COMS W 4111-002
W4111 - Introduction to Databases
Section 003/V03, Fall 2022
Take Home Final

Exam Instructions

- We will publish instructions on Ed.

Environment Setup and Test

MySQL

- Replace `root` and `dbuserdbuser` for the correct values for you MySQL instance from previous homework assignments and exams.
- You will need the sample database that comes with the recommended textbook to execute the setup test.
  - You should have already installed the database because you need for previous assignments.
  - I named my database

In [1]:
```
%load_ext sql
```

In [2]:
```
%sql mysql+pymysql://root:dbuserbdbuser@localhost
```

In [3]:
```
%sql select * from db_book.student
```

```
 * mysql+pymysql://root:***@localhost
13 rows affected.
```

Out[3]:

ID	name	dept_name	tot_cred
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

Neo4j

- Please set the values for your Neo4j database below.
- Make sure that your database is active. If you have not used it for a while, you need to log in through the website and restart the database.

In [4]:
```
neo4j_url = "neo4j+s://25653554.databases.neo4j.io"
neo4j_user = "neo4j"
neo4j_password = 'mPQJhDh6Yrauk-Y9AF9yiNRvlQ-X3ciRMUES4jd0-5c'
```

In [5]:
```
from py2neo import Graph
```

In [6]:
```
def t1():
    graph = Graph(neo4j_url, auth=(neo4j_user, neo4j_password))
    q = "match (r:Person) where r.name='Tom Hanks' return r"
    res = graph.run(q)

    for r in res:
        print(r)
```

- Please rerun the following cell.

In [17]:
```
t1()
```

```
Node('Person', born=1956, name='Tom Hanks')
```

MongoDB

- Please set your URL for MongoDB Atlas and make sure that your cluster is not suspended.

In [8]:
```
mongodb_url = "mongodb+srv://wdas:Hn6X3n5VrHdRB8e@cluster0.kkpls2i.mongodb.net/test"
```

In [9]:
```
import pymongo
```

In [10]:
```
def connect():
    client = pymongo.MongoClient(
        mongodb_url
    )
    return client


def t_connect():
    c = connect()
    print("Databases = ", list(c.list_databases()))
```

In [11]:
```
#
# Note, you list of local databases will be different. The values do not matter.
#
t_connect()
```

```
Databases =  [{'name': 'hw4', 'sizeOnDisk': 430080, 'empty': False}, {'name': 'testdb', 'sizeOnDisk': 49152,
'empty': False}, {'name': 'w4111_final', 'sizeOnDisk': 352256, 'empty': False}, {'name': 'admin', 'sizeOnDis
k': 385024, 'empty': False}, {'name': 'local', 'sizeOnDisk': 6253371392, 'empty': False}]
```

Written Questions — General Knowledge

- The written questions require a short, succinct answer.
- Remember, "If you can't explain it simply, you don't understand it well enough."
- Some questions will research using the web, lecture slides, etc. You cannot cut and paste from sources. Your answer must show that you read the material and understand the concept.
- If you use a source other than lecture material, please provide a URL to the source(s) you read.

G1

Question: List at least two reasons why database systems support data manipulation using a declarative query language such as SQL, instead of just providing a library of C or C + + functions to carry out data manipulation.

Answer:

- Declarative languages are simpler to use and understand, as they require users to specify what they want achieved, as opposed to how—this allows for more concise and complex queries that can be made in an intuitive way.
- Declarative languages make it easier to maintain and alter code, since they are formatted concisely and at a higher level of abstraction.
- Declarative languages allow for flexible queries, and can potentially apply to different types of data in different settings.

G2

Question: List four significant differences between:

- Processing data by writing programs that manipulate files.
- Using a database management system and query language.

Answer:

- **Data storage**: A DBMS provides a structured format for manipulating data with tables and relations, allowing for the use of dynamic queries.
- **Data performance**: A DBMS provides a query language to allow for dynamic, efficient, and faster queries to access and update data in the database, as opposed to file systems.
- **Data recovery**: A DBMS comes with features to backup and recover data if it is lost, while file systems generally do not.
- **Data concurrency**: A DBMS allows for tables and data to be modified and accessed by multiple users concurrently, and features to enable this, such as locks and transactions.

Source: https://www.geeksforgeeks.org/difference-between-file-system-and-dbms/

G3

Question: List five responsibilities (functionality provided) of a database-management system. For each responsibility, explain the potential problems that would occur with the functionality.

Answer:

- **Data Security**:
  - Protects access and manipulation of data from unauthorized users.
  - Potential problems: data breaches and corruption by unauthorized users, and potentially leaking of access codes/passwords.
- **Data Integrity**:
  - Enforces constraints and rules to control data in tables and relations between data, allowing for consistent and reliable data.
  - Potential problems: incorrect updating of data, leading to inconsistency, data loss, or creating faulty constraints that lead to inconsistent data.
- **Data Concurrency**:
  - Allows for concurrent access and modification of data using locks and transactions.
  - Potential problems: conflicting modification of data could lead to inconsistent data, as well as performance decrease when controlling for concurrency.
- **Data Recovery**:

    - Allows for backups of data in case data is lost, using restore functions and transaction logs.
    - Potential problems: If data is lost and new data is added to database, when recovering lost data it could conflict with the current data in database and cause inconsistencies; backup process may fail, cause errors, or run incompletely.
- **Data Querying**:
    - Allows for dynamic querying of data using query languages, and indexing of data to manage efficient querying.
    - Potential problems: performance decrease with lots of indexes and large tables, incorrect queries leading to incorrect results, inconsistent use of indexes.

