **inserted** into a table.
- Often this is the **primary key** field that we would like to be created automatically every time a new record is inserted.
- The following **SQL** statement defines the `ID` column to be an auto-increment **primary key** field in the **person** table:

```
CREATE TABLE company(
   id INTEGER PRIMARY KEY AUTOINCREMENT,
   name           text      NOT NULL,
   age            int       NOT NULL,
   address        char(50),
   salary         real
);
```

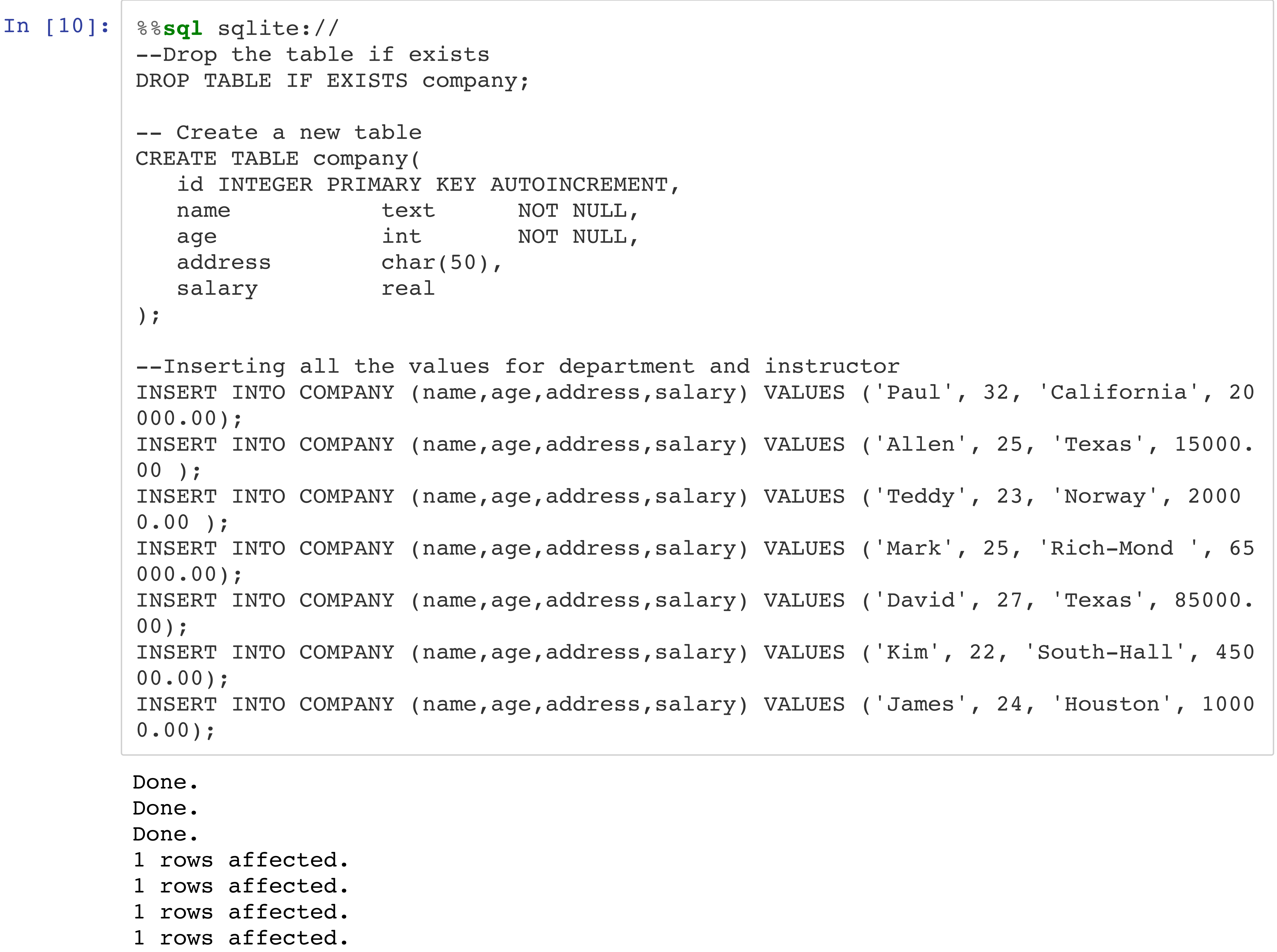
AUTOINCREMENT FIELD (EXAMPLE)

- To insert a new record into the **person** table, we will NOT have to specify a value for the `ID` column (a unique value will be added automatically):

```
INSERT INTO company (name,age,address,salary) VALUES ('Paul', 32, 'California', 20000.00);
```

