Practical Session #12 - Basic SQL, Basic Python, Basic SQLite

Tom Mahler
In This Recitation We’ll Cover

- Basic SQL
- Basic Python
- Basic SQLite
Basic SQL
SQL - Structured Query Language

- We would like to have a system that stores data that will allow us to:
  - Store and access data without having to deal with the low level implementation details of storing it (like the syntax of XML files)
  - Access specific records without having to read entire files
  - Define relations between different "files"
SQL - Structured Query Language

- SQL is the standard way to interact with relational databases

- It will allow us to interact with a wide variety of database systems that solve those issues
  - Like MySQL, PostgreSQL, Microsoft SQL Server, and more

- SQL consists of two parts:
  - Data Definition Language - used to build tables, databases, and make general design decisions
  - Data Manipulation Language - used to manipulate data
SQL - Structured Query Language

- The first is used to define tables, including their layout, fields, what type each field has, what foreign keys they have

- The second is used to insert/update/delete data

- All data inserted with DML has to be 100% consistent with the former decisions of the DDL
Definitions

- **Table** - A table in the database, it holds only one kind of records

- In the following example each record (row) is comprised of (int, string, string)

<table>
<thead>
<tr>
<th></th>
<th>ID</th>
<th>Name</th>
<th>Office Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Majeed</td>
<td>Wed 10:00-12:00</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Boaz</td>
<td>Tue 16:00-18:00</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Matan</td>
<td>Tue 14:00-16:00</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Dan</td>
<td>Tue 12:00-14:00</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Hagit</td>
<td>Tue 16:00-18:00</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Tom</td>
<td>Tue 19:00-21:00</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Hussein</td>
<td>Tue 18:00-20:00</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Itay</td>
<td>Sun 18:00-20:00</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Morad</td>
<td>Wed 10:00-12:00</td>
<td></td>
</tr>
</tbody>
</table>
Definitions

- **Record** - a row from the table
  - This represents a single entry in a table

| 4 | Dan | Tue 12:00-14:00 |

- **Primary key** - A field that is unique in a table
  - The field $id$ is the primary key in this example
Definitions

- **Foreign key** - A "pointer" to another record in another table
  - This will allow us to define relations between tables

In the following table, the field **TA** is a foreign key, pointing at records in the TEACHING_ASSISTANTS table:

<table>
<thead>
<tr>
<th>TA</th>
<th>Group</th>
<th>Location</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>11</td>
<td>90-234</td>
<td>Sun 14-16</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>34-205</td>
<td>Sun 18-20</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>90-125</td>
<td>Thu 14-16</td>
</tr>
<tr>
<td>6</td>
<td>21</td>
<td>28-145</td>
<td>Thu 14-16</td>
</tr>
<tr>
<td>1</td>
<td>22</td>
<td>28-107</td>
<td>Thu 14-16</td>
</tr>
<tr>
<td>2</td>
<td>23</td>
<td>72-213</td>
<td>Thu 14-16</td>
</tr>
<tr>
<td>5</td>
<td>31</td>
<td>90-145</td>
<td>Thu 14-16</td>
</tr>
<tr>
<td>6</td>
<td>32</td>
<td>90-127</td>
<td>Thu 14-16</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>90-134</td>
<td>Thu 14-16</td>
</tr>
<tr>
<td>1</td>
<td>41</td>
<td>90-235</td>
<td>Thu 14-16</td>
</tr>
<tr>
<td>5</td>
<td>42</td>
<td>90-235</td>
<td>Thu 14-16</td>
</tr>
</tbody>
</table>
Data Definition Language

- The Data Definition Language (DDL) is used to create and destroy databases and database objects.

- These commands will primarily be used by database administrators during the setup and removal phases of a database project.
Create table

▶ Creates a tables in memory:
  ▶ Syntax

```
CREATE [TEMP] [CACHED|MEmORY|TEXT] TABLE name
( columnDefinition [, ...] )
```

columnDefinition:

```
column DataType [ [NOT] NULL] [PRIMARY KEY]
```

DataType:

```
{ INTEGER | DOUBLE | VARCHAR | DATE | TIME | ... }
```
Create table

Example

```sql
CREATE TABLE STORE ( columnDefinition [, ...] )

(id INTEGER, Name VARCHAR(30), Type VARCHAR(30))
```
A primary key is used to uniquely identify each row in a table.

It can either be part of the actual record itself, or it can be an artificial field (one that has nothing to do with the actual record).
Primary Key

- A primary key can consist of one or more fields on a table

- When multiple fields are used as a primary key, they are called a composite key
Primary Key

Example:

```
CREATE TABLE address
(id INTEGER, Street VARCHAR(50), City VARCHAR(30),
 HouseNumber INTEGER, ZipCode INTEGER,
 primary key (id))
```
Example:

- The part $primary key (id)$ defines the field $id$ to be unique in the whole table.
- Meaning if we have one field where $id=5$, then no other record in that table can have the value 5 in the id field.
Primary Key

- Try inserting the same record twice to 'address' table

- Can this succeed?

- What do you expect?
Foreign Key

- A foreign key is a field (or fields) that points to the primary key of another table.

- The purpose of the foreign key is to ensure referential integrity of the data.

- In other words, only values that are supposed to appear in the database are permitted.
Foreign Key

- For example, say we have two tables, like before
  - A Teaching Assistant table, and a PRACTICAL_SESSIONS table

- In the practical session you may insert only TAs that are already in the TEACHING_ASSISTANTS table

- In this case, we will place a foreign key on the PRACTICAL_SESSIONS table and have it relate to the primary key of the TEACHING_ASSISTANTS table
Foreign Key

- This way, we can ensure that all practical sessions in the PRACTICAL_SESSIONS table are related to a TA in the Teaching Assistant table.

- In other words, the PRACTICAL_SESSIONS table cannot contain information on a TA that is not in the Teaching Assistant table.
The structure of these two tables will be as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>type</th>
<th>characteristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>INTEGER</td>
<td>Primary Key</td>
</tr>
<tr>
<td>Name</td>
<td>VARCHAR(50)</td>
<td></td>
</tr>
<tr>
<td>Office hours</td>
<td>VARCHAR(9)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>type</th>
<th>characteristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA</td>
<td>INTEGER</td>
<td>Foreign Key</td>
</tr>
<tr>
<td>Group</td>
<td>INTEGER</td>
<td>Primary Key</td>
</tr>
<tr>
<td>Location</td>
<td>VARCHAR(50)</td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>VARCHAR(9)</td>
<td></td>
</tr>
</tbody>
</table>
In the above example, the TA column in the PRACTICAL_SESSIONS table is a foreign key pointing to the ID column in the TEACHING_ASSISTANTS table.
Foreign Key

- Syntax:

```sql
CREATE TABLE TEACHING_ASSISTANTS
(ID INTEGER,
Name VARCHAR(50),
OfficeHours VARCHAR(9),
Primary Key (ID),
)"
```
Foreign Key

Syntax:

CREATE TABLE PRACTICAL_SESSIONS
(TA INTEGER,
Group INTEGER,
Location VARCHAR(50),
Time VARCHAR(9),
Primary Key (Group),
Foreign Key (TA) references TEACHING_ASSISTANTS(ID))
Data Manipulation Language

- The Data Manipulation Language (DML) is used to retrieve, insert and modify database information.

- These commands will be used by all database users during the routine operation of the database.
Insert

- Adds one or more new rows of data into a table

- Syntax:

  INSERT INTO table [ (column [, ...] ) ]
  { VALUES(Expression [, ...]) | SelectStatement }

- Write a statement that adds a row to the PRACTICAL_SESSIONS table
Delete

- Removes rows in a table (And not the table itself!)

- Syntax:

  DELETE FROM table [WHERE Expression]

- Delete one of the records you just added
The UPDATE command can be used to modify information contained within a table, either in bulk or individually.

Syntax:

```
UPDATE table SET column = Expression [, ...] [WHERE Expression]
```
Example: Our university decided to add an additional practical session to Morad

The following SQL command would do that:

```
INSERT INTO PRACTICAL_SESSIONS (TA, Group, Location, Time)
VALUES (9, 43, "72-218", "Sun 10-12")
```
Update

Later there turns out to be a mistake and the Time should be “Thu 10-12“

The following SQL command corrects that:

```
UPDATE PRACTICAL_SESSIONS
SET Time = "Thu 10-12"
WHERE Group = 43
```
Simple Select

- The SELECT command is the most commonly used command in SQL

- It allows database users to retrieve the specific information they desire from an operational database
Simple Select

Syntax:

SELECT [DISTINCT]
{ selectExpression | table.* | * } [, ... ]
[INTO [CACHED|TEMP|TEXT] newTable]
FROM tableList
[WHERE Expression]
[GROUP BY Expression [, ...] ]
[ORDER BY selectExpression [{ASC | DESC}] [, ...] ]
[LIMIT n m]
[UNION [ALL] selectStatement]
Simple Select

- The command shown below retrieves all of the information contained within the CUSTOMER table.

- Note that the asterisk is used as a wildcard in SQL.

- This literally means "Select everything from the TEACHING_ASSISTANT table."

SELECT *
FROM TEACHING_ASSISTANT
Simple Select

- Alternatively, users may want to limit the attributes that are retrieved from the database

- For example, you may require a list of the last names of all TAs

- The following SQL command would retrieve only that information:

```
SELECT Name
FROM TEACHING_ASSISTANT
```
Finally, the WHERE clause can be used to limit the records that are retrieved to those that meet specified criteria.

We might be interested in reviewing the records of TAs that have office hours at Sunday.
The following command retrieves all of the data contained within TEACHING_ASSISTANT for records that have Office hours at Sunday.

SELECT *
FROM TEACHING_ASSISTANT
WHERE OfficeHours LIKE 'Sun%'

Simple Select
Join Operation

- Now that we have seen a simple select operation from a single table, we would like to access information across a few tables

- The join action will allow us to do so
Join Operation

The following code will retrieve all a Cartesian product of the information from TEACHING_ASSISTANTS and PRACTICAL_SESSIONS

```
SELECT *
FROM TEACHING_ASSISTANTS,PRACTICAL_SESSIONS
```

Most of the time we would like to use the join operation to connect related information from given tables.
Join Operation

- For example both TEACHING_ASSISTANTS and PRACTICAL_SESSIONS have a column referring to the ID of a given teacher assistant.

- The following code will retrieve a table containing the name of each TA and his practical session time.
Join Operation

```
SELECT ta.name, ps.group, ps.Location, ps.Time
FROM TEACHING_ASSISTANTS AS ta
JOIN PRACTICAL_SESSIONS AS ps
ON ta.id = ps.TA
```

- The ON operation lets us choose what is the connection between the two tables we would like to connect on
Join Operation

- Multiple attributes can be compared by using the AND OR keyword between comparisons

- The AS keyword can be used to give a temporary name to a table or selected attribute

- The basic JOIN operation (also called inner join) will ignore any line from TEACHING_ASSISTANTS that does not fit any line in PRACTICAL_SESSIONS
Join Operation

- For example if TEACHING_ASSISTANTS had another TA that had no PRACTICAL_SESSIONS, he would not appear in an inner join.

- We can force the join to ignore such lines and show them any way (with NULL values in the values concerning PRACTICAL_SESSIONS columns) by using the LEFT JOIN.
SQL Injection
Basic Python
What is python?

- Python is an open-source, general purpose programming language that is dynamic, strongly-typed, object-oriented, functional, and memory-managed.

- Python is an interpreted language, meaning that it uses an interpreter to translate and run its code.
What is python?

- The interpreter reads and executes each line of code one at a time, just like a script, and hence the term “scripting language”

- Python is dynamic which means that the types are checked only at runtime

- But Python is also strongly typed, just like Java
What is python?

- You can only execute operations that are supported by the target type

- Note that this means that variable names do not have a type in Python, their values do
Coding Python

- Python source files use the ".py" extension and are called "modules"

- With a Python module hello.py, the easiest way to run it is with the shell command `python hello.py Morad` which calls the Python interpreter to execute the code in hello.py, passing it the command line argument "Morad", Command line arguments will reside in the `argv` structure of the standard module `sys`
Examples

- See code... (See spl12.py)
Special variables

- Notice the part if __name__ == '__main__': main()

- Remember that we said that Python is a 'scripting' language

- If we omit this part, then python will execute the code line by line in a top-bottom manner

- But since we want a definitive start point, we can use __name__ and compare it to '__main__'
Special variables

- Before executing the code, python will define a few special variables

- For example, if the python interpreter is running that module (the source file) as the main program, it sets the special `__name__` variable to have a value "__main__"

- If this file is being imported from another module, `__name__` will be set to the module's name
One Python feature is that the whitespace indentation of a piece of code affects its meaning.

A logical block of statements such as the ones that make up a function should all have the same indentation, set in from the indentation of their parent function or "if" or whatever.

If one of the lines in a group has a different indentation, it is flagged as a syntax error.
String literals can be enclosed by either double or single quotes, although single quotes are more commonly used:

myStr = 'hello.'
mySecondStr = "hello."
Strings

- Python strings are "immutable" - they cannot be changed after they are created

- For example:

```python
str1 = 'hello'
str2 = str1 + ' world'
```
Strings

- In the above code, the expression `str2 = str1 + ' world'` takes the two strings `str1` and 'world' and builds a new string out of them.

- We can use the `print` function to print to the screen.

- We can also use the `str(_)` function to convert non-string variables to string.
Examples

- See code...
  (See spl12.py)
Strings

- There are a lot of helpful string methods:
  - `s.lower()` - returns the lowercase version of a string
  - `s.upper()` - returns the upper version of a string
  - `s.strip()` - returns a string with the whitespaces removed from the start to the end
Strings

There are a lot of helpful string methods:

- `s.replace('old', 'new')` - returns a new string where all occurrences of 'old' have been replaced by 'new'
- `s.split('delim')` - returns a list of substrings separated by the given delimiter
  - For example: 'aaa,bbb,ccc'.split(',') gives ['aaa', 'bbb', 'ccc']
Strings

- Python also has a \printf()-like facility to put together a string: The \% operator

```
text = "SPL %d is the most awesome course ever. %s and I take it this year!" % (171, 'Mary')
```
Examples

- See code...
  (See spl12.py)
'If' statement

- Notice the colon at the end of the if, elif, else lines
- `elif` is basically `else if` in JAVA.

- Notice also the operators `or` and `and` - these are the logical operators in Python.
Python Lists

- Lists are written within square brackets []

- Lists work similarly to strings - use the len() function and square brackets [] to access data

- As in JAVA, the first element is at index 0

- Yes, you can mix types in a list in Python
  - But that is not recommended for your own type-safety
Examples

- See code...
  (See spl12.py)
List methods

- `list.append(elem)` - adds a single element to the end of the list. Does not return the new list, just modifies the original.

- `list.insert(index, elem)` - inserts the element at the given index, shifting elements to the right.

- `list.remove(elem)` - searches for the first instance of the given element and removes it.
List methods

- `list.extend(list2)` - adds the elements in `list2` to the end of the list
  - You can also use `+` or `+=` on a list for the same effect

- `list.sort()` - sorts the list in place (does not return it)

- `list.pop(index)` - removes the element at the given index
For, in, while, if in

- There are no loops of the syntax for $i=0;i<\ldots;\ldots$ in Python

- Next, we show how we run on the elements of a list and how you check if an element is a member of a list

- Notice that python does not have the support for the syntax $i++$
  - Only the $i = i + \text{someNumber}$ or $i += \text{someNumber}$ syntax is accepted
Examples

See code...
(See spl12.py)
List and String slicing

- In Python, lists and strings support slicing: it is a useful way to refer to sub-parts of sequences.

- \texttt{s[x:y]} is the elements of \texttt{s} starting at \texttt{index x} until \texttt{index y} \textbf{EXCLUDING} element at index \texttt{y}. 
List and String slicing

- If $x$ is omitted, then it assumes you want the elements starting from the first element until index $y$ exclusive.

- If $y$ is omitted, then it assumes you want the elements from index $x$ until the end of $s$.

- Python also supports negative indexes in slicing.
  - If the string is $s = \text{"Morad"}$, then $s[-1]$ is $d$, $s[-2]$ is $a$, etc..
Examples

See code... (See spl12.py)
Handling standard user input

- We can use the strong `sys` module’s `stdin` in order to infer user input

- Here is an echo example, that exits whenever we type `exit`
Examples

► See code...
(See echo.py)
File operations

- We can open a file given the filename using `open(filename)`

- Then we can run on each line easily

- Here is a short example that opens a file given in the command-line arguments
Examples

- See code...
  (See file_example.py)
The `with` keyword states that Python should automatically close the resources for us (i.e., the filestream) when:

- An exception occurs
- The scope ends
  - In this case, the scope is the whole program run
File operations

- It is very similar to the try-with-resources concept of Java

- A useful method that checks whether a file exists is:

  ```python
  import os
  os.path.isfile(filename)
  ```
Tuples and Dicts

Tuples and Dicts are very strong structures in Python which we are interested in.

Read about tuples here:
  - https://www.tutorialspoint.com/python/python_tuples.htm

Read about dicts here:
  - https://www.tutorialspoint.com/python/python_dictionary.htm
Basic SQLite

- SQLite is a library that provides a lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language.

- There exists a module called sqlite3 that provides an SQLite interface for Python.
Connection and access

- To use the module we must first create a connection to the database

- If the given database name does not exist, it will be created

```python
import sqlite3
dbcon = sqlite3.connect('example.db')
```
Connection and access

- Once we have a connection, we would like to execute commands.

- In order to be able to do so, we must create a cursor object.

- A cursor object is an object that lets you traverse records in a database.
Connection and access

- Cursors facilitate subsequent processing in conjunction with the traversal, such as retrieval, addition and removal of database records
Examples

- See code...
  (See sql_example.py)
Executing commands

Now that we have a cursor, we can execute SQL commands

Notice that (1, 'Morad',) is a Python tuple

Note the syntax of ?; these are fields that are replaced one by one using the given tuple
Queries with results

As we have seen, some SQL queries yield result(s), like the SELECT query

These can be fetched from the cursor after executing the query
Queries with results

- `cursor.fetchone()`
  - Fetches one of the results of the query in the form of a tuple, or None if none is available.

- `cursor.fetchall()`
  - Fetches all of the results of the query, returning a list of tuples.
Examples

▶ See code...
  (See sql_example.py)
Notes:

1. In order not to try to create the tables and put the entries each time, we checked if the file existed
   - If it did, we know that we have created the database, the tables, and entered the data in the past, so we don't need to do that again
   - This was achieved using `os.path.isfile`
Notes:

2. Note how the last example gave us None because no student exists with id 5
Notes:

3. Why are we so sure that we should use fetchone and not fetchall in the second and third calls to the db?

   Because we select rows by the field ID which is a primary key, so we are sure it is unique, so at most one row exists with such an ID.