HW4A Python

May 15, 2025

W4111_2025_002_1: Introduction to Databases:Homework 4A

1 Overview

1.1 Scope

The material in scope for this homework is: - The content of lectures: - All material from lectures 1 to lecture 10. - This includes any material in the slides, even if not explicitly presented in lecture. - Any information provided or discussed in lectures, even if not in slides. - The slides associated with the recommended textbook for - All material from the textbook slides that were in scope for HW 3A. - Chapter 4. - Chapter 5: Slides 1-4, slide 5.13, 5.18 - 5.27, 5.31 to the end. - Chapter 6. - Chapter 7: Slides 7.1 - 7.41, 7.89 to the end. - Chapter 12. - Chapter 13. - Chapter 14: 14.1 - 14.45, 14.51 - 14.66. - Chapter 15: 15.1 - 15.42, 15.44 - 15.47, 15.51 - 15.58.

[ ]:

1.2 Submission Instructions

Note to DFF: Create necessary links.

- Due date: 2025-April-19, 11:59 PM EDT on GradeScope.
- You submit on GradeScope. We will create a GradeScope submission for the homework.
- Your submission is a PDF of this notebook. You must tag the submission with locations in the PDF for each question. You must solve problems you experience producing a PDF including images. Please do not wait until the last minute.

There is a post/mega-thread on Ed Discussions that we will use to resolve questions and issues with respect to homework 4A.

1.3 Brevity

Brevity

Students sometimes just write a lot of words hoping to get something right. We will deduct points if your answer is too long.

2 Initialization

2.1 Python Environment

```
[ ]:
```

```
[16]: import copy
```

```
[17]: import json
```

```
[18]: import pandas
```

```
[95]: # You should have installed the packages for previous homework assignments
      #
      import pymysql
      import sqlalchemy
      from sqlalchemy import create_engine
```

```
[96]: import numpy
```

```
[97]: # You have installed and configured ipython-sql for previous assignments.
      # https://pypi.org/project/ipython-sql/
      #
      engine = create_engine(db_url)
      print(engine)

      %sql SHOW TABLES
```

```
Engine(mysql+pymysql://root:***@localhost/classicmodels)
 * mysql+pymysql://root:***@localhost/classicmodels
   mysql+pymysql://root:***@localhost?local_infile=1
8 rows affected.
```

```
[97]: [('customers',),
       ('employees',),
       ('offices',),
       ('orderdetails',),
       ('orders',),
       ('payments',),
       ('productlines',),
       ('products',)]
```

```
[98]: # This is a hack to fix a version problem/incompatibility  with some of the␣
       ↪packages and magics.
      #
      %config SqlMagic.style = '_DEPRECATED_DEFAULT'
```

[99]:

```
# Make sure that you set these values to the correct values for your␣
 ↪installation and
# configuration of MySQL
#
db_user = "root"
db_password = "rootpass"
```

[100]:

```
# Create the URL for connecting to the database.
# Do not worry about the local_infile=1, I did that for wizard reasons that you␣
 ↪should not have to use.
#
db_url = f"mysql+pymysql://{db_user}:{db_password}@localhost/classicmodels"
```

[130]:

```
print(db_url)
```

```
mysql+pymysql://root:rootpass@localhost/classicmodels
```

[131]:

```
# Initialize ipython-sql
#
%sql $db_url
```

[137]:

```
# Setup SQL Magic for Jupyter
%load_ext sql
%sql mysql+pymysql://root:rootpass@localhost/classicmodels
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

[138]:

```
  %reload_ext sql
```

[139]:

```
# Your answer will be different based on the databases that you have created on␣
 ↪your local MySQL instance.
#
%sql show tables from classicmodels
```

```
 * mysql+pymysql://root:***@localhost/classicmodels
   mysql+pymysql://root:***@localhost?local_infile=1
8 rows affected.
```

[139]:

```
[('customers',),
 ('employees',),
 ('offices',),
 ('orderdetails',),
 ('orders',),
 ('payments',),
 ('productlines',),
 ('products',)]
```

```
[140]: from sqlalchemy import create_engine  #  not from .future

       engine = create_engine(db_url)
       df = pandas.read_sql("SELECT * FROM customers", con=engine)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[140], line 4
      1 from sqlalchemy import create_engine  #  not from .future
      3 engine = create_engine(db_url)
----> 4 df = pandas.read_sql("SELECT * FROM customers", con=engine)

File ~/anaconda3/lib/python3.11/site-packages/pandas/io/sql.py:590, in␣
 ↪read_sql(sql, con, index_col, coerce_float, params, parse_dates, columns,␣
 ↪chunksize)
    581     return pandas_sql.read_table(
    582         sql,
    583         index_col=index_col,
   (…)
    587         chunksize=chunksize,
    588     )
    589 else:
--> 590     return pandas_sql.read_query(
    591         sql,
    592         index_col=index_col,
    593         params=params,
    594         coerce_float=coerce_float,
    595         parse_dates=parse_dates,
    596         chunksize=chunksize,
    597     )

File ~/anaconda3/lib/python3.11/site-packages/pandas/io/sql.py:1560, in␣
 ↪SQLDatabase.read_query(self, sql, index_col, coerce_float, parse_dates,␣
 ↪params, chunksize, dtype)
   1512 """
   1513 Read SQL query into a DataFrame.
   1514
   (…)
   1556
   1557 """
   1558 args = _convert_params(sql, params)
-> 1560 result = self.execute(*args)
   1561 columns = result.keys()
   1563 if chunksize is not None:

File ~/anaconda3/lib/python3.11/site-packages/pandas/io/sql.py:1405, in␣
 ↪SQLDatabase.execute(self, *args, **kwargs)
   1403 def execute(self, *args, **kwargs):
```

```
   1404     """Simple passthrough to SQLAlchemy connectable"""
-> 1405     return self.connectable.execution_options().execute(*args, **kwargs

AttributeError: 'OptionEngine' object has no attribute 'execute'
```

[154]:

```python
from sqlalchemy import create_engine, text

engine = create_engine(db_url)

with engine.connect() as connection:
    query = text("SELECT * FROM employees")
    result = connection.execute(query)
    df = pandas.DataFrame(result.fetchall(), columns=result.keys())

df
```

[154]:

```
    employeeNumber   lastName firstName extension  \
0             1002     Murphy     Diane     x5800
1             1056  Patterson      Mary     x4611
2             1076   Firrelli      Jeff     x9273
3             1088  Patterson   William     x4871
4             1102     Bondur    Gerard     x5408
5             1143        Bow   Anthony     x5428
6             1165   Jennings    Leslie     x3291
7             1166   Thompson    Leslie     x4065
8             1188   Firrelli     Julie     x2173
9             1216  Patterson     Steve     x4334
10            1286      Tseng  Foon Yue     x2248
11            1323     Vanauf    George     x4102
12            1337     Bondur      Loui     x6493
13            1370  Hernandez    Gerard     x2028
14            1401   Castillo    Pamela     x2759
15            1501       Bott     Larry     x2311
16            1504      Jones     Barry      x102
17            1611     Fixter      Andy      x101
18            1612      Marsh     Peter      x102
19            1619       King       Tom      x103
20            1621      Nishi      Mami      x101
21            1625       Kato   Yoshimi      x102
22            1702     Gerard    Martin     x2312

                              email officeCode  reportsTo  \
0      dmurphy@classicmodelcars.com          1        NaN
1     mpatterso@classicmodelcars.com          1     1002.0
2     jfirrelli@classicmodelcars.com          1     1002.0
3   wpatterson@classicmodelcars.com          6     1056.0
4      gbondur@classicmodelcars.com          4     1056.0
```

```
5         abow@classicmodelcars.com          1     1056.0
6    ljennings@classicmodelcars.com          1     1143.0
7    lthompson@classicmodelcars.com          1     1143.0
8    jfirrelli@classicmodelcars.com          2     1143.0
9   spatterson@classicmodelcars.com          2     1143.0
10      ftseng@classicmodelcars.com          3     1143.0
11     gvanauf@classicmodelcars.com          3     1143.0
12     lbondur@classicmodelcars.com          4     1102.0
13   ghernande@classicmodelcars.com          4     1102.0
14   pcastillo@classicmodelcars.com          4     1102.0
15       lbott@classicmodelcars.com          7     1102.0
16      bjones@classicmodelcars.com          7     1102.0
17     afixter@classicmodelcars.com          6     1088.0
18      pmarsh@classicmodelcars.com          6     1088.0
19       tking@classicmodelcars.com          6     1088.0
20      mnishi@classicmodelcars.com          5     1056.0
21       ykato@classicmodelcars.com          5     1621.0
22     mgerard@classicmodelcars.com          4     1102.0

                jobTitle
0              President
1               VP Sales
2           VP Marketing
3   Sales Manager (APAC)
4    Sale Manager (EMEA)
5     Sales Manager (NA)
6              Sales Rep
7              Sales Rep
8              Sales Rep
9              Sales Rep
10             Sales Rep
11             Sales Rep
12             Sales Rep
13             Sales Rep
14             Sales Rep
15             Sales Rep
16             Sales Rep
17             Sales Rep
18             Sales Rep
19             Sales Rep
20             Sales Rep
21             Sales Rep
22             Sales Rep
```

3 Written Questions

3.1 ER Modeling

Question

The following diagram uses the visual notation associated with the recommended textbook.

ER Modelling

Both *customer* and *shipper* have partial participation in the relationship *is_shipper.* In the relationship *is_supplier*, *customer* has total participation and *supplier* has partial participation.

Convert the diagram to an equivalent Crow's Foot logical ER diagram. Please your diagram below.

ER Modelling

The above is a logical diagram because it explicitly states the data type as well as the constraints.

Answer

ER Modelling

3.2 Relational Algebra

Question

This question uses the UIBK - R, S, T Dataset from the RelaX Calculator. It depicts a semi-join

R S

Semi-Join

In the answer section, write an equivalent relational algebra that uses only the operators , , and . Put a screen shot of your execution in RelaX in the answer cell.

Answer

R.a, R.b, R.c (R (R.b = S.b) S)

Replace the image.

Relational Algebra Answer

3.3 Triggers and Functions

Question

List 3 differences between triggers and functions.

List two differences between functions and procedures.

Answer

“A trigger is a statement that the system executes automatically as a side effect of a modification to the database.” (§ 5.3, p. 203)

Triggers vs. Functions “While both triggers and functions share a common syntax, they differ significantly in purpose, invocation, and control.” 1. Invocation: A trigger is automatically executed by the system in response to INSERT, UPDATE, or DELETE events on a table or view, but a function is explicitly called by user or from within a query or other code. 2. Purpose: Trigger is typically used for side effects like enforcing business logic, auditing, or validating data changes. Function is used to return a computed value (e.g., scalar or table-valued result). 3. Return Value: A trigger does not return a value to the caller. A function must return a value, either scalar or a table (depending on function type).

Functions vs. Procedures 1. Return requirement: A function must return a single value or result set (table). A procedure may return zero or many values via OUT or INOUT parameters, but not required to return anything. 2. Usability in queries: A function can be used inside SELECT statements and expressions (if deterministic and “side-effect-free”); however, procedure cannot be used inside SELECT; must be called via CALL statement.

3.4 Security Concepts

Question

Briefly explain the concepts of: 1. Digital identity 2. Authentication 3. Authorization 4. Roles 5. Privilege

Answer

1. A digital identity refers to the unique representation of a user or entity within a database system. It is often based on a username (e.g., dff9) or user account, and serves as the basis for applying access control, authentication, and auditing.
2. Authentication is the process of verifying the digital identity of a user or system. It ensures that the person or application accessing the database is indeed who they claim to be.
3. Authorization is the process of determining what an authenticated user is allowed to do. It also covers access to database objects (e.g., tables, views) and operations (e.g., SELECT, INSERT, UPDATE).
4. A role is a named group of privileges that can be assigned to one or more users. Roles provide a scalable way to manage permissions, especially in systems with many users.

5. A privilege is a specific permission to perform a particular action on a database object. Examples include permission to: • SELECT from a table, • UPDATE a column, • EXECUTE a stored procedure.

EXTRA: - Example of granting specific privileges to a user on a given table: ‘GRANT SELECT, INSERT ON Grades TO dff9;’ - Assigns pre-defined role ‘GRANT instructor_role TO dff9;’ - Replace with REVOKE to do the opposite.

Example code %%sql – Create a role for instructors CREATE ROLE instructor_role;

– Grant relevant permissions to the role GRANT SELECT, UPDATE ON Grades TO instructor_role;

– Assign the role to a user GRANT instructor_role TO dff9;

3.5 Recursion

Question

Despite massively freaking out Professor Ferguson, recursion in SQL queries provides a very valuable capability. What is that capability and provide a description of a query using Classic Models that would use the capability.

Answer

We might use recursive querying in the case that we want to try and compute the transitive closure of a relation. “The transitive closure of a relation describes all possible paths (or reachability) in a graph from one node to others by following edges.” Here, nodes become entities and paths/edges become relations. The textbook gives the example of pre-requisite courses for a particular class, but we want to see all the pre-requisite for courses, even those pre-requisites of its pre-requisites, in order to see all the classes that would need to be taken in order to enroll in a particular class. This allows us to iteratively move towards the first required course without a pre-requisite. We should still be careful to avoid the possibility for writing non-terminating recursive code. In “ClassicModels,” we could thinking about a hierarchy of employees, where want to see the full chain of how reports to how in a specific hierarchy.

3.6 Normalization

Question

Briefly explain: 1. Two evils/downsides of data redundancy. 2. Decomposition, Lossy and Lossless. 3. Functional dependencies. 4. The concept of the *closure* of functional dependency, denoted F+. 5. What capability/result is achievable with 3NF but not BCNF?

Answer

1. Two downsides of data redundancy

a. Possibility for inconsistency between files stored in different locations
b. “Wasted” space, insofar as multiple hard drives have to contain the same data, which

2. On Decomposition, Lossy, Lossless

a. Decomposition - splitting a table in two or smaller tables in order to improve design/functionality/eliminate redundancy.

b. Lossy - Data is lost during the split, meaning the original table cannot be reconstructed using join operations.
c. Lossless - Means no data is lost during the split and the original data can be formed out joins of the parts.

3. Functional Dependencies

a. a term for describing/expressing a relation, where one set of attributes determines another (§7.2, p. 309)

4. Closure of FD (i.e., F+)

a. F+ is the set that contains all "functional dependencies that can be inferred from the given set F." (pg. 312)

3.7 Disks and Storage

Question