- The **SQL** statement above would insert a new record into the **company** table.
- **The "ID" column would be assigned a unique value.**

- Select the cell below and press `Shift` + `Enter`:



In [10]:

```
%%sql sqlite://
--Drop the table if exists
DROP TABLE IF EXISTS company;

-- Create a new table
CREATE TABLE company(
   id INTEGER PRIMARY KEY AUTOINCREMENT,
   name           text      NOT NULL,
   age            int       NOT NULL,
   address        char(50),
   salary         real
);

--Inserting all the values for department and instructor
INSERT INTO COMPANY (name,age,address,salary) VALUES ('Paul', 32, 'California', 20
000.00);
INSERT INTO COMPANY (name,age,address,salary) VALUES ('Allen', 25, 'Texas', 15000.
00 );
INSERT INTO COMPANY (name,age,address,salary) VALUES ('Teddy', 23, 'Norway', 2000
0.00 );
INSERT INTO COMPANY (name,age,address,salary) VALUES ('Mark', 25, 'Rich-Mond ', 65
000.00);
INSERT INTO COMPANY (name,age,address,salary) VALUES ('David', 27, 'Texas', 85000.
00);
INSERT INTO COMPANY (name,age,address,salary) VALUES ('Kim', 22, 'South-Hall', 450
00.00);
INSERT INTO COMPANY (name,age,address,salary) VALUES ('James', 24, 'Houston', 1000
0.00);
```

```
Done.
Done.
Done.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
```

- Select the cell below and press `Shift` + `Enter`:

In [11]:

```
%%sql sqlite://
SELECT * FROM company;
```

```
Done.
```

Out[11]:

id	name	age	address	salary
1	Paul	32	California	20000.0
2	Allen	25	Texas	15000.0
3	Teddy	23	Norway	20000.0
4	Mark	25	Rich-Mond	65000.0
5	David	27	Texas	85000.0
6	Kim	22	South-Hall	45000.0
7	James	24	Houston	10000.0

`UPDATE`

- The `UPDATE` statement changes the values in a table's existing rows.
- You can use `UPDATE` to change:
    - **All rows** in a table
    - **Specific rows** in a table
- To update rows, you specify:
    - The **table** to update
    - The **names of the columns** to update and their new values
    - An optional **search condition** that specifies which rows to update

UPDATE

```
UPDATE TableName
SET columnName 1 = dataValue 1 [, columnName 2 = dataValue 2 ...]
[WHERE searchCondition]
```

- `TableNam` can be name of a base table or an updatable view.
- `SET` clause specifies names of one or more columns that are to be updated.

UPDATE

- `WHERE` clause is optional:
  - if **omitted**, named columns are updated for all rows in table;
  - if **specified**, only those rows that satisfy the `searchCondition` are updated.
- New **data values** must be compatible with the original **data types** for corresponding column.

`UPDATE` EXAMPLE

- Give all staff a **3% pay increase**, i.e. **all rows**

```
UPDATE company
SET salary = salary*1.03;
```

- Select the cell below and press `Shift` + `Enter`:

In [12]:

```
%%sql sqlite://
SELECT * FROM company;
```

```
Done.
```

Out[12]:

id	name	age	address	salary
1	Paul	32	California	20000.0
2	Allen	25	Texas	15000.0
3	Teddy	23	Norway	20000.0
4	Mark	25	Rich-Mond	65000.0
5	David	27	Texas	85000.0
6	Kim	22	South-Hall	45000.0
7	James	24	Houston	10000.0

In [13]:

```
%%sql sqlite://
UPDATE company
SET salary = salary*1.05;
SELECT * FROM company;
```

```
7 rows affected.
Done.
```

Out[13]:

id	name	age	address	salary
1	Paul	32	California	21000.0
2	Allen	25	Texas	15750.0
3	Teddy	23	Norway	21000.0
4	Mark	25	Rich-Mond	68250.0
5	David	27	Texas	89250.0
6	Kim	22	South-Hall	47250.0
7	James	24	Houston	10500.0

UPDATE EXAMPLE (2)

- Update certain rows only i.e. all **Texas** workers a 5% pay increase.

```
UPDATE company
SET salary = salary*1.05
WHERE address = 'Texas';
```

- Select the cell below and press `Shift` + `Enter`:

```
In [14]: %%sql sqlite://
         SELECT * FROM company;
```

```
Done.
```

Out[14]:

id	name	age	address	salary
1	Paul	32	California	21000.0
2	Allen	25	Texas	15750.0
3	Teddy	23	Norway	21000.0
4	Mark	25	Rich-Mond	68250.0
5	David	27	Texas	89250.0
6	Kim	22	South-Hall	47250.0
7	James	24	Houston	10500.0

```
In [15]: %%sql sqlite://
         UPDATE company
         SET salary = salary*1.05
         WHERE address = 'Texas';
         SELECT * FROM company;
```

```
2 rows affected.
Done.
```

Out[15]:

id	name	age	address	salary
1	Paul	32	California	21000.0
2	Allen	25	Texas	16537.5
3	Teddy	23	Norway	21000.0
4	Mark	25	Rich-Mond	68250.0
5	David	27	Texas	93712.5
6	Kim	22	South-Hall	47250.0
7	James	24	Houston	10500.0

UPDATE EXAMPLE (3)

- Update certain rows only i.e. specify Teddy's salary to a specific value.

```
UPDATE company
SET salary = '52000'
WHERE id = '3';
```

- Select the cell below and press `Shift` + `Enter`:

In [16]:

```
%%sql sqlite://
SELECT * FROM company;
```

```
Done.
```

Out[16]:

id	name	age	address	salary
1	Paul	32	California	21000.0
2	Allen	25	Texas	16537.5
3	Teddy	23	Norway	21000.0
4	Mark	25	Rich-Mond	68250.0
5	David	27	Texas	93712.5
6	Kim	22	South-Hall	47250.0
7	James	24	Houston	10500.0

In [17]:

```
%%sql sqlite://
UPDATE company
SET salary = '52000'
WHERE id = '3';
SELECT * FROM company;
```

```
1 rows affected.
Done.
```

Out[17]:

id	name	age	address	salary
1	Paul	32	California	21000.0
2	Allen	25	Texas	16537.5
3	Teddy	23	Norway	52000.0
4	Mark	25	Rich-Mond	68250.0
5	David	27	Texas	93712.5
6	Kim	22	South-Hall	47250.0
7	James	24	Houston	10500.0

UPDATE MULTIPLE COLUMNS

- Change Kim's (id="6") age and update the salary to 100000.

```sql
UPDATE company
SET salary = '100000', age = '23'
WHERE id = '6';
SELECT * FROM company;
```

- Select the cell below and press `Shift` + `Enter`:

In [18]:

```
%%sql sqlite://
SELECT * FROM company;
```

```
Done.
```

Out[18]:

id	name	age	address	salary
1	Paul	32	California	21000.0
2	Allen	25	Texas	16537.5
3	Teddy	23	Norway	52000.0
4	Mark	25	Rich-Mond	68250.0
5	David	27	Texas	93712.5
6	Kim	22	South-Hall	47250.0
7	James	24	Houston	10500.0

In [19]:

```
%%sql sqlite://
UPDATE company
SET salary = '100000', age = '23'
WHERE id = '6';
SELECT * FROM company;
```

```
1 rows affected.
Done.
```

Out[19]:

id	name	age	address	salary
1	Paul	32	California	21000.0
2	Allen	25	Texas	16537.5
3	Teddy	23	Norway	52000.0
4	Mark	25	Rich-Mond	68250.0
5	David	27	Texas	93712.5
6	Kim	23	South-Hall	100000.0
7	James	24	Houston	10500.0

DELETE

```
DELETE FROM TableName
[WHERE searchCondition]
```

- `TableName` can be name of a basetable or an updatable view.
- `searchCondition` is **optional**:
  - If **omitted**, then **all rows** are deleted from the table
- The `DELETE FROM TableName` statement does not delete table, but it's content
  - If `search_condition` is **specified**, only those **rows that satisfy a condition** are deleted.

- Select the cell below and press `Shift` + `Enter`:

In [20]:

```
%%sql sqlite://
SELECT * FROM company;
```

```
Done.
```

Out[20]:

id	name	age	address	salary
1	Paul	32	California	21000.0
2	Allen	25	Texas	16537.5
3	Teddy	23	Norway	52000.0
4	Mark	25	Rich-Mond	68250.0
5	David	27	Texas	93712.5
6	Kim	23	South-Hall	100000.0
7	James	24	Houston	10500.0

In [5]:

```
%%sql sqlite://
DELETE FROM company
WHERE id = '7';
SELECT * FROM company;
```

```
1 rows affected.
Done.
```

Out[5]:

id	name	age	address	salary
1	Paul	32	California	20000.0
2	Allen	25	Texas	15000.0
3	Teddy	23	Norway	20000.0
4	Mark	25	Rich-Mond	65000.0
5	David	27	Texas	85000.0
6	Kim	22	South-Hall	45000.0

DELETE SPECIFIC ROWS

- Delete all records from the **`company`** table.

```
DELETE FROM company;
```

- Select the cell below and press `Shift` + `Enter`:

```
In [22]:
%%sql sqlite://
SELECT * FROM company;
```

```
Done.
```

Out[22]:

id	name	age	address	salary
1	Paul	32	California	21000.0
2	Allen	25	Texas	16537.5
3	Teddy	23	Norway	52000.0
4	Mark	25	Rich-Mond	68250.0
5	David	27	Texas	93712.5
6	Kim	23	South-Hall	100000.0

```
In [23]:
%%sql sqlite://
DELETE FROM company;
SELECT * FROM company;
```

```
6 rows affected.
Done.
```

Out[23]:

id	name	age	address	salary

MORE ON `DELETE` AND `DROP`

- Delete all contents of the table, but retain its structure and the table itself:
    - `DELETE FROM company;`
- Delete the table `company` and all of it's contents:
    - `DROP TABLE company;`
- Use the `DROP TABLE` statement to completely remove a table `IF EXISTS` in the database
    - `DROP TABLE IF EXISTS company;`
- **Use `IF EXISTS` to prevent an error from occurring for tables that do not exist.**

- Select the cell below and press `Shift` + `Enter`:

In [24]:

```
%%sql sqlite://
SELECT * FROM company;
```

```
Done.
```

Out[24]:

id	name	age	address	salary

In [25]:

```
%%sql sqlite://
DROP TABLE IF EXISTS company;

-- Shows all the existing tables
SELECT name FROM sqlite_master WHERE type='table' AND name != 'sqlite_sequence';
```

```
Done.
Done.
```

Out[25]:

name
department
instructor

ALTER TABLE

- **Use the `ALTER TABLE` statement to modify a table definition by adding, altering or dropping columns and constraints.**

- **Despite the SQL standard, the implementation of ALTER TABLE varies greatly by DBMS:**

  - To determine what you can alter and the conditions under which alterations are allowed, search your DBMS documentation for `ALTER TABLE`.

  - Depending on your DBMS, some of the modifications that you can make by using `ALTER TABLE` are:

    - Add or drop a column

    - Alter a column's data type

    - Add, alter, or drop a column's default value or a nullability constraint

    - Add, alter, or drop column or table constraints such as primary-key, foreign-key, unique, and check constraints

    - Rename a column

    - Rename a table

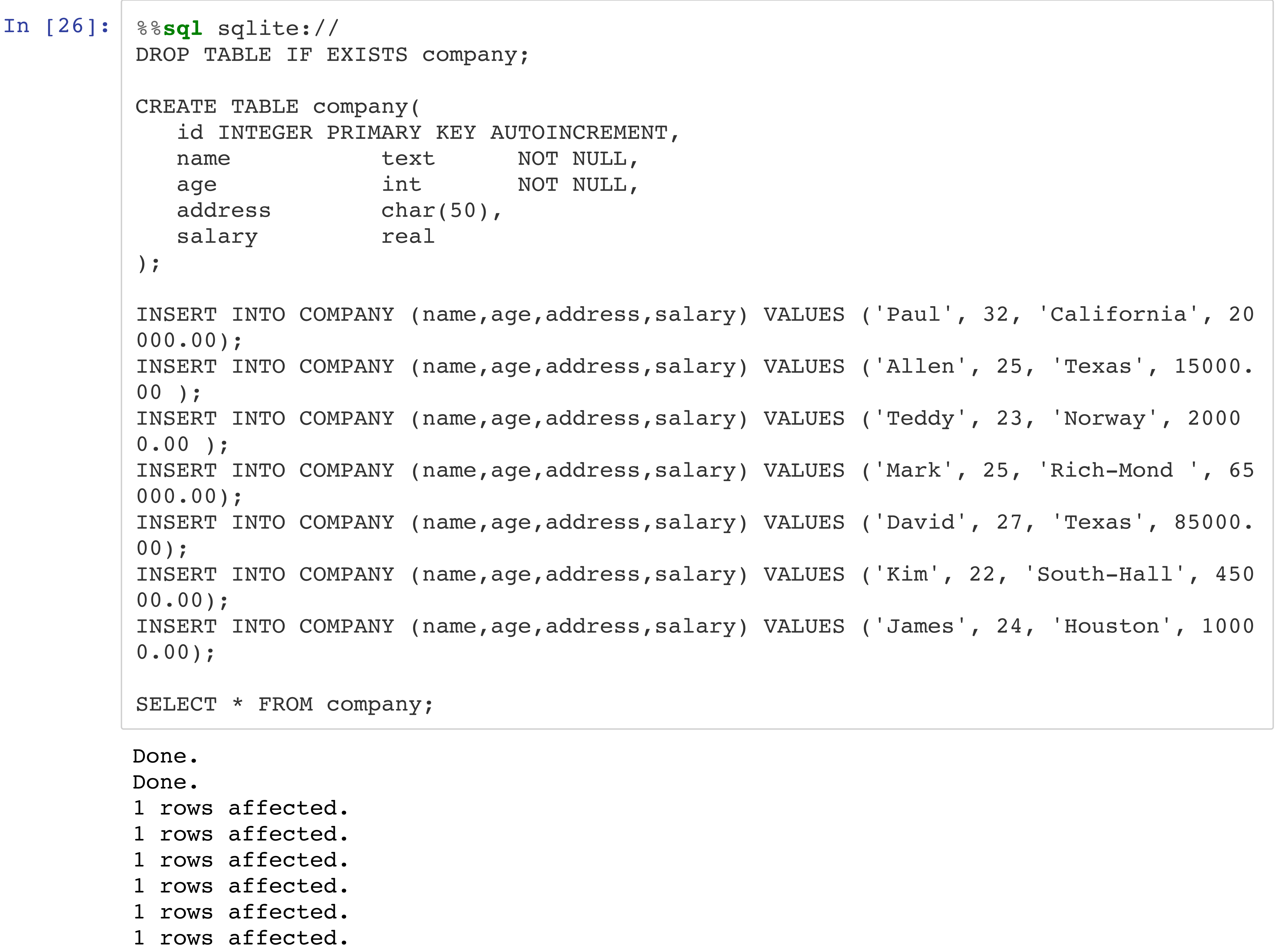
ALTER TABLE

- `ALTER TABLE R ADD A D`
  - Where **A** (column) is the name of the attribute to be added to relation **R** (table) and **D** is the domain of **A** (column).
  - All tuples in the relation are assigned **null** as the value for the new attribute.
- `ALTER TABLE R DROP A`
  - Where **A** (column) is the name of an attribute of relation **R** (table)
  - Dropping of attributes (columns) not supported by many databases
- `ALTER TABLE R RENAME old_attribute to new_attribute`
  - Renames an attribute (column)

`ALTER TABLE` EXAMPLE

```
ALTER TABLE company
RENAME TO employees;
```

- Select the cell below and press `Shift` + `Enter`:



```
In [26]: %%sql sqlite://
DROP TABLE IF EXISTS company;

CREATE TABLE company(
   id INTEGER PRIMARY KEY AUTOINCREMENT,
   name          text      NOT NULL,
   age           int       NOT NULL,
   address       char(50),
   salary        real
);

INSERT INTO COMPANY (name,age,address,salary) VALUES ('Paul', 32, 'California', 20
000.00);
INSERT INTO COMPANY (name,age,address,salary) VALUES ('Allen', 25, 'Texas', 15000.
00 );
INSERT INTO COMPANY (name,age,address,salary) VALUES ('Teddy', 23, 'Norway', 2000
0.00 );
INSERT INTO COMPANY (name,age,address,salary) VALUES ('Mark', 25, 'Rich-Mond ', 65
000.00);
INSERT INTO COMPANY (name,age,address,salary) VALUES ('David', 27, 'Texas', 85000.
00);
INSERT INTO COMPANY (name,age,address,salary) VALUES ('Kim', 22, 'South-Hall', 450
00.00);
INSERT INTO COMPANY (name,age,address,salary) VALUES ('James', 24, 'Houston', 1000
0.00);

SELECT * FROM company;
```

```
Done.
Done.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
```

In [27]:

```
%%sql sqlite://
ALTER TABLE company
RENAME TO employees;

-- Shows all the existing tables
SELECT name FROM sqlite_master WHERE type='table' AND name != 'sqlite_sequence';
```

```
Done.
Done.
```

Out[27]:

name
department
instructor
employees

Summary

- `CREATE TABLE`
    - **integrity constraints**
- **Data Types**
- `INSERT`
    - `AUTOINCREMENT`
- `UPDATE`
- `DELETE`
- `DROP`
- `ALTER TABLE`

TUTORIAL

1. `CREATE` a new table e.g `company` table
2. `INSERT` company employees into the table
3. `ALTER` the table attributes e.g. change a name of a column
4. `INSERT` a new employee
5. `UPDATE` the salary for this employee
6. `DELETE` that new employee
7. `DROP` the table `company`

In [28]:

```
%%sql sqlite://
DROP TABLE IF EXISTS company;
--Write your SQL starting after the next line:
--1

--2

--3

--4

--5

--6

--7
```

```
Done.
0 rows affected.
```

Out[28]:

```
[]
```