SIM Global Education

Block 3
Introduction to Relational Database Management Systems

Learning Outcomes

After completing this topic and the recommended reading, you should be able to:

- Use relational database models and structured query languages (SQL).
- Gain experience with interfacing SQL from R and Python.

1. Database

Database

- An organised collection of structured information or data.
- Stored electronically in a computer system.

Relational Databases

- A type of database that stores and provides access to data points that are related to one another.
- Store data in structured, tabular form.
- The columns of the table hold **attributes** of the data.
- The rows of the table hold **records** (**tuples**), and each record usually has a value for each attribute.

Nonrelational Databases

- Often called **NoSQL** databases.
- Store data in unstructured and semi-structured, non-tabular form.
- Graph database
  - Stores data in terms entities (nodes) and relationships (edges) between entities.
- Document-oriented database
  - Stores data in the form of JSON-like documents.
  - Stores all information for a given object in a single instance in the database.
  - Every stored object can be different from another.
- Examples:
  - *MongoDB*: uses JSON-like documents with optional schemas.

Database Management System (DBMS)

- A database is usually controlled by a DBMS.
- The data can then be easily accessed, managed, modified, updated, controlled and organised.

Relational Database Management System (RDBMS)

- A system used to maintain relational databases.
- Stores data in a row-based table structure which connects related data elements.
- Using **Structured Query Language (SQL)** to access the database.
- Includes functions that maintain the security, accuracy, integrity and consistency of the data.

RDBMS Terminology

- **Relation (Table)**
  - Collection of rows and columns.
  - Each table usually represent an entity.

- **Attribute (Column)**
  - Each attribute has a type or domain.

- **Tuple (Row)**
  - Each tuple represent a record, a set of attribute values.

- **Schema**
  - The description on how the database table is constructed.

- **Primary Key**
  - The attribute which is the unique identifier (**ID**) for each tuple/record.
  - The set of attributes whose combine values are unique.
  - Cannot have null values.

- **Foreign Key**
  - The primary key that is used in another table that provides a link/relationship between data in two tables.

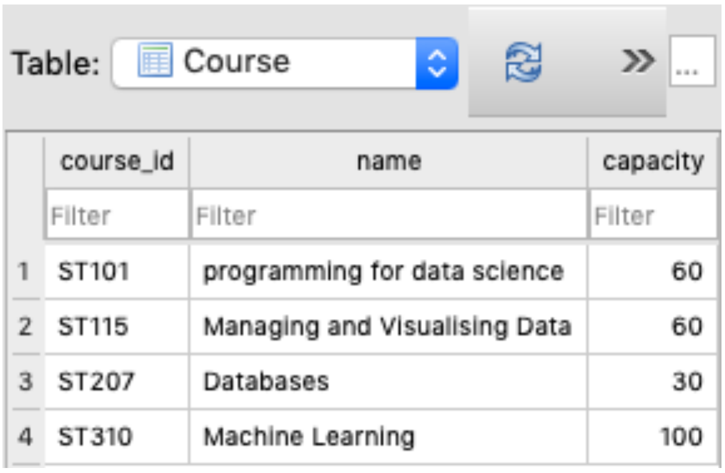


Table: Course

	course_id	name	capacity
1	ST101	programming for data science	60
2	ST115	Managing and Visualising Data	60
3	ST207	Databases	30
4	ST310	Machine Learning	100

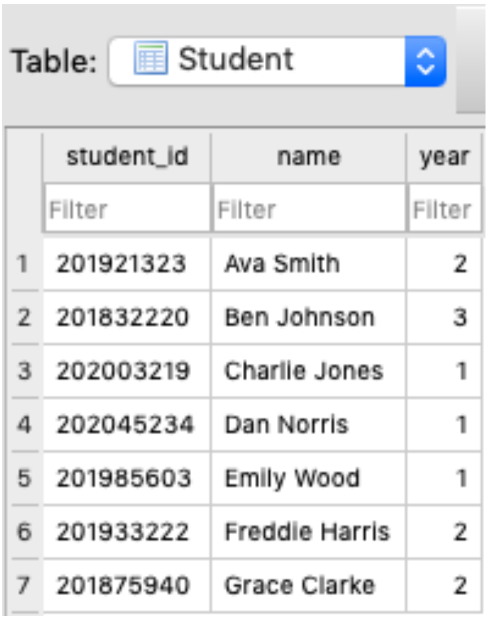


Table: Student

	student_id	name	year
1	201921323	Ava Smith	2
2	201832220	Ben Johnson	3
3	202003219	Charlie Jones	1
4	202045234	Dan Norris	1
5	201985603	Emily Wood	1
6	201933222	Freddie Harris	2
7	201875940	Grace Clarke	2

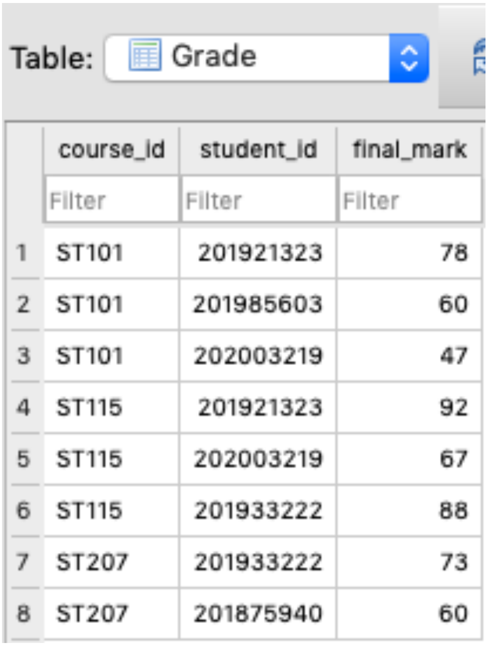
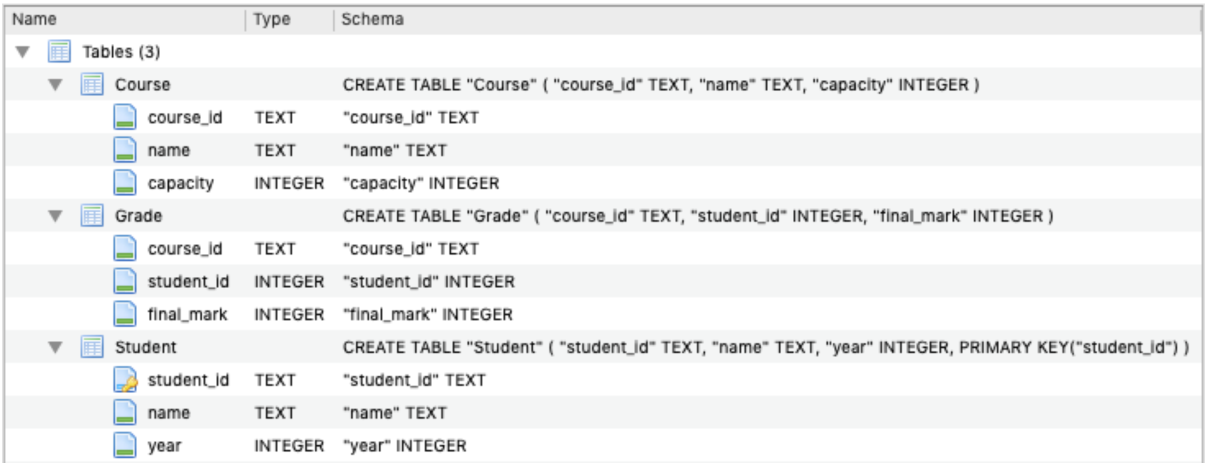


Table: Grade

	course_id	student_id	final_mark
1	ST101	201921323	78
2	ST101	201985603	60
3	ST101	202003219	47
4	ST115	201921323	92
5	ST115	202003219	67
6	ST115	201933222	88
7	ST207	201933222	73
8	ST207	201875940	60



Name	Type	Schema
Tables (3)		
Course		CREATE TABLE "Course" ("course_id" TEXT, "name" TEXT, "capacity" INTEGER)
course_id	TEXT	"course_id" TEXT
name	TEXT	"name" TEXT
capacity	INTEGER	"capacity" INTEGER
Grade		CREATE TABLE "Grade" ("course_id" TEXT, "student_id" INTEGER, "final_mark" INTEGER)
course_id	TEXT	"course_id" TEXT
student_id	INTEGER	"student_id" INTEGER
final_mark	INTEGER	"final_mark" INTEGER
Student		CREATE TABLE "Student" ("student_id" TEXT, "name" TEXT, "year" INTEGER, PRIMARY KEY("student_id"))
student_id	TEXT	"student_id" TEXT
name	TEXT	"name" TEXT
year	INTEGER	"year" INTEGER

2. Structured Query Language

Structured Query Language (SQL)

- A standardised domain-specific language used in programming and designed for managing data held in a RDBMS.
- Used to create, maintain (insert, update, delete), and retrieve (query) the relational database.

SQLite

- RDBMS contained in a C library.
- Lightweight, non-client-server database engine.

DB Browser for SQLite (DB4S)

- Tool to create, design, and edit database files that are compatible with SQLite.
- https://sqlitebrowser.org

Basic SQLite Syntax (Creating & Manipulating Databases)

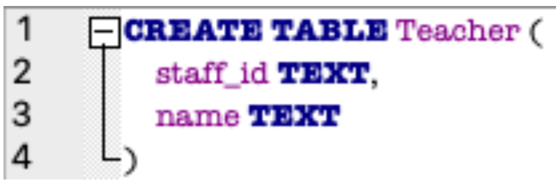
- **Add a Table**
  - CREATE TABLE *table_name* (

    attribute1 datatype,

    attribute2 datatype,

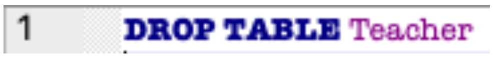
    …

)



```
CREATE TABLE Teacher (
    staff_id TEXT,
    name TEXT
)
```

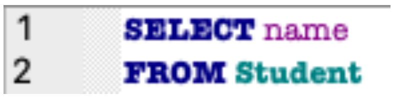
- **Delete a Table**
  - DROP TABLE *table_name*



```sql
DROP TABLE Teacher
```

- **Select Data**
  - SELECT *attribute1, attribute2, …*

    FROM *table_name*

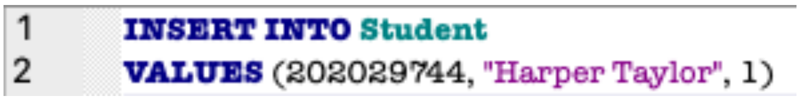


```sql
SELECT name
FROM Student
```

- **Insert Tuples/Rows**
  - INSERT INTO *table_name* (*attribute1, attribute2, …*)

    VALUES (*value1, value2, …*)
  - INSERT INTO *table_name*

    VALUES (*value1, value2, …*)

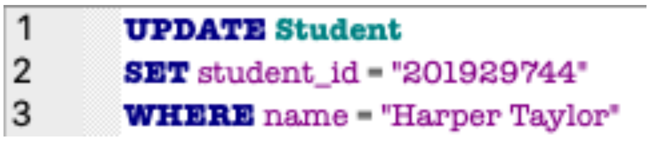


```sql
INSERT INTO Student
VALUES (202029744, "Harper Taylor", 1)
```

- **Update Tuples/Rows**
  - UPDATE *table_name*

    SET *attribute1* = *value1*, *attribute2* = *value2*, …

    WHERE *conditions*



```sql
UPDATE Student
SET student_id = "201929744"
WHERE name = "Harper Taylor"
```

- **Delete Tuples/Rows**
  - DELETE FROM *table_name*

    WHERE *conditions*

```sql
DELETE FROM Student
WHERE name = "Harper Taylor"
```

Basic SQLite Queries

- **Conditions**
  - SELECT *attribute1, attribute2, …*

    FROM *table_name*

    WHERE *conditions*

```sql
SELECT student_id
FROM Grade
WHERE course_id = 'ST101'
```

```sql
SELECT *
FROM Grade
WHERE course_id = 'ST101'
```

- **Several Tables**
  - SELECT *table.attribute1, table.attribute2, …*

    FROM *table1, table2, …*

    WHERE *condition1* AND/OR *condition2* …

```sql
SELECT Student.name
FROM Grade, Student
WHERE Grade.course_id = 'ST101' AND Student.student_id = Grade.student_id
```

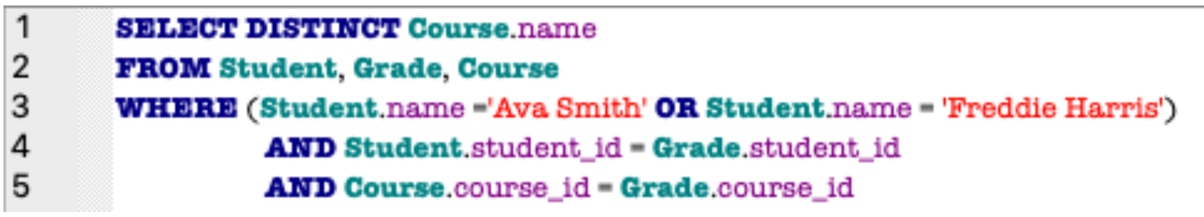
```sql
SELECT name
FROM Grade, Student
WHERE Grade.course_id = 'ST101' AND Student.student_id = Grade.student_id
```

- **Multiple Conditions**

```sql
SELECT Course.name
FROM Student, Grade, Course
WHERE (Student.name ='Ava Smith' OR Student.name = 'Freddie Harris')
            AND Student.student_id = Grade.student_id
            AND Course.course_id = Grade.course_id
```

- SELECT DISTINCT *attribute1, attribute2, …*

  FROM *table_name*



```
SELECT DISTINCT Course.name
FROM Student, Grade, Course
WHERE (Student.name ='Ava Smith' OR Student.name = 'Freddie Harris')
            AND Student.student_id = Grade.student_id
            AND Course.course_id = Grade.course_id
```

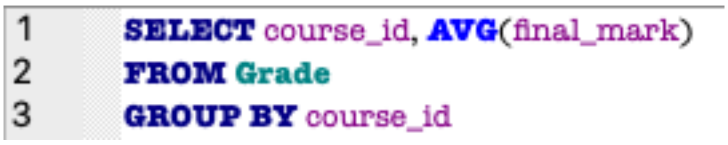
- **Aggregation**
  - SELECT *attributes, aggregation_functions* AS *column_name*

    FROM *table_name*

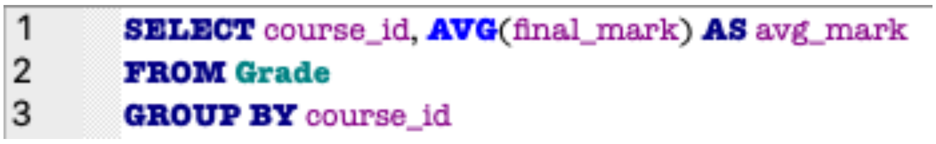
    WHERE *conditions*

    GROUP BY *attributes*

    ORDER BY *attributes*



```
SELECT course_id, AVG(final_mark)
FROM Grade
GROUP BY course_id
```



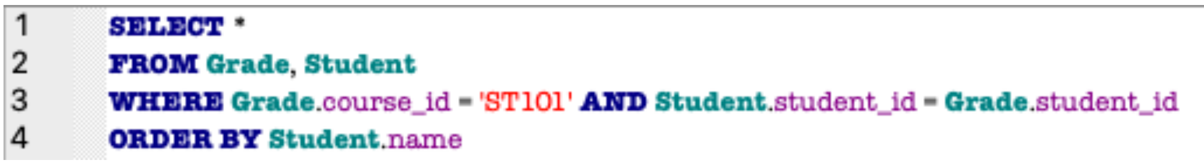
```
SELECT course_id, AVG(final_mark) AS avg_mark
FROM Grade
GROUP BY course_id
```

  - Aggregation functions
    - COUNT(), MAX(), MIN(), SUM(), AVG()

Basic SQLite Joins

- A ***JOIN*** clause is used to combine rows from two or more tables, based on a related column between them.

- **Inner Join**
  - Selects records that have matching values in both tables

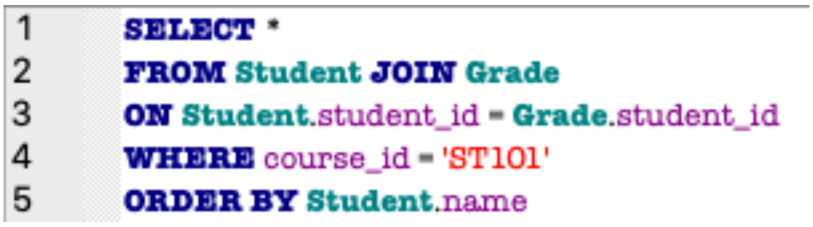


```
SELECT *
FROM Grade, Student
WHERE Grade.course_id = 'ST101' AND Student.student_id = Grade.student_id
ORDER BY Student.name
```

- SELECT *attributes*

  FROM *table1* JOIN *table2*

  ON *table1.attribute* = *table2.attribute*

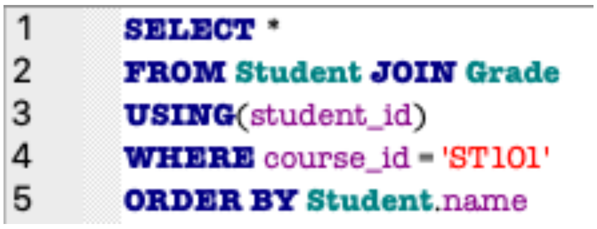


```
SELECT *
FROM Student JOIN Grade
ON Student.student_id = Grade.student_id
WHERE course_id = 'ST101'
ORDER BY Student.name
```

- SELECT *attributes*

  FROM *table1* JOIN *table2*

  USING(*attribute)*



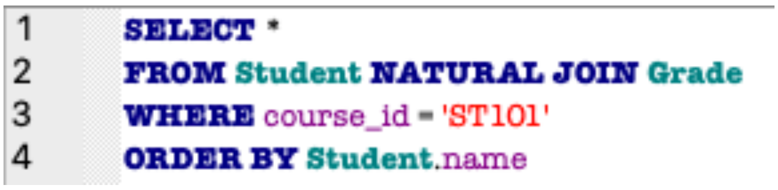
```
SELECT *
FROM Student JOIN Grade
USING(student_id)
WHERE course_id = 'ST101'
ORDER BY Student.name
```

- **Natural Join**
  - The join condition is automatically identified
  - SELECT *attributes*

    FROM *table1* NATURAL JOIN *table2*

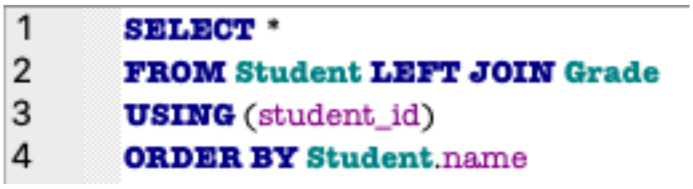
    WHERE *conditions*

    ORDER BY *attribute1*



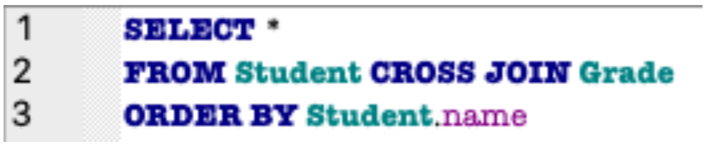
```
SELECT *
FROM Student NATURAL JOIN Grade
WHERE course_id = 'ST101'
ORDER BY Student.name
```

- **Left Join**
  - Returns all records from the left table (table1), and the matching records from the right table (table2)
  - *NULL* value for right table (table2) attributes with no corresponding record
  - SELECT *attributes*
    FROM *table1* LEFT JOIN *table2*
    USING (*attribute*)



```sql
SELECT *
FROM Student LEFT JOIN Grade
USING (student_id)
ORDER BY Student.name
```

- **Cross Join**
  - Returns all records when there is a match in left table (table1) or right table (table2) records
  - SELECT *attributes*
    FROM *table1* CROSS JOIN *table2*
    WHERE *conditions*
    ORDER BY *attribute1*



```sql
SELECT *
FROM Student CROSS JOIN Grade
ORDER BY Student.name
```

3. Using Databases with R

Connecting to Databases in R (DBI)

- Use "RSQLite" and "DBI" packages
  - install.packages("RSQLite")
  - library(DBI)

- Set working directory
  - setwd("~/ST2195/database/")

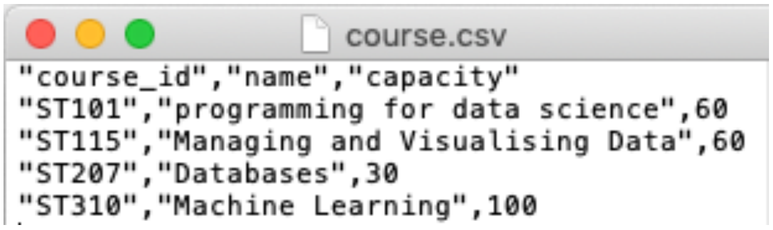
- Remove existing database
  - if (file.exists("University.db")

    file.remove("University.db")

- Create connection to database
  - conn <- dbConnect(RSQLite::SQLite(), "University.db")

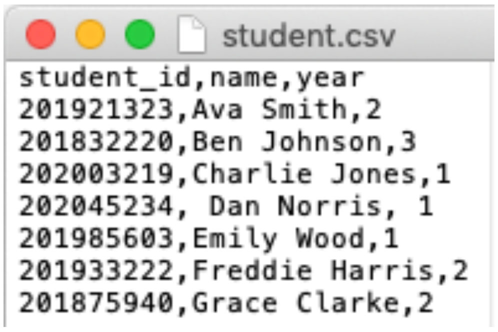
Creating Tables in R (DBI)

- Read CSV files as data frames
  - course <- read.csv("course.csv", header = TRUE)



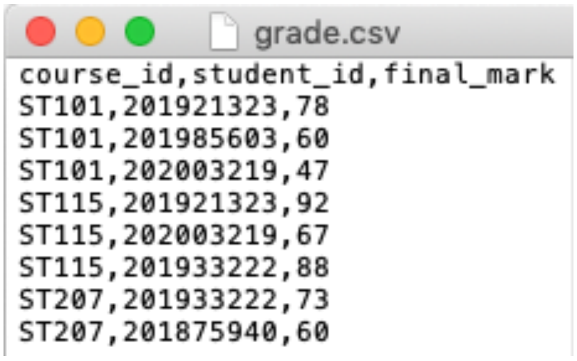
```
"course_id","name","capacity"
"ST101","programming for data science",60
"ST115","Managing and Visualising Data",60
"ST207","Databases",30
"ST310","Machine Learning",100
```

  - student <- read.csv("student.csv", header = TRUE)



```
student_id,name,year
201921323,Ava Smith,2
201832220,Ben Johnson,3
202003219,Charlie Jones,1
202045234, Dan Norris, 1
201985603,Emily Wood,1
201933222,Freddie Harris,2
201875940,Grace Clarke,2
```

- grade <- read.csv("grade.csv", header = TRUE)



```
course_id,student_id,final_mark
ST101,201921323,78
ST101,201985603,60
ST101,202003219,47
ST115,201921323,92
ST115,202003219,67
ST115,201933222,88
ST207,201933222,73
ST207,201875940,60
```

- Copy data frames to database tables
  - dbWriteTable(conn, "Course", course)
  - dbWriteTable(conn, "Student", student)
  - dbWriteTable(conn, "Grade", grade)

- Adding a new table
  - dbCreateTable(conn, "Teacher",
    c(staff_id = "TEXT", name = "TEXT"))

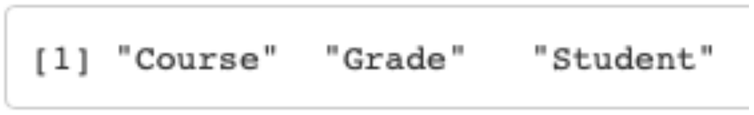
    or
  - dbExecute(conn, "CREATE TABLE Teacher (
    staff_id TEXT PRIMARY KEY,
    name TEXT)")

- Deleting a table
  - dbRemoveTable(conn, "Teacher")

or

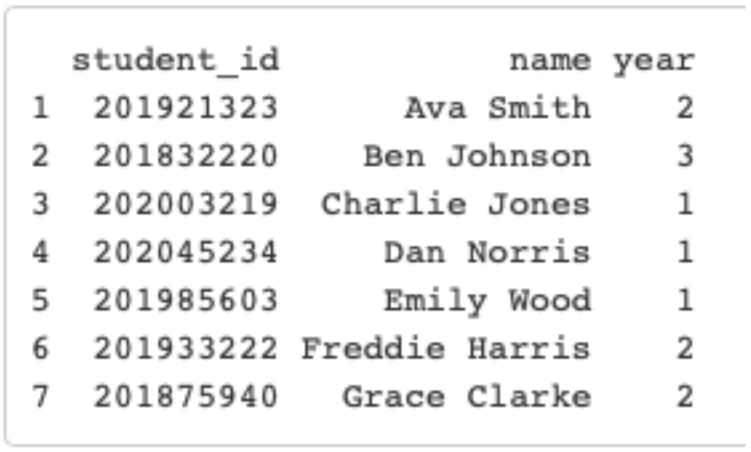
- dbExecute(conn, “DROP TABLE Teacher”)

- List database tables
  - dbListTables(conn)



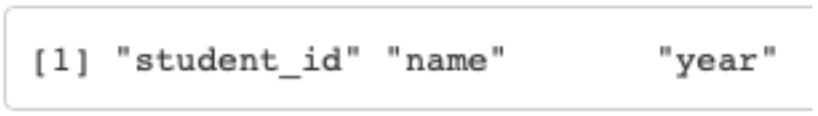
```
[1] "Course"  "Grade"   "Student"
```

- Browse database table
  - dbReadTable(conn, “Student”)



```
  student_id           name year
1  201921323      Ava Smith    2
2  201832220    Ben Johnson    3
3  202003219  Charlie Jones    1
4  202045234     Dan Norris    1
5  201985603     Emily Wood    1
6  201933222 Freddie Harris    2
7  201875940    Grace Clarke    2
```

- Show attributes
  - dbListFields(conn, “Student”)



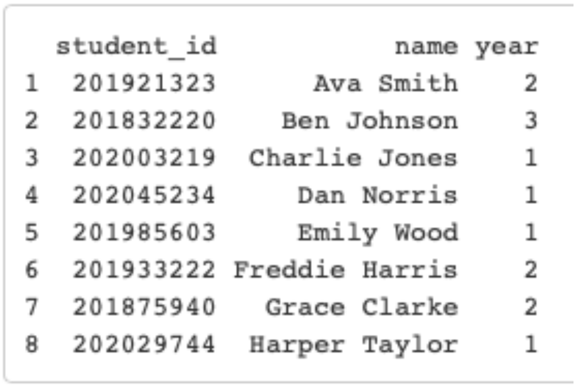
```
[1] "student_id" "name"       "year"
```

Manipulating Data in R (DBI)

- Inserting tuples/rows
  - dbAppendTable(conn, “Student”,
                              data.frame(student_id = “202029744”,
                              name = “Harper Taylor”, year = 1))

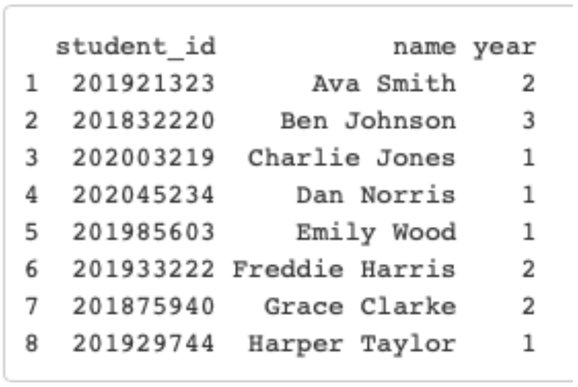
or

- dbExecute(conn, “INSERT INTO Student
  VALUES(202029744, ‘Harper Taylor’, 1)”)



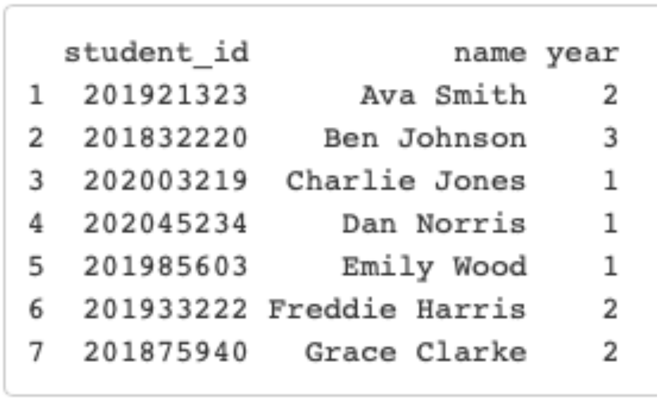
```
  student_id          name year
1  201921323     Ava Smith    2
2  201832220   Ben Johnson    3
3  202003219 Charlie Jones    1
4  202045234    Dan Norris    1
5  201985603    Emily Wood    1
6  201933222 Freddie Harris   2
7  201875940  Grace Clarke    2
8  202029744 Harper Taylor    1
```

- Updating tuples/rows
  - dbExecute(conn, “UPDATE Student
    SET student_id = ‘201929744’
    WHERE name = ‘Harper Taylor’”)



```
  student_id          name year
1  201921323     Ava Smith    2
2  201832220   Ben Johnson    3
3  202003219 Charlie Jones    1
4  202045234    Dan Norris    1
5  201985603    Emily Wood    1
6  201933222 Freddie Harris   2
7  201875940  Grace Clarke    2
8  201929744 Harper Taylor    1
```

- Deleting tuples/rows
  - dbExecute(conn, “DELETE FROM Student
    WHERE name = ‘Harper Taylor’”)

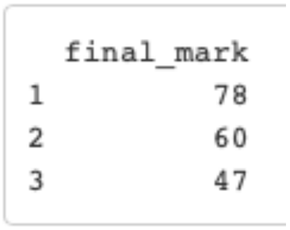


```
  student_id           name year
1  201921323      Ava Smith    2
2  201832220    Ben Johnson    3
3  202003219  Charlie Jones    1
4  202045234     Dan Norris    1
5  201985603     Emily Wood    1
6  201933222 Freddie Harris    2
7  201875940   Grace Clarke    2
```

- Disconnecting from database
  - dbDisconnect(conn)

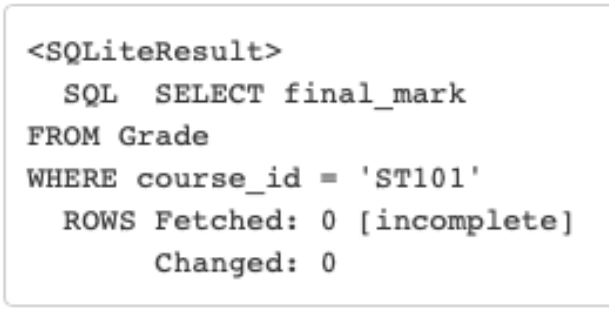
Querying Databases in R (DBI)

- Getting query result in data frame
  - q1 <- dbGetQuery(conn, "SELECT final_mark
    FROM Grade
    WHERE course_id = 'ST101'")

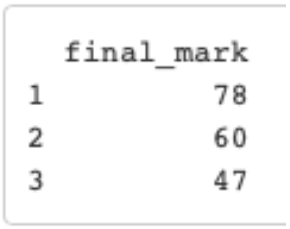


```
  final_mark
1         78
2         60
3         47
```

- Sending query to database engine
  - q1 <- dbSendQuery(conn, "SELECT final_mark
    FROM Grade
    WHERE course_id = 'ST101'")


```
<SQLiteResult>
  SQL  SELECT final_mark
FROM Grade
WHERE course_id = 'ST101'
  ROWS Fetched: 0 [incomplete]
       Changed: 0
```

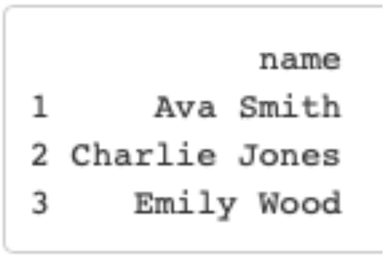
- Fetch query result from database engine
    - dbFetch(q1)


```
  final_mark
1         78
2         60
3         47
```

- Getting results in alphabetical order
    - dbGetQuery(conn, “SELECT Student.name
      FROM Grade, Student
      WHERE Grade.course_id = ‘ST101’ AND
      Student.student_id = Grade.student_id
      ORDER BY Student.name”)

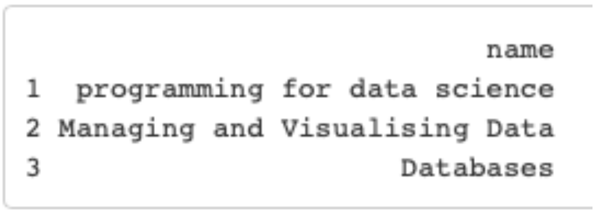
  or

    - dbGetQuery(conn, “SELECT Student.name
      FROM Student NATURAL JOIN Grade
      WHERE course_id = ‘ST101’
      ORDER BY Student.name”)


```
          name
1    Ava Smith
2 Charlie Jones
3   Emily Wood
```

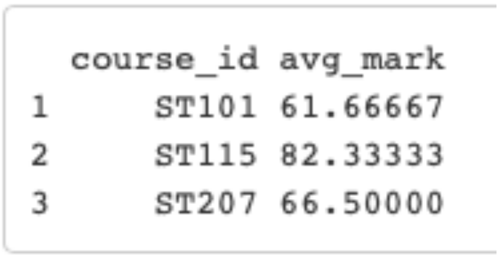
- Getting distinct results
    - dbGetQuery(conn, “SELECT DISTINCT Course.name
      FROM Student, Grade, Course
      WHERE (Student.name = ‘Ava Smith’ OR
      Student.name = ‘Freddie Harris’) AND
      Student.student_id = Grade.student_id
      AND Course.course_id = Grade.course_id”)

      or
    - dbGetQuery(conn, “SELECT DISTINCT Course.name
      FROM (Student NATURAL JOIN Grade)
      S JOIN Course
      ON Course.course_id = S.course_id
      WHERE S.name = ‘Ava Smith’ OR
      S.name = ‘Freddie Harris’”)



```
                        name
1  programming for data science
2 Managing and Visualising Data
3                     Databases
```

- Getting calculated results
    - dbGetQuery(conn, “SELECT course_id, AVG(final_mark)
      AS avg_mark
      FROM Grade
      GROUP BY course_id”)

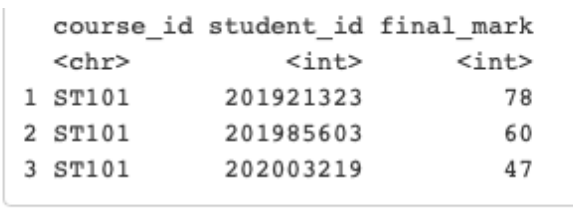


```
  course_id avg_mark
1     ST101 61.66667
2     ST115 82.33333
3     ST207 66.50000
```

Querying Databases in R (dplyr)

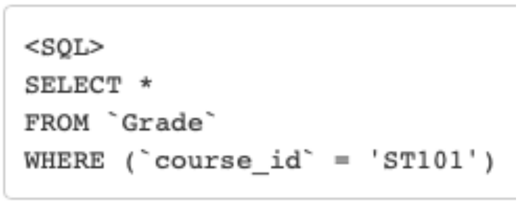
- Creating a reference to tables
  - library(dplyr)
  - student_db <- tbl(conn, “Student”)
  - grade_db <- tbl(conn, “Grade”)
  - course_db <- tbl(conn, “Course”)

- Getting rows with conditions
  - q1 <- grade_db %>% filter(course_id == “ST101”)



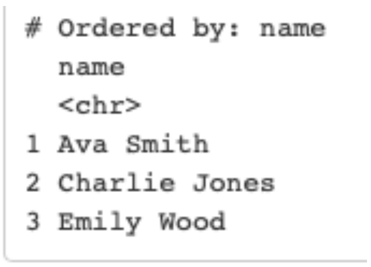
```
  course_id student_id final_mark
  <chr>          <int>      <int>
1 ST101      201921323         78
2 ST101      201985603         60
3 ST101      202003219         47
```

  - show_query(q1)



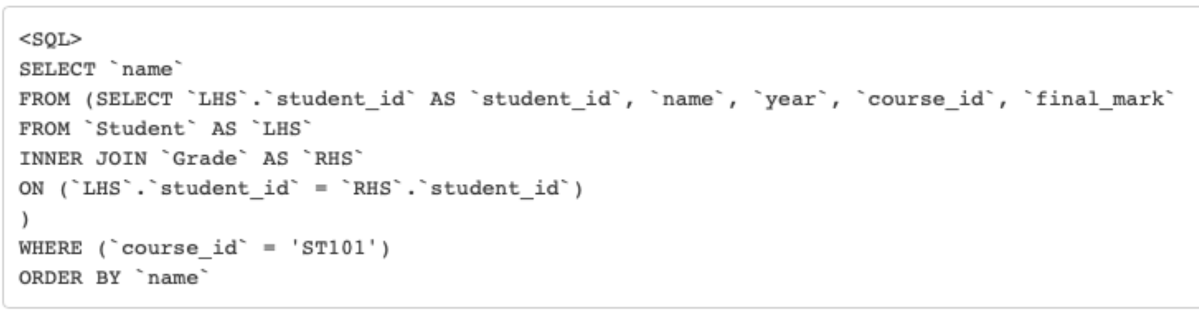
```
<SQL>
SELECT *
FROM `Grade`
WHERE (`course_id` = 'ST101')
```

- Getting rows in alphabetical order
  - q2 <- inner_join(student_db, grade_db) %>%
    filter(course_id == “ST101”) %>%
    select(name) %>%
    arrange(name)



```
# Ordered by: name
  name
  <chr>
1 Ava Smith
2 Charlie Jones
3 Emily Wood
```

- show_query(q2)



```
<SQL>
SELECT `name`
FROM (SELECT `LHS`.`student_id` AS `student_id`, `name`, `year`, `course_id`, `final_mark`
FROM `Student` AS `LHS`
INNER JOIN `Grade` AS `RHS`
ON (`LHS`.`student_id` = `RHS`.`student_id`)
)
WHERE (`course_id` = 'ST101')
ORDER BY `name`
```

- Getting distinct rows
  - q3 <- inner_join(student_db, grade_db, by = “student_id”) %>%

    inner_join(course_db, by = “course_id”,

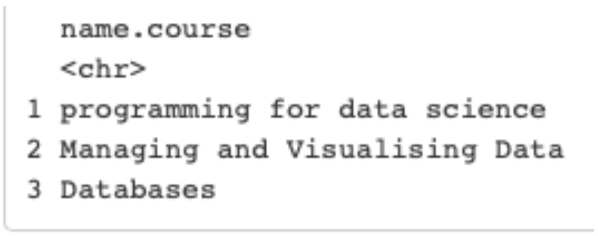
    suffix = c(“.student”, “.course”) %>%

    filter(name.student == ‘Ava Smith’ |

    name.student == ‘Freddie Harris’) %>%

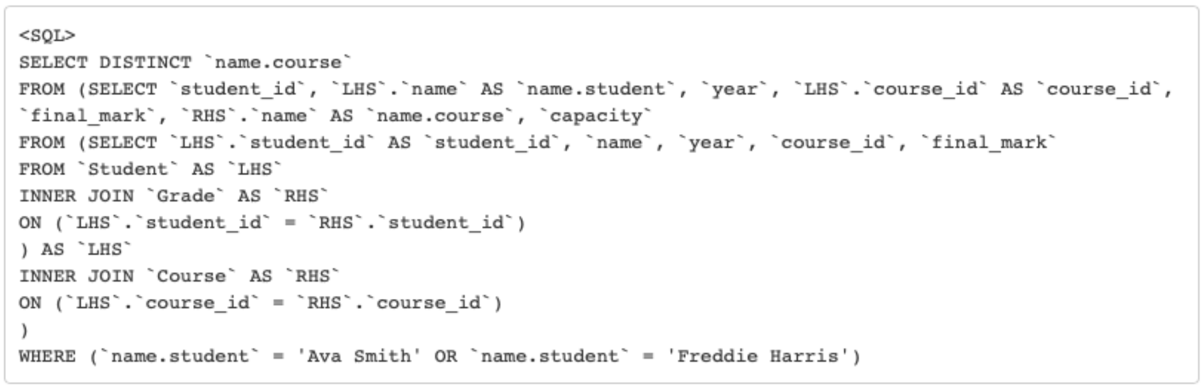
    select(name.course) %>%

    distinct()



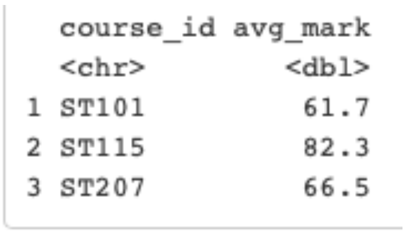
```
  name.course
  <chr>
1 programming for data science
2 Managing and Visualising Data
3 Databases
```

  - show_query(q3)



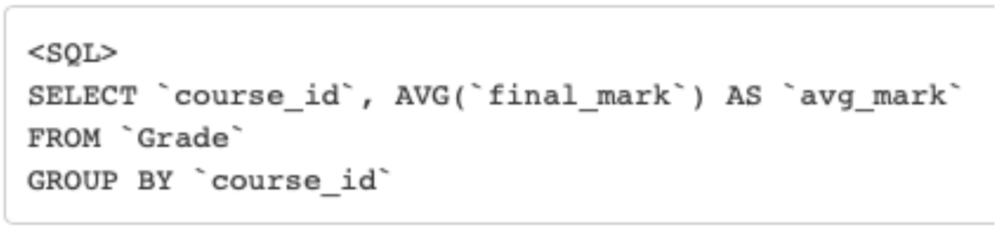
```
<SQL>
SELECT DISTINCT `name.course`
FROM (SELECT `student_id`, `LHS`.`name` AS `name.student`, `year`, `LHS`.`course_id` AS `course_id`,
`final_mark`, `RHS`.`name` AS `name.course`, `capacity`
FROM (SELECT `LHS`.`student_id` AS `student_id`, `name`, `year`, `course_id`, `final_mark`
FROM `Student` AS `LHS`
INNER JOIN `Grade` AS `RHS`
ON (`LHS`.`student_id` = `RHS`.`student_id`)
) AS `LHS`
INNER JOIN `Course` AS `RHS`
ON (`LHS`.`course_id` = `RHS`.`course_id`)
)
WHERE (`name.student` = 'Ava Smith' OR `name.student` = 'Freddie Harris')
```

- Getting calculated rows
    - q4 <- grade_db %>% group_by(course_id) %>%
      summarize(avg_mark = mean(final_mark, na.rm = TRUE))



```
  course_id avg_mark
  <chr>        <dbl>
1 ST101         61.7
2 ST115         82.3
3 ST207         66.5
```

    - show_query(q4)



```
<SQL>
SELECT `course_id`, AVG(`final_mark`) AS `avg_mark`
FROM `Grade`
GROUP BY `course_id`
```

4. Using Databases with Python

Connecting to Databases in Python

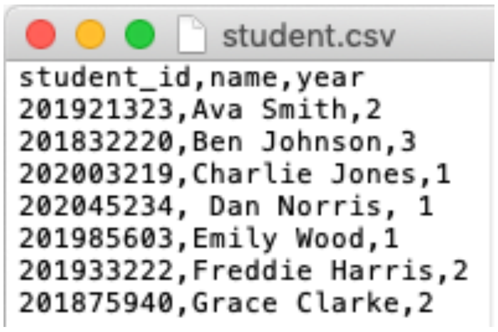
- Remove existing database
  - import os
    ```
    try:
        os.remove('University.db')
    except OSError:
        pass
    ```

- Use "sqlite3" packages
  - import sqlite3

- Create connection to database
  - conn = sqlite3.connect('University.db')

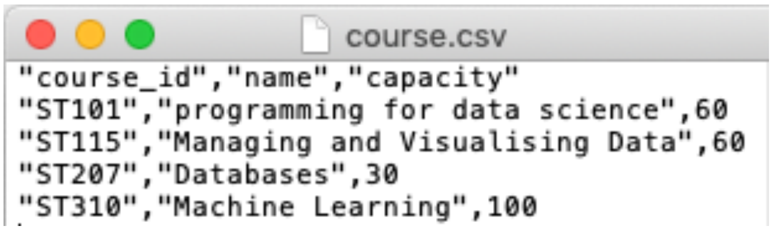
Creating Tables in Python

- Read CSV files as data frames
  - import pandas as pd
  - student = pd.read_csv("student.csv")



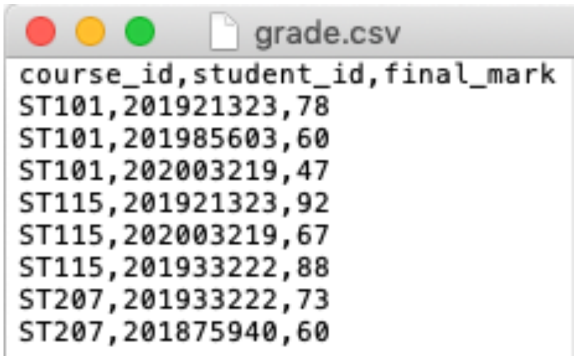
```
student_id,name,year
201921323,Ava Smith,2
201832220,Ben Johnson,3
202003219,Charlie Jones,1
202045234, Dan Norris, 1
201985603,Emily Wood,1
201933222,Freddie Harris,2
201875940,Grace Clarke,2
```

  - course = pd.read_csv("course.csv")



```
"course_id","name","capacity"
"ST101","programming for data science",60
"ST115","Managing and Visualising Data",60
"ST207","Databases",30
"ST310","Machine Learning",100
```

- grade = pd.read_csv("grade.csv")



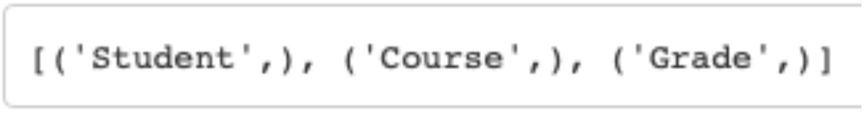
```
course_id,student_id,final_mark
ST101,201921323,78
ST101,201985603,60
ST101,202003219,47
ST115,201921323,92
ST115,202003219,67
ST115,201933222,88
ST207,201933222,73
ST207,201875940,60
```

- Copy data frames to database tables
  - student.to_sql('Student', con = conn, index = False)
  - course.to_sql('Course', con = conn, index = False)
  - grade.to_sql('Grade', con = conn, index = False)

Manipulating Data in Python

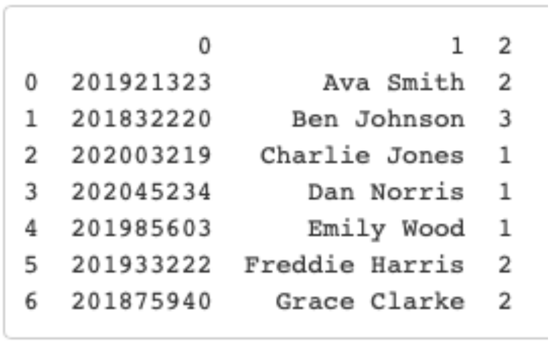
- Create cursor object
  - c = conn.cursor()

- Execute SQL commands to get all tables
  - c.execute(''' SELECT name
                FROM sqlite_master
                WHERE type='table'
                ''')
- Fetch all tables
  - c.fetchall()



```
[('Student',), ('Course',), ('Grade',)]
```

- Browse database table
  - q = c.execute(“SELECT * FROM Student”).fetchall()
  - pd.DataFrame(q)

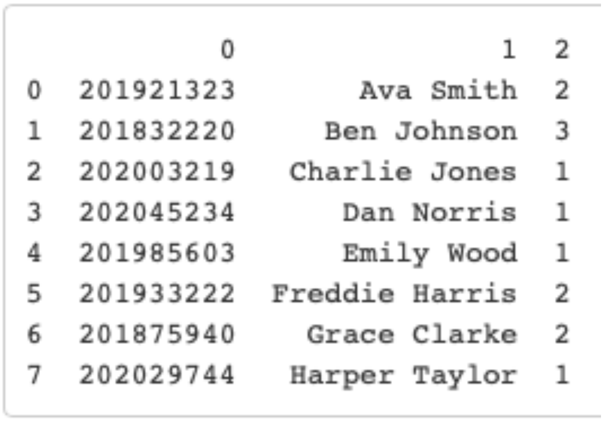


```
           0               1  2
0  201921323       Ava Smith  2
1  201832220     Ben Johnson  3
2  202003219   Charlie Jones  1
3  202045234       Dan Norris  1
4  201985603       Emily Wood  1
5  201933222  Freddie Harris  2
6  201875940    Grace Clarke  2
```

- Add a new table
  - c.execute(‘‘‘ CREATE TABLE Teacher
    (staff_id TEXT PRIMARY KEY, name TEXT)
    ’’’)
  - conn.commit()

- Delete a table
  - c.execute(“DROP TABLE Teacher”)
  - conn.commit()

- Insert tuples/rows
  - c.execute(‘‘‘ INSERT INTO Student
    VALUES(202029744, ‘Harper Taylor’, 1)
    ’’’)
  - conn.commit()



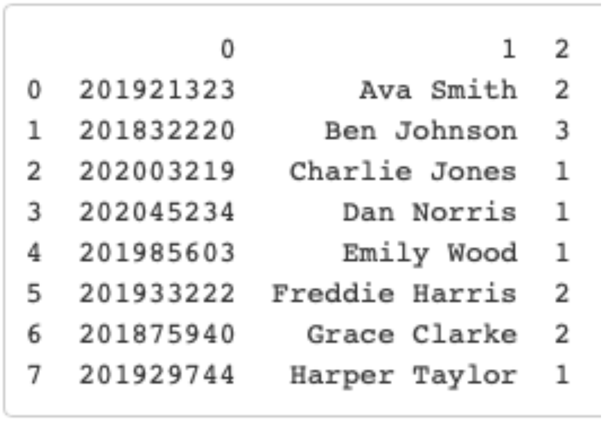
```
           0             1  2
0  201921323     Ava Smith  2
1  201832220   Ben Johnson  3
2  202003219  Charlie Jones  1
3  202045234    Dan Norris  1
4  201985603    Emily Wood  1
5  201933222 Freddie Harris  2
6  201875940  Grace Clarke  2
7  202029744  Harper Taylor  1
```

- Update tuples/rows
  - c.execute(‘‘‘ UPDATE Student

    SET student_id = “201929744”

    WHERE name = “Harper Taylor”

    ’’’)
  - conn.commit()

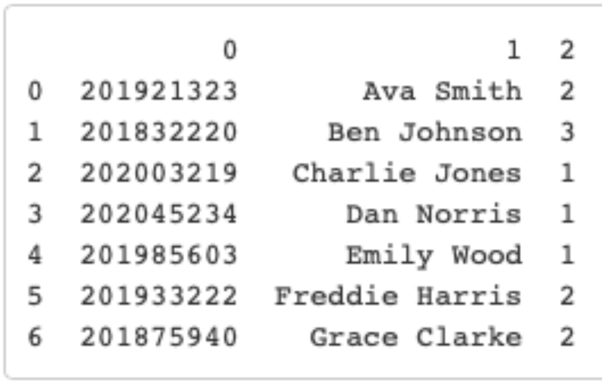


```
           0             1  2
0  201921323     Ava Smith  2
1  201832220   Ben Johnson  3
2  202003219  Charlie Jones  1
3  202045234    Dan Norris  1
4  201985603    Emily Wood  1
5  201933222 Freddie Harris  2
6  201875940  Grace Clarke  2
7  201929744  Harper Taylor  1
```

- Delete tuples/rows
  - c.execute(‘‘‘ DELETE FROM Student

    WHERE name = “Harper Taylor”

    ’’’)
  - conn.commit()

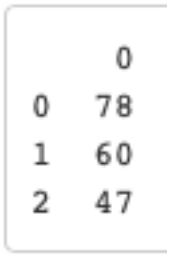


```
           0             1  2
0  201921323     Ava Smith  2
1  201832220   Ben Johnson  3
2  202003219  Charlie Jones 1
3  202045234     Dan Norris 1
4  201985603    Emily Wood  1
5  201933222 Freddie Harris 2
6  201875940   Grace Clarke 2
```

- Disconnecting from database
  - conn.close()

Querying Databases in Python

- Get query results in data frame
  - q1 = c.execute(''' SELECT final_mark
    FROM Grade
    WHERE course_id = 'ST101'
    ''').fetchall()
  - pd.DataFrame(q1)

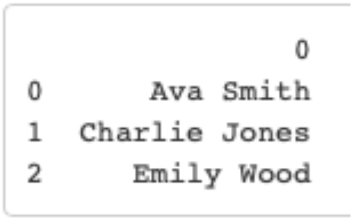


```
    0
0  78
1  60
2  47
```

- Get results in alphabetical order
  - q2 = c.execute(''' SELECT Student.name
    FROM Grade, Student
    WHERE Grade.course_id='ST101' AND
    Student.student_id=Grade.student.id

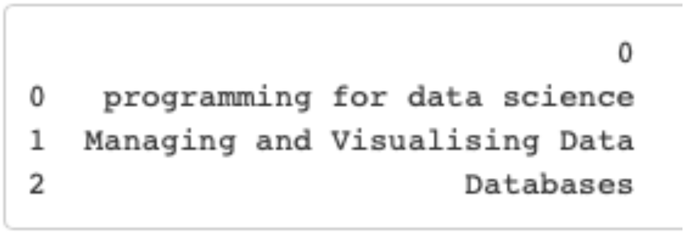
ORDER BY Student.name
’’’).fetchall()

- pd.DataFrame(q2)



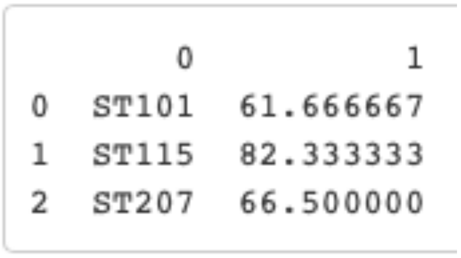
```
             0
0      Ava Smith
1  Charlie Jones
2     Emily Wood
```

- Get distinct results
  - q3 = c.execute(‘‘‘ SELECT DISTINCT Course.name
    FROM Student, Grade, Course
    WHERE (Student.name = ‘Ava Smith’ OR
    Student.name = ‘Freddie Harris’) AND
    Student.student_id = Grade.student_id
    AND Course.course_id = Grade.course_id
    ’’’).fetchall()
  - pd.DataFrame(q3)



```
                             0
0   programming for data science
1  Managing and Visualising Data
2                      Databases
```

- Get calculated results
  - q4 = c.execute(‘‘‘ SELECT course_id, AVG(final_mark)
    AS avg_mark
    FROM Grade
    GROUP BY course_id
    ’’’).fetchall()
  - pd.DataFrame(q4)



```
       0          1
0  ST101  61.666667
1  ST115  82.333333
2  ST207  66.500000
```

Useful Resources

-
    - http://