Hard disk drives typically have many cylinders. Some database systems in some scenarios only use a subset of the cylinders and not others. Why?

Would the database prefer outer cyclinders or inner cylinder? Would the database prefer contiguous cylinders or would it have empty cyclinders in between ones that it would use?

Enter your answer below. Include a brief explanation of your answer.

Answer

A database system would prefer to use the outer cylinders of a hard disk and would favor contiguous cylinder allocation over skipping cylinders, for performance reasons. Because not all cylinders offer the same performance, and database systems often optimize for speed, they may restrict data placement to only the outermost cylinders, which provide higher data transfer rates due to physical disk geometry. Outer cylinders have greater linear velocity and therefore store more data per track than inner cylinders. When the disk spins at a constant angular velocity (as is the case for HDDs), the read/write head covers more physical space per second on the outer edge. As a result, data transfer rates are higher on outer cylinders than on inner ones. Therefore, database systems prefer outer cylinders for storing frequently accessed or sequential data to maximize throughput. In relation to the preference for contiguity, accessing contiguous cylinders minimizes seek time and rotational latency, especially for large sequential reads/writes, and if the database skips cylinders, whether due to fragmentation or poor allocation, the disk head must perform more random seeks, which degrades performance.

3.8 Database File Organization

Question

What are 5 approaches/designs for organizing records in a file?

Consider the following assumptions for a scenario: 1. The original logical model had a single table $Orders(orderId, customerId, orderDate, productCode, quantityOrdered)$. There would be one row/record for each product in an order. 2. For design reasons the designer split the table into two tables: 1. $Orders(orderId, customerId, orderDate)$. 2.

$OrdersItem(orderId, productCode, quantityOrdered)$. 3. Defined a view that recreated the original table definition. 4. The most common access pattern was to read the data through the view.

What record organization approach would you use and why?

Answer 1. Heap (Unordered) - Records are stored in no particular order. New records are added to the end of the file or in available free space. Although this might work for smaller DBs, there is a major drawback of needing a full scan for queries without indexes. 2. Sorted - Records are physically stored in order based on one or more attributes (e.g., orderId or orderDate). Pro: Efficient for range queries and ordered scans. Cons: Insertions are expensive. 3. Hashed - Records are placed using a hash function on a key attribute (e.g., orderId). Works great for equality searches (e.g., "Find order by ID"), but bad for range queries. 4. Clustered - Records of two or more related tables are stored together in the same blocks to optimize join performance. Works great when queries often accesses data from multiple tables via join. 5. Indexed - Primary file (heap or sorted) with one or more index files to support efficient searching; Can be combined with any of the above approaches, and it is especially helpful when there are multiple access patterns (e.g., frequent lookups by orderDate, customerId).

Given that the normalized schema with Orders and OrderItem is most commonly accessed through a view that is used represent the denormalized form, which means frequent joins between Orders and OrderItem, as well as likely read-only or read-mostly access… Use a Clustered File to co-locate records from Orders and OrderItem that share the same orderId.

3.9 Buffer Replacement Policies

Question

A common buffer replacement policy/algorithm id is *least recently used (LRU).* Give two query scenarios for which the buffer manager might use a different algorithm, in which one would it use. Explain your answer.

Answer "Most operating systems use a least recently used (LRU) scheme, in which the block that was referenced least recently is written back to disk and is removed from the buffer" (Pg. 605).

1. When you are dealing with constant full Table Scan (i.e., SELECT COUNT(*) FROM Orders;), MRU (Most Recently Used), which evicts the most recently used page, is a better choice, given the assumption that you read each page only once and move on.
2. In a query like ''SELECT * FROM Customers c, Orders o WHERE c.customerId = o.customerId;' ' where Customers is small and repeatedly accessed and Orders is large, scanned fully per join iteration, we might want to adopt a pinning policy. Customers should be pinned, so they aren't evicted while reading Orders because Customers pages are reused on every join iteration.

3.10 Indexes

Question

Can a table have more than one *clustered index.*? Why?

Does a *sparse index* need to be clustered? Why?

Answer While it is possible to have more than one index of differing types (or even indices on multiple keys with a composite search key), there is only allowed to be one clustered index per table. “A clustering index is an index whose search key also defines the sequential order of the file. Clustering indices are also called primary indices” (625). Because a clustered index determines the physical order of rows in the data file itself, only one clustered index per table is allowed.

A sparse index contains only a subset of the index entries — typically one entry per data block/page, pointing to the first record in that block. While a sparse index is allowed to be clustered, there is no requirement for it to be so. Sparse indexes work best when the data is clustered, because it’s easy to find a block using just one key per block when the records are sorted and the system can scan forward efficiently once the correct block is located. However, there is no requirement.

4 Practical Questions

4.1 Some Fun with SQl Functions

4.1.1 Fun with Strings

Question

You will use Classic Models for this question.

There is a strange “dependency” in the schema for *products.* The `productCode` begins with strings like `S12_` and `S18_`. A little examination indicates that this prefix appears to be derived from the `productScale` column’s value. Unfortunately, this is NOT always the case. Write a SQL query that produces the following table.

Analyzing Product Scale

The fields are: - `productCode` is the value from `products`. - `productCodeScale` is the number in `productCode` in between `S` and `_`. - `productCodeNumber` is the value in `productCode` after the `_`. - `productScale` is the value from `products`. - `productScaleNumerator` is the value in `produceScale` before `:`. - `productScaleDenominator` is the value in `produceScale` after `:`. - `computedProductScale` is `productScaleNumerator/productScaleDenominator`.

The result contains rows for which `productCodeScale != computedProductScale`.

Write a query that produces the table.

Answer

Write and execute your query below.

[170]:

```
from sqlalchemy import create_engine, text

# Multiline SQL query wrapped in triple quotes
query = text("""
SELECT
    productCode,

    -- Extract number between 'S' and '_' in productCode
    CAST(SUBSTRING_INDEX(SUBSTRING_INDEX(productCode, '_', 1), 'S', -1) AS␣
 ↪DECIMAL(5,2)) AS productCodeScale,

    -- Extract number after '_'
    SUBSTRING_INDEX(productCode, '_', -1) AS productCodeNumber,

    productScale,

    -- Extract numerator and denominator from productScale
    CAST(SUBSTRING_INDEX(productScale, ':', 1) AS DECIMAL(5,2)) AS␣
 ↪productScaleNumerator,
    CAST(SUBSTRING_INDEX(productScale, ':', -1) AS DECIMAL(5,2)) AS␣
 ↪productScaleDenominator,

    -- Compute ratio
    CAST(SUBSTRING_INDEX(productScale, ':', 1) AS DECIMAL(5,2)) /
    CAST(SUBSTRING_INDEX(productScale, ':', -1) AS DECIMAL(5,2)) AS␣
 ↪computedProductScale

FROM
    products

-- Filter only mismatches between code-implied scale and actual computed scale
WHERE
    CAST(SUBSTRING_INDEX(SUBSTRING_INDEX(productCode, '_', 1), 'S', -1) AS␣
 ↪DECIMAL(5,2)) !=
    CAST(SUBSTRING_INDEX(productScale, ':', 1) AS DECIMAL(5,2)) /
    CAST(SUBSTRING_INDEX(productScale, ':', -1) AS DECIMAL(5,2));
""")

# Run query and fetch into DataFrame
with engine.connect() as connection:
    result = connection.execute(query)
    df = pandas.DataFrame(result.fetchall(), columns=result.keys())

# Display result
df
```

[170]:

```
    productCode productCodeScale productCodeNumber productScale  \
0      S10_1678            10.00              1678         1:10
1      S10_1949            10.00              1949         1:10
2      S10_2016            10.00              2016         1:10
3      S10_4698            10.00              4698         1:10
4      S10_4757            10.00              4757         1:10
..          …                …                 …           …
105   S700_3505           700.00              3505        1:700
106   S700_3962           700.00              3962        1:700
```

```
107   S700_4002           700.00              4002        1:700
108    S72_1253            72.00              1253         1:72
109    S72_3212            72.00              3212         1:72

    productScaleNumerator productScaleDenominator computedProductScale
0                    1.00                   10.00             0.100000
1                    1.00                   10.00             0.100000
2                    1.00                   10.00             0.100000
3                    1.00                   10.00             0.100000
4                    1.00                   10.00             0.100000
..                    …                      …                  …
105                  1.00                  700.00             0.001429
106                  1.00                  700.00             0.001429
107                  1.00                  700.00             0.001429
108                  1.00                   72.00             0.013889
109                  1.00                   72.00             0.013889

[110 rows x 7 columns]
```

4.1.2 Fun with Dates

Question

You will use Classic Models for this question.

The table `orders` has columns: 1. `customerNumber` 2. `orderNumber` 3. `orderDate` 4. `requiredDate` 5. `shippedDate`

Write a query that produces a table of the form `customerOrderSummary` with 1. `customerNumber` 2. `noOfOrders` is the number of orders from the customer. 3. `minimumShippingDays` is the minimum number of days between `shippedDate` and `orderDate` 4. `maximumShippingDays` is the maximum number of days between `shippedDate` and `orderDate` 3. `averageShippingDays` is the average number of days between `shippedDate` and `orderDate`

The table should be ordered by `averageShippingDays` descending. The various number of days must be an integer.

For reference, the first 10 rows in the result is

Shipping Days Information

Answer

Write and execute your query below.

[172]:

```
%%sql
USE classicmodels;
```

```
 * mysql+pymysql://root:***@localhost/classicmodels
   mysql+pymysql://root:***@localhost?local_infile=1
0 rows affected.
```

[172]:

```
[]
```

[173]:

```
%%sql
SELECT
    customerNumber,
    COUNT(*) AS noOfOrders,
    MIN(DATEDIFF(shippedDate, orderDate)) AS minimumShippingDays,
    MAX(DATEDIFF(shippedDate, orderDate)) AS maximumShippingDays,
    FLOOR(AVG(DATEDIFF(shippedDate, orderDate))) AS averageShippingDays
FROM
    orders
WHERE
    shippedDate IS NOT NULL
    AND orderDate IS NOT NULL
GROUP BY
    customerNumber
ORDER BY
    averageShippingDays DESC;
```

```
 * mysql+pymysql://root:***@localhost/classicmodels
   mysql+pymysql://root:***@localhost?local_infile=1
98 rows affected.
```

[173]:

```
[(148, 5, 1, 65, 14),
 (177, 2, 7, 8, 7),
 (363, 3, 4, 6, 5),
 (276, 4, 4, 6, 5),
 (240, 2, 5, 6, 5),
 (219, 2, 5, 6, 5),
 (209, 3, 5, 6, 5),
 (205, 3, 4, 6, 5),
 (204, 2, 4, 6, 5),
 (462, 3, 3, 6, 5),
 (328, 2, 4, 6, 5),
 (198, 3, 5, 6, 5),
 (455, 2, 5, 5, 5),
 (448, 2, 5, 6, 5),
 (344, 2, 5, 6, 5),
 (398, 4, 2, 8, 5),
 (161, 4, 4, 6, 5),
 (385, 3, 5, 6, 5),
 (256, 2, 2, 6, 4),
 (250, 3, 3, 6, 4),
 (350, 3, 3, 5, 4),
 (347, 2, 3, 6, 4),
 (175, 3, 4, 6, 4),
 (386, 3, 4, 6, 4),
 (181, 3, 3, 5, 4),
 (424, 3, 3, 5, 4),
 (339, 2, 2, 6, 4),
 (189, 2, 4, 5, 4),
 (456, 2, 3, 5, 4),
 (458, 3, 3, 5, 4),
 (320, 3, 2, 6, 4),
 (128, 4, 2, 5, 4),
 (471, 3, 3, 6, 4),
 (475, 2, 3, 5, 4),
 (319, 2, 3, 5, 4),
 (311, 3, 3, 6, 4),
 (484, 2, 3, 6, 4),
 (299, 2, 3, 5, 4),
 (233, 3, 2, 5, 4),
 (282, 3, 3, 5, 4),
 (486, 3, 1, 6, 4),
 (278, 3, 4, 6, 4),
 (489, 2, 4, 5, 4),
 (171, 2, 3, 5, 4),
 (496, 4, 2, 6, 3),
 (157, 3, 2, 5, 3),
 (260, 2, 3, 4, 3),
 (249, 2, 2, 5, 3),
 (487, 2, 2, 5, 3),
 (239, 2, 1, 5, 3),
 (286, 2, 2, 4, 3),
 (114, 5, 1, 5, 3),
 (119, 3, 1, 6, 3),
 (216, 3, 1, 5, 3),
 (121, 4, 3, 5, 3),
 (211, 2, 1, 5, 3),
 (202, 2, 3, 3, 3),
 (321, 4, 1, 6, 3),
 (473, 2, 2, 5, 3),
 (324, 3, 2, 4, 3),
 (141, 24, 1, 6, 3),
 (333, 3, 1, 6, 3),
 (334, 3, 3, 4, 3),
 (144, 3, 1, 5, 3),
 (447, 3, 1, 5, 3),
 (173, 2, 3, 3, 3),
 (172, 3, 1, 4, 3),
 (353, 5, 1, 6, 3),
 (146, 3, 1, 6, 3),
 (412, 3, 2, 4, 3),
 (167, 3, 1, 6, 3),
 (379, 3, 1, 4, 3),
 (406, 3, 3, 5, 3),
 (382, 4, 2, 5, 3),
 (259, 2, 2, 4, 3),
 (166, 4, 1, 4, 2),
 (151, 4, 2, 3, 2),
 (381, 4, 1, 5, 2),
 (362, 2, 1, 4, 2),
 (357, 2, 1, 4, 2),
 (145, 5, 1, 6, 2),
 (450, 3, 1, 4, 2),
 (452, 3, 2, 3, 2),
 (187, 3, 1, 3, 2),
 (131, 3, 1, 3, 2),
 (129, 3, 1, 5, 2),
 (124, 16, 1, 6, 2),
 (323, 5, 1, 3, 2),
 (227, 2, 1, 3, 2),
 (298, 2, 2, 2, 2),
 (242, 3, 2, 4, 2),
 (112, 3, 1, 4, 2),
 (495, 2, 2, 2, 2),
 (103, 3, 1, 4, 2),
 (415, 1, 1, 1, 1),
 (186, 3, 1, 3, 1),
 (201, 3, 1, 3, 1),
 (314, 2, 1, 1, 1)]
```

4.1.3 Fun

Fun

4.2 A Lot Less Fun with Functions, Procedures and Triggers

Setup

You will use the database associated with the recommended textbook for this question. The tables in scope for the question are: 1. `takes` 2. `section` 3. `classroom`

The following SQL script creates a copy of the data that you can use for this question.

[176]:

```
%%sql
USE db_book
```

```
 * mysql+pymysql://root:***@localhost/classicmodels
   mysql+pymysql://root:***@localhost?local_infile=1
0 rows affected.
```

[176]:
```
[]
```

[177]:
```
%%sql

drop schema if exists hw4;

create schema hw4;

use hw4;

create table student like db_book.student;
create table section like db_book.section;
create table classroom like db_book.classroom;
create table takes like db_book.takes;

insert into student select * from db_book.student;
insert into section select * from db_book.section;
insert into classroom select * from db_book.classroom;
insert into takes select * from db_book.takes;

update classroom set capacity=6;

create or replace view section_room_summary as
with one as (select *
             from section
                      join classroom using (building, room_number)),
two as (
        select concat(course_id, '_', sec_id, '_', semester, '_', `year`) as␣
 ↪section_code,
               one.* from takes join one using(course_id, sec_id, semester,␣
 ↪`year`)
    ),
three as (
    select section_code, building, room_number, capacity, count(*) as␣
 ↪no_of_students
    from two
    group by section_code, building, room_number
)
select * from three;

create table if not exists hw4.section_waitlist
(
    ID                varchar(5)                                  not null,
    course_id       varchar(8)                        not null,
    sec_id          varchar(8)                        not null,
    semester        varchar(6)                        not null,
    year            decimal(4)                        not null,
    added_timestamp datetime default CURRENT_TIMESTAMP not null,
    primary key (ID, course_id, sec_id, semester, year)
);

create index course_id
    on hw4.section_waitlist (course_id, sec_id, semester, year);
```

```
 * mysql+pymysql://root:***@localhost/classicmodels
   mysql+pymysql://root:***@localhost?local_infile=1
6 rows affected.
1 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
13 rows affected.
15 rows affected.
5 rows affected.
22 rows affected.
5 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
```

[177]:

```
[]
```

[178]:

```
%sql select * from section_room_summary;
```

```
 * mysql+pymysql://root:***@localhost/classicmodels
   mysql+pymysql://root:***@localhost?local_infile=1
14 rows affected.
```

[178]:

```
[('CS-101_1_Fall_2017', 'Packard', '101', Decimal('6'), 6),
 ('CS-101_1_Spring_2018', 'Packard', '101', Decimal('6'), 1),
 ('FIN-201_1_Spring_2018', 'Packard', '101', Decimal('6'), 1),
 ('MU-199_1_Spring_2018', 'Packard', '101', Decimal('6'), 1),
 ('BIO-101_1_Summer_2017', 'Painter', '514', Decimal('6'), 1),
 ('BIO-301_1_Summer_2018', 'Painter', '514', Decimal('6'), 1),
 ('HIS-351_1_Spring_2018', 'Painter', '514', Decimal('6'), 1),
 ('CS-190_2_Spring_2017', 'Taylor', '3128', Decimal('6'), 2),
 ('CS-319_2_Spring_2018', 'Taylor', '3128', Decimal('6'), 1),
 ('CS-347_1_Fall_2017', 'Taylor', '3128', Decimal('6'), 2),
 ('EE-181_1_Spring_2017', 'Taylor', '3128', Decimal('6'), 1),
 ('CS-319_1_Spring_2018', 'Watson', '100', Decimal('6'), 1),
 ('PHY-101_1_Fall_2017', 'Watson', '100', Decimal('6'), 1),
 ('CS-315_1_Spring_2018', 'Watson', '120', Decimal('6'), 2)]
```

question

First, write a trigger on `takes` that prevents an insert `takes` if an `insert` would exceed the room capacity. You can use the view above in your trigger.

[180]:

```
%%sql
CREATE TRIGGER prevent_overenrollment
BEFORE INSERT ON takes
FOR EACH ROW
BEGIN
    DECLARE current_count INT;
    DECLARE room_capacity INT;

    -- Get current enrollment for the section
    SELECT COUNT(*) INTO current_count
    FROM takes t
    JOIN section s USING (course_id, sec_id, semester, year)
    JOIN classroom c ON s.building = c.building AND s.room_number = c.
 ↪room_number
    WHERE t.course_id = NEW.course_id
      AND t.sec_id = NEW.sec_id
      AND t.semester = NEW.semester
      AND t.year = NEW.year;

    -- Get room capacity
    SELECT c.capacity INTO room_capacity
    FROM section s
    JOIN classroom c ON s.building = c.building AND s.room_number = c.
 ↪room_number
    WHERE s.course_id = NEW.course_id
      AND s.sec_id = NEW.sec_id
      AND s.semester = NEW.semester
      AND s.year = NEW.year
    LIMIT 1;

    -- If inserting this row would exceed capacity, block it
    IF current_count >= room_capacity THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Enrollment exceeds classroom capacity.';
    END IF;
END;
```

```
 * mysql+pymysql://root:***@localhost/classicmodels
   mysql+pymysql://root:***@localhost?local_infile=1
0 rows affected.
```

[180]: []

[182]:
```
%%sql
-- Success Case
INSERT INTO takes (ID, course_id, sec_id, semester, year)
VALUES ('12345', 'CS101', '1', 'Fall', 2023);
```

```
 * mysql+pymysql://root:***@localhost/classicmodels
   mysql+pymysql://root:***@localhost?local_infile=1
1 rows affected.
```

[182]: []

[184]:
```
%%sql
-- Fail Case
INSERT INTO takes (ID, course_id, sec_id, semester, year)
VALUES ('67890', 'CS101', '1', 'Fall', 2023);
```

```
 * mysql+pymysql://root:***@localhost/classicmodels
   mysql+pymysql://root:***@localhost?local_infile=1
(pymysql.err.IntegrityError) (1062, "Duplicate entry '67890-CS101-1-Fall-2023'
for key 'takes.PRIMARY'")
[SQL: INSERT INTO takes (ID, course_id, sec_id, semester, year)
VALUES ('67890', 'CS101', '1', 'Fall', 2023);]
(Background on this error at: https://sqlalche.me/e/20/gkpj)
```

The next task is to implement a procedure. The procedure's input are: 1. A string encoding of a section's information, e.g. `CS-101_1_Spring_2018`. 2. A student's ID, e.g. `00128`.

The procedures: 1. Validates that the student ID exists. 2. Computes the `course_id, sec_id, semester` and `year` from the input string. 3. Ensures that enrolling the student will not exceed the capacity of the classroom. If the enrollment would exceed the capacity, the procedure adds the student to the `section_waitlist` table.

The following is the signature of the procedure. You should implement and test the procedure.

[188]:
```
%%sql
USE db_book;
```

```
 * mysql+pymysql://root:***@localhost/classicmodels
   mysql+pymysql://root:***@localhost?local_infile=1
0 rows affected.
```

[188]: []

[189]:
```
%%sql

drop procedure if exists hw4.enroll_student;

create
```

```
    definer = root@localhost procedure hw4.enroll_student(IN section_code varchar(32), IN student_id varchar(16))
begin
    DECLARE course_id VARCHAR(8);
    DECLARE sec_id VARCHAR(8);
    DECLARE semester VARCHAR(6);
    DECLARE year INT;
    DECLARE current_enrollment INT;
    DECLARE capacity INT;
    DECLARE student_exists INT;

    -- 1. Check if student exists
    SELECT COUNT(*) INTO student_exists
    FROM hw4.student
    WHERE ID = student_id;

    IF student_exists = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Student does not exist';
    END IF;

    -- 2. Parse section_code into parts using substring functions
    -- Expected format: 'CS-101_1_Spring_2018'
    SET course_id = SUBSTRING_INDEX(section_code, '_', 1);
    SET sec_id = SUBSTRING_INDEX(SUBSTRING_INDEX(section_code, '_', 2), '_', -1);
    SET semester = SUBSTRING_INDEX(SUBSTRING_INDEX(section_code, '_', 3), '_', -1);
    SET year = CAST(SUBSTRING_INDEX(section_code, '_', -1) AS UNSIGNED);

    -- 3. Compute current enrollment
    SELECT COUNT(*) INTO current_enrollment
    FROM hw4.takes
    WHERE course_id = course_id
      AND sec_id = sec_id
      AND semester = semester
      AND year = year;

    -- 4. Get classroom capacity
    SELECT c.capacity INTO capacity
    FROM hw4.section s
    JOIN hw4.classroom c ON s.building = c.building AND s.room_number = c.room_number
    WHERE s.course_id = course_id
      AND s.sec_id = sec_id
      AND s.semester = semester
      AND s.year = year
    LIMIT 1;

    -- 5. If capacity not full, insert into takes
    IF current_enrollment < capacity THEN
        INSERT INTO hw4.takes(ID, course_id, sec_id, semester, year)
        VALUES (student_id, course_id, sec_id, semester, year);
    ELSE
        -- 6. Else insert into section_waitlist
        INSERT INTO hw4.section_waitlist(ID, course_id, sec_id, semester, year)
        VALUES (student_id, course_id, sec_id, semester, year);
    END IF;
end;
```

```
 * mysql+pymysql://root:***@localhost/classicmodels
   mysql+pymysql://root:***@localhost?local_infile=1
0 rows affected.
0 rows affected.
```

[189]: []

[190]:

```
%%sql
-- Try enrolling a valid student to a valid section
CALL hw4.enroll_student('CS-101_1_Fall_2023', '00128');
```

```
 * mysql+pymysql://root:***@localhost/classicmodels
   mysql+pymysql://root:***@localhost?local_infile=1
1 rows affected.
```

[190]: []

[191]:

```
%%sql
-- Invalid
CALL hw4.enroll_student('CS-101_1_Fall_2023', '99999');
```

```
 * mysql+pymysql://root:***@localhost/classicmodels
   mysql+pymysql://root:***@localhost?local_infile=1
(pymysql.err.OperationalError) (1644, 'Student does not exist')
[SQL: -- Invalid
CALL hw4.enroll_student('CS-101_1_Fall_2023', '99999');]
(Background on this error at: https://sqlalche.me/e/20/e3q8)
```

[ ]: