HW2_A_Common

May 15, 2025

W4111_2025_002_1: Introduction to Databases:Homework 2

1 Overview

1.1 Scope

The material in scope for this homework is: - The content of lectures 1, 2 and 3. - The slides associated with the recommended textbook for - Chapter 1. - Chapter 2. - Chapter 3. - Chapter 4 slides 4.4 to 4.13, 4.36 to 4.50 except for slide 4.35 (Transactions). - Chapter 6 slides 6.1 to 6.53.

1.2 Submission Instructions

- Due date: 2025-Feb-23, 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.

There is a post/mega-thread on Ed Discussions that we will use to resolve questions and issues with respect to homework 2.

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

[1]:
```
import copy
```

[2]:
```
import json
```

[26]:
```
import pandas
```

[5]:
```
# You should have installed the packages for previous homework assignments
#
import pymysql
import sqlalchemy
```

[8]:
```
import numpy
```

[27]:
```
# You have installed and configured ipython-sql for previous assignments.
# https://pypi.org/project/ipython-sql/
#
%load_ext sql
```

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

[28]:
```
# This is a hack to fix a version problem/incompatibility  with some of the␣
 ↪packages and magics.
#
%config SqlMagic.style = '_DEPRECATED_DEFAULT'
```

[29]:
```
# Make sure that you set these values to the correct values for your␣
 ↪installation and
# configuration of MySQL
#
db_user = "root"
db_password = "rootpass"
```

[30]:
```
# 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?local_infile=1"
```

[31]:
```
# Initialize ipython-sql
#
%sql $db_url
```

[32]:
```
# Your answer will be different based on the databases that you have created on␣
 ↪your local MySQL instance.
#
%sql show tables from db_book
```

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

[32]:
```
[('advisor',),
 ('classroom',),
 ('course',),
```

```
('department',),
('instructor',),
('prereq',),
('section',),
('student',),
('takes',),
('teaches',),
('time_slot',)]
```

[11]:
```
from sqlalchemy import create_engine
default_engine = create_engine(db_url)
```

[13]:
```
result_df = pandas.read_sql(
    "show tables from db_book", con=default_engine
)
result_df
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[13], line 1
----> 1 result_df = pandas.read_sql(
      2     "show tables from db_book", con=default_engine
      3 )
      4 result_df

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'
```

3 Written Questions

3.1 Data Types and Domains

Question

Columbia University has an online directory of classes. One of the properties in the data defining a class is the section key. The section key for our database class this spring is "20251COMS4111W002." The section key for one of this spring's Calculus I classes is "20251MATH1101V002." The "data type" for section key is clearly a text string. The domain of this attribute is related to the data type but is different. Briefly explain the concept of a domain and how it differs from a data type. Use section key and your knowledge of Columbia University to provide examples of the difference.

Answer

A data type defines the kind of values a variable or attribute can hold—such (like INT, VARCHAR, etc.) and is enforced at the system level to provide consistency, storage efficiency, and basic validation. Domain is a user defined set of acceptable values mand may extend a data type with additional constraints with additional constraints or sematic meaning. Ideally, a database relation should use atomic domains, since a relation is in First Normal Form (1NF) if all attributes of the relation have atomic (indivisible) values (Silberschatz et al., p. 342). Columbia's section key is an identifier composed of a term code, department code, course number, section type and section number. While it could serve as a unique identifier (primary key), it is semantically very dense as it encodes multiple attributes into a single string; therefore, it would be better to decompose this identifier into its component attributes, and form a composite primary key if needed.

3.2 Associative Entity

Question

When modeling a relationship between two entity sets using Crow's Foot Notation or implementing in SQL, what are the two reasons that you must use an associative entity?

Answer

Relational databases do not allow many-to-many relationships because a direct M:N relationship between two entity sets cannot be stored in a single table without losing uniqueness or introducing repeating groups, which would violate First Normal Form (1NF). Decomposing an M:N relationship into two 1:M relationships via an associative entity allows us to preserve normalization and data integrity more largely.

3.3 Recognizing Entity Types

Question

Examine the schema/SQL DDL for the sample database associated with the recommended textbook. Which tables are associative entities, and which tables are weak entities? Briefly explain your answer.

Answer

Takes, teaches, and advisor are the associative entities. Takes bridges courses to student, teaches bridges instructor to course, and advisor bridges student to instructor, all of which are strong entities. Regarding weak entities, section seems to be the only reasonable candidate because it relies on the existence of a course.

3.4 Atomic Domains

Question

The lecture 3 slides contained the following: - "Every domain must contain atomic values(smallest indivisible units) which means composite and multi-valued attributes are not allowed." - This is sometimes known as "First Normal Form." We will cover normalization later in the semester.

Briefly explain this concepts and give examples of atomic and non-atomic domains using people's names.

Answer

A domain is a set of valid values an attribute can take, and an atomic domain contains values that cannot be decomposed meaningfully into smaller values (at least within the context of the database). First Normal Form (1NF) is a principle in relational database design that requires all attributes in a relation/table have atomic domains (p. 342). Enforcing atomicity becomes import for keeping queries clear and avoiding ambiguity. In relationship to names, we can see this when we think about a legal name being decomposed in to separate attributes first_name, middle_name, last_name). This becomes important because some people might have multiple middles names or none, so want to be sure that we are considering this when dealing with names, particularly in edge cases.

3.5 Arity

Question

For set operations in the relational algebra, the relations must have the same arity. Briefly explain the concept of arity. The relational scheme definitions for student and instructor for the data schema associated with the recommended textbook are $student(ID, name, dept\_name, tot\_cred)$ and $instructor(ID, name, dept\_name, salary)$. Do these relations have the same arity?

Answer

Arity refers to the number of attributes (columns) in a relation. In order to perform set operations, we must have "compatible relations", which includes having the same arity and compatible attribute types (54). This has to do less with databases and more with how set operations function, since we need the same number of variables to perform meaningful manipulations. In the case of union, we need to be careful because the union is made positionally—and has no reference to the semantics, so a union between two entities of arity 4 might execute without actually being meaningful. For example, in executing a union on student and instructor, the fourth column of each (tot_creds and salary respectively) are placed into the same column. Because they two tables essentially share the same attributes, this might look successful, but if you look at the last column, you will see that salary has been subsumed in to the 4th column.

3.6 Complex Attributes

Question

Typical Input Data

There are six attributes in the sample data above. 1. For each attribute, specify if the attribute is: *simple* or *composite, single valued* or *multi-valued* and *derived* or *not derived.* Explain your choices. 2. For which attributes would you use a `check constraint` and explain the constraint.

Answer

nconst - simple, single-valued, not derived; while there seems to be a prefix followed by a 7-digit number, it doesn't seem meaningful to break this down. primaryName - composite, single-valued, not derived; this is a composite name but still single-valued because we have at most one name birthYear - simple, single-valued, not derived; seemingly straightforward, a direct numeric input, not calculated. If age were stored, that would be derived from this. deathYear - simple, single-valued, not derived (?) - Again, if we had age, I would say derived, but I can only assume death is manually input. primaryProfession - simple, multi-valued, not derived; it is multivalued because it is stored as a comma-separated string, though this violates 1NF. knownForTitltes - composite, multi-valued, not derived

nconst and knownForTitles could use LIKE 'nm%' or LIKE 'tt%' - CHECK (nconst LIKE 'nm%') birthYear make sure its after a certain date but not after today. CHECK (birthYear BETWEEN 1800 AND YEAR(CURDATE())) deathYear check that it comes after birthyear but after the current year CHECK (deathYear IS NULL OR deathYear >= birthYear OR deathYear <= YEAR(CURDATE()))

3.7 Relational Algebra Assignment Operator

Question

One explanation for the assignment operator is, "With the assignment operation, a query can be written as a sequential program consisting of a series of assignments followed by an expression whose value is displayed as the result of the query."

Use the assignment operator to write a program using assignments to rewrite the query

```
 course_id, course_title, prereq_course_id, prereq_title
(
    (  course_id, course_title←title, prereq_id (course   prereq)
)
 prereq_id=prereq_course_id
(  prereq_course_id←course_id, prereq_title←title (course)))
```

What are two benefits of writing complex queries using a set of statements?

Answer

C1= course_id, title, prereq_id (course prereq) C2= course_id → prereq_course_id, title → prereq_title (course)

course_id, title → course_title, prereq_course_id, prereq_title (C1 prereq_id = prereq_course_id C2)

By rewriting the complex query into a sequence of assignments joined by a single expression, we reduce the chance for errors, as well as allow for the possibility of using certain assignments in other queries.

3.8 Constraints

Question

What are four types of constraints that may apply to a single relation/table? What type of constraint can apply to more than one table?

Consider the partial logical schema below. A student *may* or *may not* have an advisor.

Briefly explain which constraints you would apply.

Constraints

Answer

From 4.4 Integrity Constraints (pg. 145) UNIQUE, where we ensure that (A_1…A_n) attributes form a superkey. NOT NULL, where we ensure that every instance of an attribute is provided an accepted value. Key constraint PRIMARY/FOREIGN - ensures that foreign keys refer to a valid attribute in another table CHECK constraint - Enforces condition/restricts values to a specified domain

In the Student DDL, I would do a FOREIGN KEY (advisor) REFERENCES Faculty(UNI); I would use UNIQUE and NOT NULL on both tables' UNI, Last_Name, and First_name, I would also put UNIQUE on email.

3.9 SELECT versus UNION

Question

In SQL, `SELECT` and `UNION` behave differently with respect to duplicates in the result set. Explain the difference.

Taking a step back, if tables have primary keys, how are duplicates in a query result even possible?

Answer

While SELECT doesn't eliminate duplicates by default (though you can modify with DISTINCT), as it returns all matching rows (including duplicates). UNION eliminates duplicates (similar to SELECT DISTINCT) over the combined results of the two queries (However UNION ALL preserves all duplicates). Although tables may have PRIMARY keys, which prevent duplicate row values on a specific attribute, duplicates can still appear in query results regarding the other attributes, as a query often projects only a subset of attributes. Additionally, joins and aggregations can further introduce duplicate rows, depending on the relationships and grouping conditions involved.

3.10 Associative Entity

Question

Consider the query below. What is required of the result of the two subqueries? What is the name for the type of subquery?

```
select
    s_id as student_id,
    (select name from student where student.ID=s_id) as student_name,
    i_id as advisor_id,
    (select name from instructor where instructor.ID=i_id) as instructor_name
from
    advisor;
```

[24]:
```
%%sql
select
s_id as student_id,
(select name from student where student.ID=s_id) as student_name,
i_id as advisor_id,
(select name from instructor where instructor.ID=i_id) as instructor_name
from
advisor;
```

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

[24]:
```
[('12345', 'Shankar', '10101', 'Srinivasan'),
 ('44553', 'Peltier', '22222', 'Einstein'),
 ('45678', 'Levy', '22222', 'Einstein'),
 ('00128', 'Zhang', '45565', 'Katz'),
 ('76543', 'Brown', '45565', 'Katz'),
 ('23121', 'Chavez', '76543', 'Singh'),
```

```
('98988', 'Tanaka', '76766', 'Crick'),
('76653', 'Aoi', '98345', 'Kim'),
('98765', 'Bourikas', '98345', 'Kim')]
```

Answer

Each subquery must return exactly one value (i.e., a single scalar result) per row in the outer query. This will work correctly only if: student.ID is a primary key (so only one name matches per s_id) and instructor.ID is a primary key (so only one name matches per i_id). Scalar subqueries (p. 106) is a subquery used in the SELECT clause, and it is expected to return a single value (i.e., scalar). It is evaluated once per row of the outer query.

4 Practical Questions

4.1 Set Operations in SQL

Question

Using the sample data associated with the recommended textbook, 1. What is wrong with the query below. 2. Write and execute a query that produces accurate results that contains all of the information.

```
select * from student where dept_name='Comp. Sci.'
union
select * from instructor where dept_name='Comp. Sci.'
```

Answer Even though both share some of the same attributes, the schemas between the two do not entirely match. Student has these columns (SID, name, dept_name, and tot_cred), whereas instructor has these (ID, name, dept_name, salary). The fact that they share some columns but not others is actually more problematic, since union matches positionally (and not by name). This could cause major confusion as the first three columns may seem generally correct with the union; however, the fourth column will be completely nonsensical.

Please place and execute your SQL statement below.

[18]:
```
%%sql
USE db_book
```

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

[18]:
```
[]
```

[22]:
```
%%sql
select * from student where dept_name='Comp. Sci.'
union
select * from instructor where dept_name='Comp. Sci.'
```

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

[22]:
```
[('00128', 'Zhang', 'Comp. Sci.', Decimal('102.00')),
 ('12345', 'Shankar', 'Comp. Sci.', Decimal('32.00')),
 ('54321', 'Williams', 'Comp. Sci.', Decimal('54.00')),
 ('76543', 'Brown', 'Comp. Sci.', Decimal('58.00')),
 ('10101', 'Srinivasan', 'Comp. Sci.', Decimal('65000.00')),
 ('45565', 'Katz', 'Comp. Sci.', Decimal('75000.00')),
 ('83821', 'Brandt', 'Comp. Sci.', Decimal('92000.00'))]
```

[20]:
```
%%sql
SELECT
  ID AS ID,
  name,
  dept_name,
  tot_cred AS info_value,
  'student' AS role
FROM student
WHERE dept_name = 'Comp. Sci.'

UNION

SELECT
  ID,
  name,
  dept_name,
  salary AS info_value,
  'instructor' AS role
FROM instructor
WHERE dept_name = 'Comp. Sci.';
```

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

[20]:
```
[('00128', 'Zhang', 'Comp. Sci.', Decimal('102.00'), 'student'),
 ('12345', 'Shankar', 'Comp. Sci.', Decimal('32.00'), 'student'),
 ('54321', 'Williams', 'Comp. Sci.', Decimal('54.00'), 'student'),
 ('76543', 'Brown', 'Comp. Sci.', Decimal('58.00'), 'student'),
 ('10101', 'Srinivasan', 'Comp. Sci.', Decimal('65000.00'), 'instructor'),
 ('45565', 'Katz', 'Comp. Sci.', Decimal('75000.00'), 'instructor'),
 ('83821', 'Brandt', 'Comp. Sci.', Decimal('92000.00'), 'instructor')]
```

4.2 Set Operations in Relational Algebra

Question

The query below produces information about instructors that are not advisors. You must write an equivalent relational algebra expression that contains only set operators and project. Replace the query and screen capture below with you answer.

Orignal: ID←ID, name←name (i_id=null (instructor ID=i_id advisor))

Answer

In order to get the left join, we can simply do the intersection and subtract the intersection from the "left" hand of instructor in order to retain all of its values not shared with advisor.

```
ID (instructor) - (  i_id(advisor)   ID (instructor))
```

Replace the images below with your screenshot.

Spring Courses: Your Answer

4.3 ER-Modeling

Question

Consider the following scenario. 1. There are two entity types: 1. `person` has attributes `last_name, first_name` and `UNI.` The primary key is `UNI.` 2. `phone_number` has the attributes `country_code, number` and `extension.` The primary key is a composite of all 3 attributes. 2. There is one relationships – `has_a` is a relationship between a `person` and `phone_number.` - A `person` may be related to 0, 1 or many `phone_numbers.` - A `phone_number` may be related to 0, 1 or many `persons.` - Each relationship has 3 properties: 1. `kind` is in the set `{home, mobile, work, voicemail, supporting_admin}.` It is possible that the `kind` is not known. 2. `valid_start_date` defines when the association started. 3. `valid_end_date` defines when the association ended.

Using Crow's Foot Notation and a tool like Lucidchart, draw a logical ER diagram modeling the relationship. You may add notes/comments that explain decisions you make.

Answer

Replace the images below with your screenshot.

Spring Courses: Your Answer

4.4 ER Diagram to DDL

Question

ER Diagram to DDL

Consider the preceding, **approximate** ER logical model diagram. The diagram is approximate because the definition below of the model may require minor changes in the implemented DDL relative to the diagram. For example, you may have to add constraints, columns not shown, etc.

The semantics/requirements are below.

A sample `person` record for me in `person` would" be in the form

`{dff9, Ferguson, Donald, Faculty, donald.ferguson@cs.columbia.edu, dff9@columbia.edu}`

- The default email is always of the form `uni@columbia.edu`.
- Preferred email is always `UNIQUE` but a person *may not have* a preferred email.
- The possible values for `kind` are one of `{Student, Faculty, Staff}`.

A sample `course` record for our *Intro. to Databases* course would be in the form

`{COMS, W, 4111, Introduction to Databases, OMG! This class is terrifying., COMSW4111}`

- `dept_code` is always 4 characters and will not contain a digit, space, -, or _
- `faculty_code` is one of `{W, C, E, B, G}`.
- `course_no` is always 4 digits and cannot begin with a 0.
- `full_course_no` is the concatenation of `dept_code, faculty_code, course_no`.

A sample `section` for our session of COMSW4111 would be

`{11969, COMSW4111, 002, 1, 2025, COMSW4111_002_1_2025}`

- `call_no` is always 5 digits and may begin with 0.
- `course_no` is the same as `full_course_no` in `course`.
- `section_no` is always 3 characters. It can be 3 digits and may start with 0. Or, it can be of the form `V02`, that is starts with `V` and has two digits.
- `year` has the obvious meaning and constraints.
- `section_key` is the concatenation of the fields with the _ delimiter.

A sample `person_section` for me would be

`{dff9, 11969, instructor, 20250125, 20250502}`

- The `role` is one of `{instructor, student, TA, auditor}`. A person may have nore than one `role` in a course.
- The `start_date` must be before the `end_date`.

Put, execute and test your DDL in the code cells below. You can explain assumptions and changes in the markdown cell that precedes the code cells.

Answer

General Assumptions: • All identifiers like UNI, call_no, and full_course_no are treated as strings. • Emails are stored as VARCHAR(40) and validated with basic LIKE patterns. • Composite identifiers like full_course_no and section_key are stored explicitly (not generated) and assumed to be computed correctly elsewhere. • All enumerated domains (kind, faculty_code, role) are enforced using CHECK (…) IN (…).

Issues: MySQL's CHECK cannot use REGEXP, so regex-like rules are not fully enforced in CHECK. I wanted to use dept_code ~ '[1]{4}$' but this won't work in MySQL.

Please place and execute your SQL statement below.

[35]:
```
%%sql
CREATE DATABASE IF NOT EXISTS test_hw2;
USE test_hw2;
```

[1] A-Z

```
-- person table
CREATE TABLE person (
    UNI VARCHAR(10) PRIMARY KEY,
    last_name VARCHAR(40) NOT NULL,
    first_name VARCHAR(40) NOT NULL,
    kind VARCHAR(10) NOT NULL,
    preferred_email VARCHAR(40) UNIQUE,
    default_email VARCHAR(40) UNIQUE NOT NULL,
    CHECK (kind IN ('Student', 'Faculty', 'Staff')),
    CHECK (default_email LIKE '%@columbia.edu')
);

-- course table
CREATE TABLE course (
    dept_code CHAR(4) NOT NULL,
    faculty_code CHAR(1) NOT NULL,
    course_no CHAR(4) NOT NULL,
    course_title VARCHAR(255) NOT NULL,
    course_description TEXT,
    full_course_no VARCHAR(12) PRIMARY KEY,
    CHECK (faculty_code IN ('W', 'C', 'E', 'B', 'G')),
   CHECK (dept_code RLIKE '^[A-Z]{4}$')
);

-- section table
CREATE TABLE section (
    call_no CHAR(5) PRIMARY KEY,
    course_no VARCHAR(12) NOT NULL,
    section_no VARCHAR(3) NOT NULL,
    semester TINYINT NOT NULL,
    year INT NOT NULL,
    section_key VARCHAR(30) UNIQUE,
    FOREIGN KEY (course_no) REFERENCES course(full_course_no),
    CHECK (semester IN (1, 2)),
    CHECK (year BETWEEN 2000 AND 2100),
    CHECK (section_no RLIKE '^[0-9]{3}$' OR section_no RLIKE '^V[0-9]{2}$')
);

-- person_section table
CREATE TABLE person_section (
    student_UNI VARCHAR(10) NOT NULL,
    callno CHAR(5) NOT NULL,
    role VARCHAR(10) NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    PRIMARY KEY (student_UNI, callno, role),
    FOREIGN KEY (student_UNI) REFERENCES person(UNI),
```

```
    FOREIGN KEY (callno) REFERENCES section(call_no),
    CHECK (role IN ('instructor', 'student', 'TA', 'auditor')),
    CHECK (start_date < end_date)
);
```

```
 * mysql+pymysql://root:***@localhost?local_infile=1
1 rows affected.
0 rows affected.
(pymysql.err.OperationalError) (1050, "Table 'person' already exists")
[SQL: -- person table
CREATE TABLE person (
    UNI VARCHAR(10) PRIMARY KEY,
    last_name VARCHAR(40) NOT NULL,
    first_name VARCHAR(40) NOT NULL,
    kind VARCHAR(10) NOT NULL,
    preferred_email VARCHAR(40) UNIQUE,
    default_email VARCHAR(40) UNIQUE NOT NULL,
    CHECK (kind IN ('Student', 'Faculty', 'Staff')),
    CHECK (default_email LIKE '%%@columbia.edu')
);]
(Background on this error at: https://sqlalche.me/e/20/e3q8)
```

Place some SELECT and INSERT SQL statements below that demonstrate the correctness of your schema implementation. You will likely need more than 3 tests. When you ask me how many tests you should write, I am going to respond, "Really? You need to do enough tests to show that your DDL is correct."

[51]:
```
%%sql
-- Test 1: Insert a valid person record with both emails
INSERT INTO person
VALUES ('dff9', 'Ferguson', 'Donald', 'Faculty', 'donald.ferguson@cs.columbia.
 ↪edu', 'dff9@columbia.edu');
```

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

[51]:
```
[]
```

[52]:
```
%%sql
-- Test 2: Insert a person without a preferred_email (NULL allowed)
INSERT INTO person
VALUES ('js123', 'Smith', 'Jane', 'Student', NULL, 'js123@columbia.edu');
```

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

[52]:
```
[]
```

[53]:
```
%%sql
-- Test 3: Insert a valid course
INSERT INTO course
VALUES ('COMS', 'W', '4111', 'Introduction to Databases', 'OMG! This class is␣
 ↪terrifying.', 'COMSW4111');
```

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

[53]:
```
[]
```

[59]:
```
%%sql
-- Test 4: Insert a valid section (3-digit section number)
INSERT INTO section
VALUES ('11969', 'COMSW4111', '002', 1, 2025, 'COMSW4111_002_1_2025');
```

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

[59]:
```
[]
```

[68]:
```
%%sql
-- Test 5: Insert another valid section (Vnn section number)
INSERT INTO section
VALUES ('12001', 'COMSW4111', 'V02', 2, 2025, 'COMSW4111_V02_2_2025');
```

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

[68]:
```
[]
```

[61]:
```
%%sql
-- Test 6: Insert valid roles in person_section
INSERT INTO person_section
VALUES ('dff9', '11969', 'instructor', '2025-01-25', '2025-05-02');

INSERT INTO person_section
VALUES ('js123', '11969', 'student', '2025-01-27', '2025-05-01');
```

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

[61]:
```
[]
```

[62]:
```
%%sql
-- Test 7: Insert with an invalid role → should fail
INSERT INTO person_section
VALUES ('js123', '11969', 'observer', '2025-01-27', '2025-05-01');
```

```
 * mysql+pymysql://root:***@localhost?local_infile=1
(pymysql.err.OperationalError) (3819, "Check constraint 'person_section_chk_1'
is violated.")
[SQL: -- Test 7: Insert with an invalid role → should fail
INSERT INTO person_section
VALUES ('js123', '11969', 'observer', '2025-01-27', '2025-05-01');]
(Background on this error at: https://sqlalche.me/e/20/e3q8)
```

[67]:
```
%%sql
-- Test 8: Insert a section_no that violates the format (e.g. "Z01") → should␣
 ↪fail
INSERT INTO section
VALUES ('13000', 'COMSW4111', 'Z01', 1, 2025, 'COMSW4111_Z01_1_2025');
```

```
 * mysql+pymysql://root:***@localhost?local_infile=1
(pymysql.err.OperationalError) (3819, "Check constraint 'section_chk_3' is
violated.")
[SQL: -- Test 8: Insert a section_no that violates the format (e.g. "Z01") →
should fail
INSERT INTO section
VALUES ('13000', 'COMSW4111', 'Z01', 1, 2025, 'COMSW4111_Z01_1_2025');]
(Background on this error at: https://sqlalche.me/e/20/e3q8)
```

[69]:
```
%%sql
-- Join across all tables to display instructor and student info
SELECT
    ps.role,
    p.first_name,
    p.last_name,
    c.course_title,
    s.section_no,
    s.year
FROM person_section ps
JOIN person p ON ps.student_UNI = p.UNI
JOIN section s ON ps.callno = s.call_no
JOIN course c ON s.course_no = c.full_course_no;
```

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

[69]:
```
[('student', 'Jane', 'Smith', 'Introduction to Databases', '002', 2025),
 ('instructor', 'Donald', 'Ferguson', 'Introduction to Databases', '002', 2025)]
```

4.5 SQL DML

Question

Write an SQL query that uses subqueries and does not use `JOIN` to produce a table of the form: - `student_id` - `student_name` - `student_dept_name` - `section_key`, which is a concatenation of `course_id, sec_id, semester, year` and uses `_` as the delimeter.

The result should only contain students in the `'Comp. Sci.'` department.

You should be able to figure this out from the description and examining the `db_book` data you installed. But, to simplify: 1. Use the tables `takes` and `student`. 2. The result of my implementation is below.

Query Result

Answer

[70]:
```
%%sql
USE db_book;
SELECT
    s.ID AS student_id,
    (SELECT name FROM student WHERE student.ID = s.ID) AS student_name,
    (SELECT dept_name FROM student WHERE student.ID = s.ID) AS␣
 ↪student_dept_name,
    CONCAT(course_id, '_', sec_id, '_', semester, '_', year) AS section_key
FROM
    takes s
WHERE
    (SELECT dept_name FROM student WHERE student.ID = s.ID) = 'Comp. Sci.';
```

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

[70]:
```
[('00128', 'Zhang', 'Comp. Sci.', 'CS-101_1_Fall_2017'),
 ('12345', 'Shankar', 'Comp. Sci.', 'CS-101_1_Fall_2017'),
 ('54321', 'Williams', 'Comp. Sci.', 'CS-101_1_Fall_2017'),
 ('76543', 'Brown', 'Comp. Sci.', 'CS-101_1_Fall_2017'),
 ('12345', 'Shankar', 'Comp. Sci.', 'CS-190_2_Spring_2017'),
 ('54321', 'Williams', 'Comp. Sci.', 'CS-190_2_Spring_2017'),
 ('12345', 'Shankar', 'Comp. Sci.', 'CS-315_1_Spring_2018'),
 ('76543', 'Brown', 'Comp. Sci.', 'CS-319_2_Spring_2018'),
 ('00128', 'Zhang', 'Comp. Sci.', 'CS-347_1_Fall_2017'),
 ('12345', 'Shankar', 'Comp. Sci.', 'CS-347_1_Fall_2017')]
```

[ ]: