sql-intro-2

February 21, 2018

0.1 CREATE TABLE

Used to create a new table in a database.

Syntax

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

Option 1

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

Option 2

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
    constraint1,
    constraint2,
    ....
);
```

0.2 INSERT INTO

Used to insert new records in a table.

Syntax

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Example: Create "Product" table and insert some data

In this notebook we use SQL Magic for Python (https://github.com/catherinedevlin/ipython-sql)

There are several online options, for example https://sqliteonline.com/

```
In [1]: %load_ext sql
        # Connect to an empty SQLite database
        %sql sqlite://
```

```
Out[1]: 'Connected: None@None'
```

```
In [2]: %sql SELECT name FROM sqlite_master WHERE type='table';
```

```
Done.
```

```
Out[2]: []
```

```
In [3]: %%sql
        DROP TABLE IF EXISTS Product;
        CREATE TABLE Product (
            PName VARCHAR(255) NOT NULL PRIMARY KEY,
            Price FLOAT,
            Category VARCHAR(255),
            Manufacturer VARCHAR(255)
        );
        INSERT INTO Product VALUES ('Gizmo', 19.99, 'Gadgets', 'Gizmo Works');
        INSERT INTO Product VALUES ('Powergizmo', 29.99, 'Gadgets', 'Gizmo Works');
        INSERT INTO Product VALUES ('SingleTouch', 149.99, 'Photography', 'Canon');
        INSERT INTO Product VALUES ('MultiTouch', 203.99, 'Household', 'Hitachi');
```

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

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

```
In [4]: # Find tables in the database
        %sql SELECT name FROM sqlite_master WHERE type='table';
```

```
Done.
```

```
Out[4]: [('Product',)]
```

1 Single-Table Queries

In this section 1. The **SFW** Query 2. Other useful operators: **LIKE, DISTINCT, ORDER BY**

1.0.1 SQL Query

- Basic form --> **SFW** query

```
SELECT <attributes>
FROM <relations>
WHERE <conditions>
```

```
In [5]: %%sql  -- Whithout WHERE, the query returns all data in the table(s) defined in FROM
        SELECT *
        FROM Product;
```

```
Done.
```

```
Out[5]: [('Gizmo', 19.99, 'Gadgets', 'Gizmo Works'),
         ('Powergizmo', 29.99, 'Gadgets', 'Gizmo Works'),
         ('SingleTouch', 149.99, 'Photography', 'Canon'),
         ('MultiTouch', 203.99, 'Household', 'Hitachi')]
```

1.1 SELECT

Selection is the operation of filtering tuples in a relation on some condition
Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Example: Select all the tuples from "Product" with category 'Gadgets'

```
In [6]: %%sql  -- Tuples in 'Gadgets'
        SELECT *
        FROM Product
        WHERE Category = 'Gadgets';
```

```
Done.
```

```
Out[6]: [('Gizmo', 19.99, 'Gadgets', 'Gizmo Works'),
         ('Powergizmo', 29.99, 'Gadgets', 'Gizmo Works')]
```

1.1.1 Projection

The operation of producing an output table with tuples that have a subset of their prior attributes.

> **Example:** In the previous example, the attribute "Category" is redundant, remove it from the result.

```
In [7]: %%sql
        SELECT PName, Price, Manufacturer
        FROM Product
        WHERE Category = 'Gadgets';
```

```
Done.
```

```
Out[7]: [('Gizmo', 19.99, 'Gizmo Works'), ('Powergizmo', 29.99, 'Gizmo Works')]
```

Input schema: Product(PName, Price, Category, Manufacturer)
Output schema: Answer(PName, Price, Manufacturer)

1.1.2 To consider

- SQL **commands** are **NOT** case sensitive
- **Same:** SELECT, Select, select
- **Same:** Product, product
- **Values** are case sensitive
- **Different:** 'Seatle', 'seatle'
- __S__ingle quotes are for __S__trings, __D__ouble quotes are for __D__atabase identifiers

1.2 Other WHERE Clause Conditions

Equality * = (equal) * <> (not equal)
Range * > (greater than) * >= (greater than or equal to) * < (less than) * <= (less than or equal to) * BETWEEN
Membership * IN (matches values in a list or subquery)

1.3 LIKE

Simple String pattern matching
Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

There are two wildcards used in conjunction with the LIKE operator * % (percent) any sequence of characters * _ (underscore) any single character

```
In [8]: %%sql
        SELECT *
        FROM   Product
        WHERE  PName LIKE '%gizmo%';

Done.


Out[8]: [('Gizmo', 19.99, 'Gadgets', 'Gizmo Works'),
         ('Powergizmo', 29.99, 'Gadgets', 'Gizmo Works')]
```

1.4 DISTINCT

Removes duplicates from query results

Syntax

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

Example: Show categories in "Product"

```
In [9]: %%sql
        SELECT Category
        FROM   Product;

Done.


Out[9]: [('Gadgets',), ('Gadgets',), ('Photography',), ('Household',)]
```

Example: Show **distinct** categories in "Product"

```
In [10]: %%sql
         SELECT DISTINCT Category
         FROM Product;

Done.


Out[10]: [('Gadgets',), ('Photography',), ('Household',)]
```

1.5 ORDER BY

To sort results

Syntax

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

- Ties are broken by the second attribute on the ORDER BY list
- Default ordering is *Ascending*

  Example: Get entries from "Product" where category is not 'Gadgets' and price is below 50. Sort results by price and name.

```
In [11]: %%sql
         SELECT PName, Price, Manufacturer
         FROM Product
         WHERE Category != 'Gadgets' and Price > 50
         ORDER BY Price, PName

Done.


Out[11]: [('SingleTouch', 149.99, 'Canon'), ('MultiTouch', 203.99, 'Hitachi')]
```

```
In [12]: # Modify the css style
         # from IPython.core.display import HTML
         # def css_styling():
         #     styles = open("./style/custom.css").read()
         #     return HTML(styles)
         # css_styling()
```