G4

Question: We all use SSOL to choose and register for classes. Another option would be to have a single Google sheet (shared spreadsheet) that we all use to register for classes. What are problems with using a shared spreadsheet?

Answer:

- Shared spreadsheet would not be able to be manipulated concurrently in an efficient way, leading to potentially inconsistent and conflicting updates.
- Data may be vulnerable to security breaches and unauthorized access to sensitive information.
- Shared spreadsheets may not be able to scale well with large amounts of users and data, and cause performance loss.
- Shared spreadsheets may not provide backup mechanisms to retrieve lost data.
- Shared spreadsheets may not provide efficient querying mechanisms to retrieve data.

G5

Question: NoSQL databases have become increasingly popular for supporting applications. List 3 benefits of or reasons for using NoSQL databases versus SQL/relational databases. List 3 benefits of relational databases versus NoSQL databases.

Answer:

NoSQL over SQL/relational:

- **Faster performance**: NoSQL databases are generally faster and have higher performance through queries on unstructured data that do no require the use of joins on large tables as in SQL.

- **Horizontal scalability**: NoSQL databases can scale horizontally when needing to add commodity servers, whereas SQL/relational scales generally scale vertically, requiring migrations to larger and more expensive servers.
- **Greater flexibility**: NoSQL databases often allow for more flexible creation of schemas, as well as continuous and quick adding and updating of databases based on changing requirements.

SQL/relational over NoSQL:

- **ACID Transactions**: MySQL provides strong data integrity, ensuring that data is atomic, consisteny, isolated, and durable (ACID), and overall more reliable.
- **Structured Data**: MySQL provides strucuted data in relational, tabular format, with columns, rows, and keys, allowing for better data integrity, querying, and manipulation.
- **Dynamic Querying**: MySQL databases provide a dynamic and complex querying language, allowing for concise and effective queries and manipulations of structured data.

Source: https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/relational-vs-nosql-data, https://www.mongodb.com/nosql-explained/nosql-vs-sql

Relational Model

R1

Question: A column in a relation (table) has a *type*. Consider implementing a `date` as `CHAR(10)` in the format `YYYY-MM-DD.` The lecture material states that attributes (column values) come from a *domain*. Using `date` explain the differenc between a *domain* and a *type*.

Answer:

- A domain is the range of all possible values that a column attribute can take on: the domain for `date`, for example, would be all the possible values of type `CHAR(10)` that can take on the format `YYYY-MM-DD`.
- A type, in contrast, specifies the types of values that a column attribute can hold: for `date`, the type would be `CHAR(10)` in the format of `YYYY-MM-DD`, signifying that `date` can be a string of fixed length 10 in the specified format.

R2

Question: The domain for a relation (table) attribute (column) should be *atomic*. Why?

Answer:

- The domain should be atomic to allow for easier data processing, manipulation, and entry—atomicity allows for quicker and easier indexing, querying, and enforcing of constraints by treating each value in a domain as a single unit.

R3

Question: "In the US Postal System, a delivery point is a specific set of digits between 00 and 99 assigned to every address. When combined with the ZIP + 4 code, the delivery point provides a unique identifier for every deliverable address served by the United States Postal Service."

The lecture 2 slides provide a notation for representing a relation's schema. Assume we want to define a relation for US mailing addresses, and that the columns are:

- Zip code
- +4 code
- delivery_point
- address_line_1
- address_line_2
- city
- state

Use the notation to define the schema for an address. A simple example of an address's column values might be:

- Zip code: 10027
- +4 code: 6623
- delivery_point: 99
- address_line_1: 520 W 120th St
- address_line_2: Room 402
- city: New York

- state: NY

Answer:

address (zip_code, plus_four_code, delivery_point, address_line_1, address_line_2, city, state)

R4

Note: Use the RelaX calculator and the schema associated with the recommended textbook to answer this question. Your answer should contain:

- The text for the query.
- An image showing the query execution and result.

An example of the format is:

Query

```
σ capacity >= 50 (classroom)
```

Execution

Question: Translate the following SQL statement into an equivalent relational algebra statement.

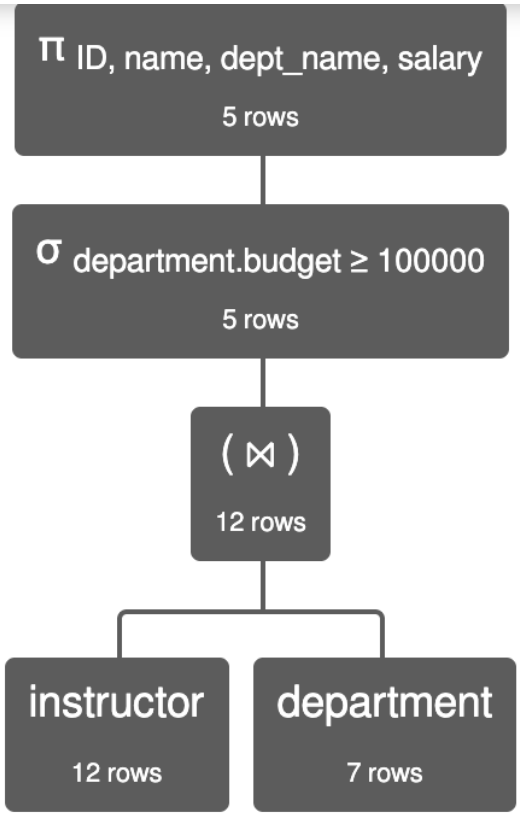
```
select
    *
from
    instructor
where
    dept_name in (select dept_name from department where budget >= 100000)
```

Answer:

πID, name, dept_name, salary σdepartment.budget ≥ 100000 (instructor ⋈ department)

In [47]:
```
from IPython.display import Image
Image("r4.png")
```

Out[47]:



$$\pi_{\text{ID, name, dept\_name, salary}} \sigma_{\text{department.budget} \geq 100000} (\text{instructor} \bowtie \text{department})$$

Execution time: 1 ms

instructor.ID	instructor.name	instructor.dept_name	instructor.salary
10101	'Srinivasan'	'Comp. Sci.'	65000

12121	'Wu'	'Finance'	90000
45565	'Katz'	'Comp. Sci.'	75000
76543	'Singh'	'Finance'	80000
83821	'Brandt'	'Comp. Sci.'	92000

R5

Use the same format to answer this question.

Question

Use the following query to compute a new table.

```
section_and_time =
    π course_id, sec_id, semester, year,
    day, start_hr, start_min (section ⋈ time_slot)
```

Using only section_and_time, write a relational algebra expression that returns a relation of overlapping courses of the form

`(course_id_1, sec_id_1, semester_1, year_1, course_id_2, sec_id_2, semester_2, year_2)`.

Your table cannot container duplicates. For example, a result containing

```
(BIO-101, 1, fall, 2022, MATH-101, 2, fall, 2022)
(MATH-101, 2, fall, 2022, BIO-101, 1, fall, 2022)
```

is incorrect.

Answer:

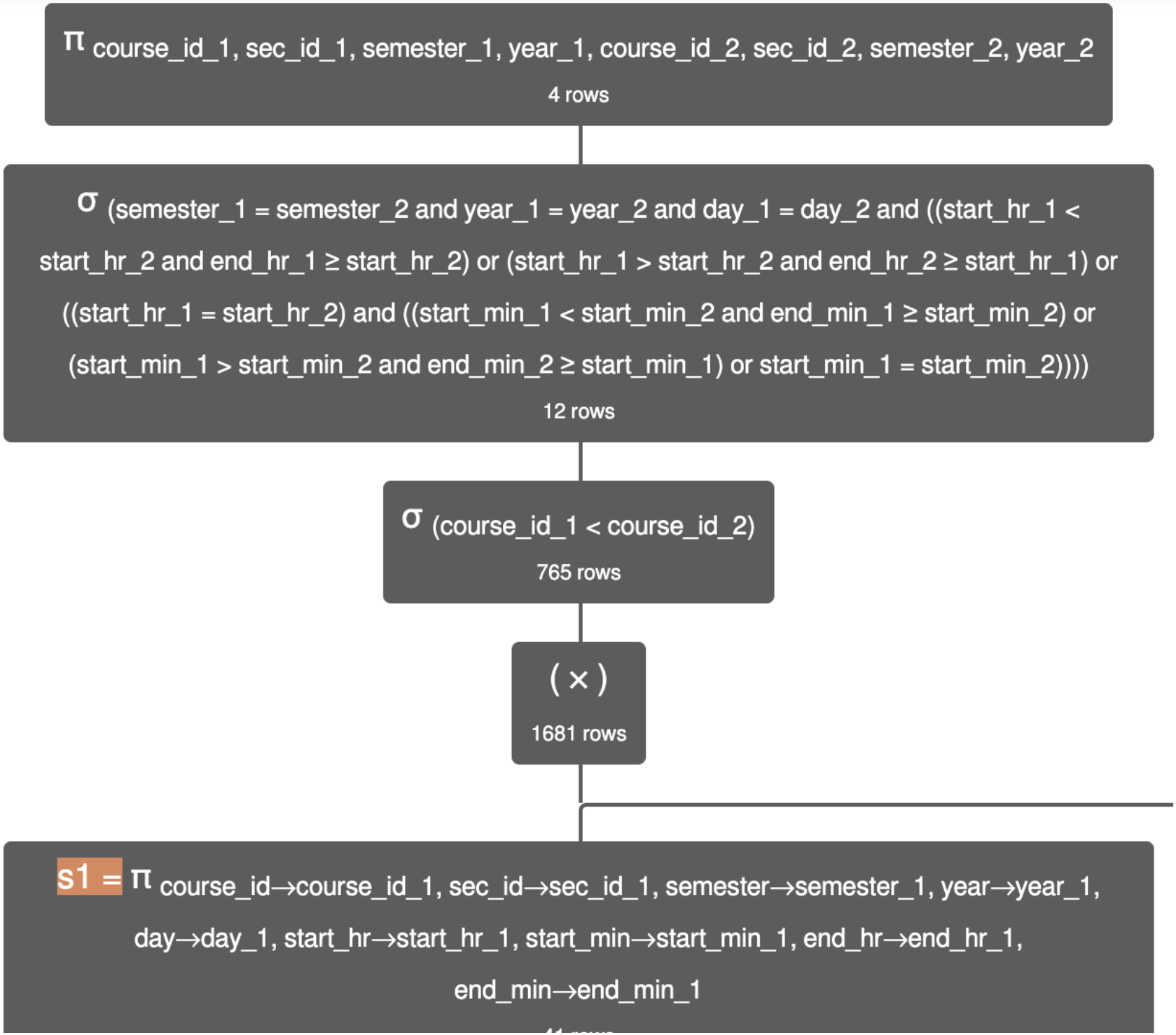
Query

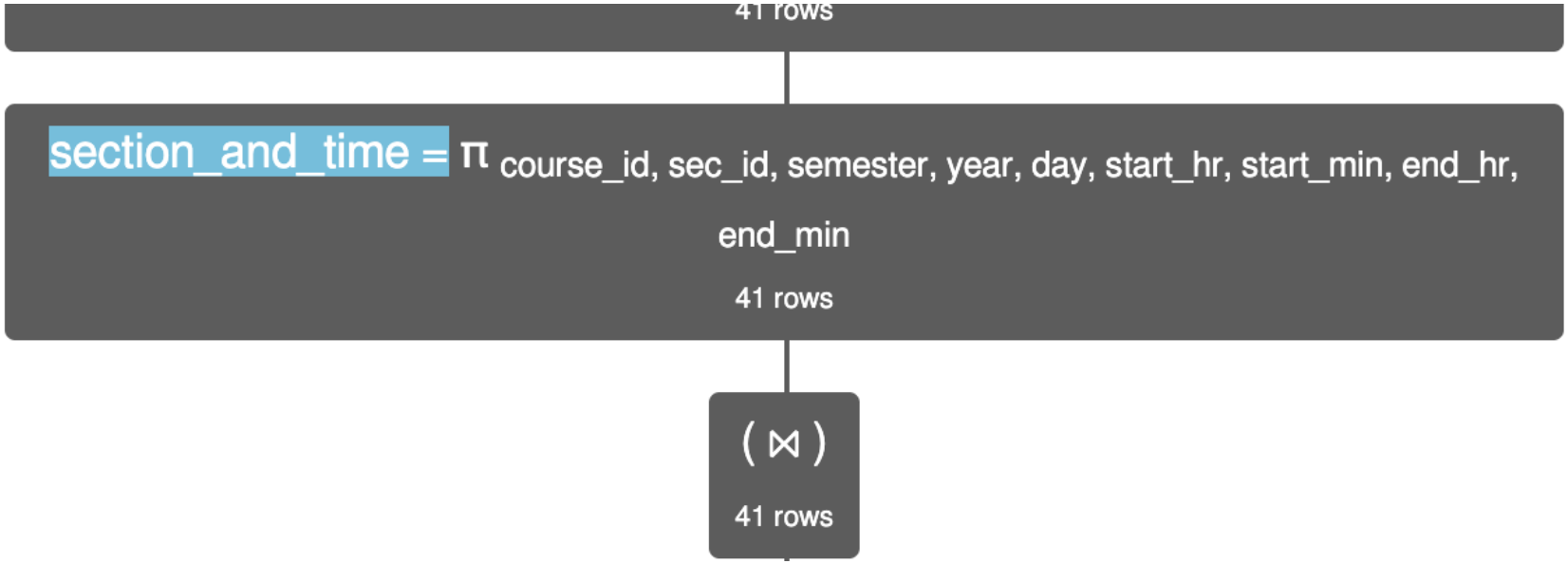
π course_id_1, sec_id_1, semester_1, year_1, course_id_2, sec_id_2, semester_2, year_2 σ (semester_1 = semester_2 and year_1 = year_2 and day_1 = day_2 and ((start_hr_1 < start_hr_2 and end_hr_1 ≥ start_hr_2) or (start_hr_1 > start_hr_2 and end_hr_2 ≥ start_hr_1) or ((start_hr_1 = start_hr_2) and ((start_min_1 < start_min_2 and end_min_1 ≥ start_min_2) or (start_min_1 > start_min_2 and end_min_2 ≥ start_min_1) or start_min_1 = start_min_2)))) (σ (course_id_1 < course_id_2) (π course_id→course_id_1, sec_id→sec_id_1, semester→semester_1, year→year_1, day→day_1, start_hr→start_hr_1, start_min→start_min_1, end_hr→end_hr_1, end_min→end_min_1 π course_id, sec_id, semester, year, day, start_hr, start_min, end_hr, end_min (section ⋈ time_slot) × π course_id→course_id_2, sec_id→sec_id_2, semester→semester_2, year→year_2, day→day_2, start_hr→start_hr_2, start_min→start_min_2, end_hr→end_hr_2, end_min→end_min_2 π course_id, sec_id, semester, year, day, start_hr, start_min, end_hr, end_min (section ⋈ time_slot)))

Execution

In [43]:
```
Image("r5_1.png")
```

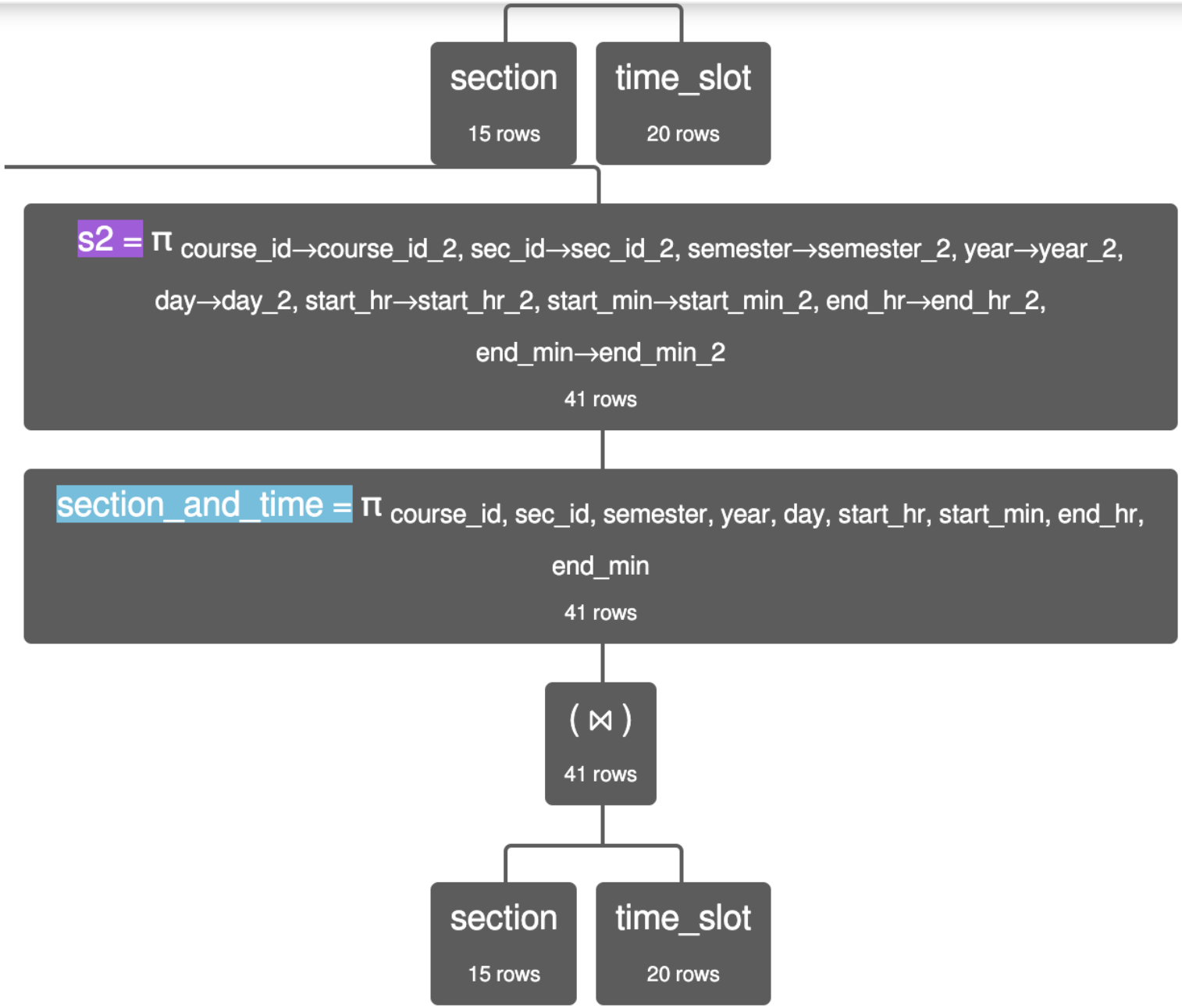
Out[43]:


π course_id_1, sec_id_1, semester_1, year_1, course_id_2, sec_id_2, semester_2, year_2
4 rows
σ (semester_1 = semester_2 and year_1 = year_2 and day_1 = day_2 and ((start_hr_1 < start_hr_2 and end_hr_1 ≥ start_hr_2) or (start_hr_1 > start_hr_2 and end_hr_2 ≥ start_hr_1) or ((start_hr_1 = start_hr_2) and ((start_min_1 < start_min_2 and end_min_1 ≥ start_min_2) or (start_min_1 > start_min_2 and end_min_2 ≥ start_min_1) or start_min_1 = start_min_2))))
12 rows
σ (course_id_1 < course_id_2)
765 rows
(×)
1681 rows
s1 = π course_id→course_id_1, sec_id→sec_id_1, semester→semester_1, year→year_1, day→day_1, start_hr→start_hr_1, start_min→start_min_1, end_hr→end_hr_1, end_min→end_min_1



In [44]:
```
Image("r5_2.png")
```

Out[44]:



π course_id_1, sec_id_1, semester_1, year_1, course_id_2, sec_id_2, semester_2, year_2 σ (semester_1 = semester_2 and year_1 = year_2 and day_1 = day_2 and

year_2 σ (semester_1 = semester_2 and year_1 = year_2 and day_1 = day_2 and ((start_hr_1 < start_hr_2 and end_hr_1 ≥ start_hr_2) or (start_hr_1 > start_hr_2 and end_hr_2 ≥ start_hr_1) or ((start_hr_1 = start_hr_2) and ((start_min_1 < start_min_2 and end_min_1 ≥ start_min_2) or (start_min_1 > start_min_2 and end_min_2 ≥ start_min_1) or start_min_1 = start_min_2)))) (σ (course_id_1 < course_id_2) (π course_id→course_id_1, sec_id→sec_id_1, semester→semester_1, year→year_1, day→day_1, start_hr→start_hr_1, start_min→start_min_1, end_hr→end_hr_1, end_min→end_min_1 π course_id, sec_id, semester, year, day, start_hr, start_min,

In [45]:
```
Image("r5_3.png")
```

Out[45]:

course_id_1	sec_id_1	semester_1	year_1	course_id_2	sec_id_2	semester_2
'CS-315'	1	'Spring'	2010	'MU-199'	1	'Spring'
'CS-319'	1	'Spring'	2010	'FIN-201'	1	'Spring'
'CS-319'	2	'Spring'	2010	'HIS-351'	1	'Spring'
'CS-347'	1	'Fall'	2009	'PHY-101'	1	'Fall'

‹ 1 ›

In [46]:
```
Image("r5_4.png")
```

Out[46]:

_1	sec_id_1	semester_1	year_1	course_id_2	sec_id_2	semester_2	year_2
	1	'Spring'	2010	'MU-199'	1	'Spring'	2010
	1	'Spring'	2010	'FIN-201'	1	'Spring'	2010
	2	'Spring'	2010	'HIS-351'	1	'Spring'	2010
	1	'Fall'	2009	'PHY-101'	1	'Fall'	2009

‹ 1 ›

SQL

- You will use the Classic Models tutorial database, which you should have already loaded into MySQL.

S1

Question: Create a view `employee_customer_sales` with the following information:

- `employeeNumber`
- `employeeLastname`
- `employeeFirstName`
- `customerNumber`
- `customerName`
- `revenue`

- The employee information is for the employee that is the `customer.customerRepEmployeeNumber`.

- `revenue` is the total revenue over all of the customer's orders.
  - The revenue for an `order` is `priceEach*quantityOrdered` for each `orderdetails` in the order.

Answer:

In [12]:
```
%%sql
USE classicmodels;

CREATE OR REPLACE VIEW employee_customer_sales AS
SELECT
    employeeNumber,
    lastName AS employeeLastname,
    firstName AS employeeFirstName,
    customerNumber,
    customerName,
    SUM(quantityOrdered * priceEach) AS revenue
FROM orderdetails
NATURAL JOIN
    orders
NATURAL JOIN
    customers
INNER JOIN
    employees
    ON employees.employeeNumber = customers.salesRepEmployeeNumber
GROUP BY
    customerNumber
ORDER BY
    employeeLastname ASC,
    revenue DESC;
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
```

Out[12]:
```
[]
```

Test Answer:

In [13]:
```
%sql select * from employee_customer_sales;
```

```
 * mysql+pymysql://root:***@localhost
98 rows affected.
```

Out[13]:

employeeNumber	employeeLastname	employeeFirstName	customerNumber	customerName	revenue
1337	Bondur	Loui	146	Saveley & Henriot, Co.	130305.35
1337	Bondur	Loui	353	Reims Collectables	126983.19
1337	Bondur	Loui	172	La Corne D'abondance, Co.	86553.52
1337	Bondur	Loui	406	Auto Canal+ Petit	86436.97
1337	Bondur	Loui	350	Marseille Mini Autos	71547.53
1337	Bondur	Loui	250	Lyon Souveniers	67659.19
1501	Bott	Larry	187	AV Stores, Co.	148410.09
1501	Bott	Larry	201	UK Collectables, Ltd.	106610.72
1501	Bott	Larry	334	Suominen Souveniers	103896.74
1501	Bott	Larry	311	Oulu Toy Supplies, Inc.	95706.15
1501	Bott	Larry	186	Toys of Finland, Co.	95546.46
1501	Bott	Larry	324	Stylish Desk Decors, Co.	80556.73
1501	Bott	Larry	240	giftsbymail.co.uk	71783.75
1501	Bott	Larry	489	Double Decker Gift Stores, Ltd	29586.15
1401	Castillo	Pamela	382	Salzburg Collectables	137480.07
1401	Castillo	Pamela	145	Danish Wholesale Imports	129085.12
1401	Castillo	Pamela	278	Rovelli Gifts	127529.69
1401	Castillo	Pamela	386	L'ordine Souveniers	125505.57
1401	Castillo	Pamela	227	Heintze Collectables	89909.80
1401	Castillo	Pamela	249	Amica Models & Co.	82223.23
1401	Castillo	Pamela	314	Petit Auto	70851.58
1401	Castillo	Pamela	452	Mini Auto Werke	51059.99
1401	Castillo	Pamela	381	Royale Belge	29217.18
1401	Castillo	Pamela	473	Frau da Collezione	25358.32
1188	Firrelli	Julie	320	Mini Creations Ltd.	101872.52
1188	Firrelli	Julie	379	Collectables For Less Inc.	73533.65

1188	Firrelli	Julie	495	Diecast Collectables	65541.74
1188	Firrelli	Julie	339	Classic Gift Ideas, Inc	57939.34
1188	Firrelli	Julie	204	Online Mini Collectables	55577.26
1188	Firrelli	Julie	173	Cambridge Collectables Co.	32198.69
1611	Fixter	Andy	114	Australian Collectors, Co.	180585.07
1611	Fixter	Andy	276	Anna's Decorations, Ltd	137034.22
1611	Fixter	Andy	282	Souveniers And Things Co.	133907.12
1611	Fixter	Andy	471	Australian Collectables, Ltd	55866.02
1611	Fixter	Andy	333	Australian Gift Network, Co	55190.16
1702	Gerard	Martin	458	Corrida Auto Replicas, Ltd	112440.09
1702	Gerard	Martin	298	Vida Sport, Ltd	108777.92
1702	Gerard	Martin	216	Enaco Distributors	68520.47
1702	Gerard	Martin	484	Iberia Gift Imports, Corp.	50987.85
1702	Gerard	Martin	344	CAF Imports	46751.14
1370	Hernandez	Gerard	141	Euro+ Shopping Channel	820689.54
1370	Hernandez	Gerard	119	La Rochelle Gifts	158573.12
1370	Hernandez	Gerard	209	Mini Caravy	75859.32
1370	Hernandez	Gerard	171	Daedalus Designs Imports	61781.70
1370	Hernandez	Gerard	242	Alpha Cognac	60483.36
1370	Hernandez	Gerard	256	Auto Associés & Cie.	58876.41
1370	Hernandez	Gerard	103	Atelier graphique	22314.36
1165	Jennings	Leslie	124	Mini Gifts Distributors Ltd.	591827.34
1165	Jennings	Leslie	450	The Sharp Gifts Warehouse	143536.27
1165	Jennings	Leslie	321	Corporate Gift Ideas Co.	132340.78
1165	Jennings	Leslie	161	Technics Stores Inc.	104545.22
1165	Jennings	Leslie	129	Mini Wheels Co.	66710.56
1165	Jennings	Leslie	487	Signal Collectibles Ltd.	42570.37

1504	Jones	Barry	448	Scandinavian Gift Ideas	120943.53
1504	Jones	Barry	121	Baane Mini Imports	104224.79
1504	Jones	Barry	167	Herkku Gifts	97562.47
1504	Jones	Barry	259	Toms Spezialitäten, Ltd	89223.14
1504	Jones	Barry	128	Blauer See Auto, Co.	75937.76
1504	Jones	Barry	299	Norway Gifts By Mail, Co.	69059.04
1504	Jones	Barry	144	Volvo Model Replicas, Co	66694.82
1504	Jones	Barry	189	Clover Collections, Co.	49898.27
1504	Jones	Barry	415	Bavarian Collectables Imports, Co.	31310.09
1612	Marsh	Peter	323	Down Under Souveniers, Inc	154622.08
1612	Marsh	Peter	496	Kelly's Gift Shop	137460.79
1612	Marsh	Peter	166	Handji Gifts& Co	107746.75
1612	Marsh	Peter	357	GiftsForHim.com	94431.76
1612	Marsh	Peter	412	Extreme Desk Decorations, Ltd	90332.38
1621	Nishi	Mami	148	Dragon Souveniers, Ltd.	156251.03
1621	Nishi	Mami	398	Tokyo Collectables, Ltd	105548.73
1621	Nishi	Mami	385	Cruz & Sons Co.	87468.30
1621	Nishi	Mami	177	Osaka Souveniers Co.	62361.22
1621	Nishi	Mami	211	King Kong Collectables, Co.	45480.79
1216	Patterson	Steve	363	Online Diecast Creations Co.	116449.29
1216	Patterson	Steve	157	Diecast Classics Inc.	104358.69
1216	Patterson	Steve	286	Marta's Replicas Co.	90545.37
1216	Patterson	Steve	462	FunGiftIdeas.com	88627.49
1216	Patterson	Steve	362	Gifts4AllAges.com	84340.32
1216	Patterson	Steve	198	Auto-Moto Classics Inc.	21554.26
1166	Thompson	Leslie	205	Toys4GrownUps.com	93803.30
1166	Thompson	Leslie	239	Collectable Mini Designs Co.	80375.24

1166	Thompson	Leslie	112	Signal Gift Stores	80180.98
1166	Thompson	Leslie	475	West Coast Collectables Co.	43748.72
1166	Thompson	Leslie	347	Men 'R' US Retailers, Ltd.	41506.19
1166	Thompson	Leslie	219	Boards & Toys Co.	7918.60
1286	Tseng	Foon Yue	151	Muscle Machine Inc	177913.95
1286	Tseng	Foon Yue	181	Vitachrome Inc.	72497.64
1286	Tseng	Foon Yue	455	Super Scale Inc.	70378.65
1286	Tseng	Foon Yue	424	Classic Legends Inc.	69214.33
1286	Tseng	Foon Yue	233	Québec Home Shopping Network	68977.67
1286	Tseng	Foon Yue	456	Microscale Inc.	29230.43
1323	Vanauf	George	131	Land of Toys Inc.	149085.15
1323	Vanauf	George	175	Gift Depot Inc.	95424.63
1323	Vanauf	George	328	Tekni Collectables Inc.	81806.55
1323	Vanauf	George	319	Mini Classics	78432.16
1323	Vanauf	George	486	Motor Mint Distributors Inc.	77726.59
1323	Vanauf	George	202	Canadian Gift Exchange Network	70122.19
1323	Vanauf	George	260	Royal Canadian Collectables, Ltd.	66812.00
1323	Vanauf	George	447	Gift Ideas Corp.	49967.78

S2

Question:

- Below, there is a query that creates a view. Run the query.
- Using the view, write a query that produces a table of the form `(productCode, productName)` for products that no customer in Asia has ordered.
- For this questions purposes, the Asian countries are:
  - Japan
  - Singapore

    - Philipines
    - Hong King
- You must not use a JOIN.

In [14]:
```
#
# Create the view
#
%sql create or replace view orders_all as \
    select * from orders join orderdetails using(orderNumber)
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
```

Out[14]:
```
[]
```

Answer:

In [15]:
```
%%sql
WITH
    asiaCustomers AS
    (
    SELECT
        customerNumber
    FROM customers
    WHERE
        country IN ('Japan', 'Hong Kong', 'Singapore', 'Philippines')
    ),
    asiaOrderedProducts AS
    (
    SELECT
        productCode
    FROM orders_all
    WHERE
        customerNumber IN
            (
            SELECT
                customerNumber
            FROM asiaCustomers
            )
    )
SELECT
    productCode,
    productName
FROM products
```

```
WHERE
    productCode NOT IN
        (
        SELECT
            productCode
        FROM asiaOrderedProducts
        )
```

```
 * mysql+pymysql://root:***@localhost
15 rows affected.
```

Out[15]:

productCode	productName
S10_1678	1969 Harley Davidson Ultimate Chopper
S10_4757	1972 Alfa Romeo GTA
S12_2823	2002 Suzuki XREO
S18_1342	1937 Lincoln Berline
S18_1367	1936 Mercedes-Benz 500K Special Roadster
S18_2795	1928 Mercedes-Benz SSK
S18_2870	1999 Indy 500 Monte Carlo SS
S18_3029	1999 Yamaha Speed Boat
S18_3233	1985 Toyota Supra
S18_3320	1917 Maxwell Touring Car
S18_3856	1941 Chevrolet Special Deluxe Cabriolet
S24_2022	1938 Cadillac V-16 Presidential Limousine
S24_2972	1982 Lamborghini Diablo
S24_4258	1936 Chrysler Airflow
S700_3505	The Titanic

S3

Question:

- Use the `customers` and `orders` for this query.

- Shipping days is the number of days between `orderDate` and `shippedDate.`
- Product a table of the form:
  - `customerNumber`
  - `customerName`
  - `noOfOrders` is the number of orders the customer placed.
  - `averageShippingDays`, which is the average shipping days.
  - `minimumShippingDays`, which is the minimum shipping days.
  - `maximumShippingDays`, which is the maximum shipping days.
- The table should only contain entries where:
  - `noOfOrders >= 3`
  - `averageShippingDays >= 5` or `maximumShippingDays >= 10.`

Answer:

In [30]:

```
%%sql
WITH groupedData AS
    (
    SELECT
        customerNumber,
        customerName,
        COUNT(customerNumber) as noOfOrders,
        AVG(DATEDIFF(shippedDate, orderDate)) as averageShippingDays,
        MIN(DATEDIFF(shippedDate, orderDate)) as minimumShippingDays,
        MAX(DATEDIFF(shippedDate, orderDate)) as maximumShippingDays
    FROM customers
    NATURAL JOIN
        orders
    GROUP BY
        customerNumber
    )
SELECT
    *
FROM groupedData
WHERE
    noOfOrders >= 3 AND # assuming that two conditions as stated in problem should be AND rather than OR
    (averageShippingDays >= 5 OR
    maximumShippingDays >= 10)
```

```
ORDER BY
    noOfOrders DESC;
```

```
 * mysql+pymysql://root:***@localhost
12 rows affected.
```

Out[30]:

customerNumber	customerName	noOfOrders	averageShippingDays	minimumShippingDays	maximumShippingDays
148	Dragon Souveniers, Ltd.	5	14.6000	1	65
161	Technics Stores Inc.	4	5.2500	4	6
276	Anna's Decorations, Ltd	4	5.0000	4	6
398	Tokyo Collectables, Ltd	4	5.5000	2	8
363	Online Diecast Creations Co.	3	5.0000	4	6
385	Cruz & Sons Co.	3	5.3333	5	6
198	Auto-Moto Classics Inc.	3	5.6667	5	6
205	Toys4GrownUps.com	3	5.3333	4	6
462	FunGiftIdeas.com	3	5.0000	3	6
448	Scandinavian Gift Ideas	3	5.5000	5	6
328	Tekni Collectables Inc.	3	5.0000	4	6
209	Mini Caravy	3	5.6667	5	6

Graph Database — Neo4j

- You will use your online/cloud Neo4j database for these problems.
- You must have loaded the Movie sample data.

N1

Question:

- The relationship `REVIEWED` connects a `Person` and `Movie`, and has the properties `rating` and `summary`.

- Write Python code using `py2neo` that produces the following table.

In [20]:
```
import pandas as pd
```

Answer:

In [25]:
```
def get_reviewed_relations():
    graph = Graph(neo4j_url, auth=(neo4j_user, neo4j_password))
    q = "MATCH (p:Person)-[r:REVIEWED]-(m:Movie) RETURN p.name as reviewer_name, " \
        "r.rating as rating, r.summary as movie_summary, " \
        "m.title as movie_title, m.released as movie_released ORDER BY CASE " \
        "WHEN reviewer_name = \"Angela Scope\" THEN 1 WHEN reviewer_name = \"Jessica Thompson\" THEN 2 " \
        "WHEN reviewer_name = \"James Thompson\" THEN 3 ELSE 4 END"
    res = graph.run(q)
    rels = []
    for r in res:
        rels.append(r)
    return rels
```

In [26]:
```
reviewed_relations = [dict(r) for r in get_reviewed_relations()]
reviewed_relations_df = pd.DataFrame(reviewed_relations)
reviewed_relations_df
```

Out[26]:

	reviewer_name	rating	movie_summary	movie_title	movie_released
0	Angela Scope	62	Pretty funny at times	The Replacements	2000
1	Jessica Thompson	92	You had me at Jerry	Jerry Maguire	2000
2	Jessica Thompson	65	Silly, but fun	The Replacements	2000
3	Jessica Thompson	45	Slapstick redeemed only by the Robin Williams ...	The Birdcage	1996
4	Jessica Thompson	85	Dark, but compelling	Unforgiven	1992
5	Jessica Thompson	95	An amazing journey	Cloud Atlas	2012
6	Jessica Thompson	68	A solid romp	The Da Vinci Code	2006
7	James Thompson	100	The coolest football movie ever	The Replacements	2000
8	James Thompson	65	Fun, but a little far fetched	The Da Vinci Code	2006

N2

Question:

- There are relationships `ACTED_IN` and `DIRECTED` between `Person` and `Movie`.
- Write Python code that produces the following table that shows people or both acted in and directed a movie.

In [31]:
```
def get_acted_directed_relations():
    graph = Graph(neo4j_url, auth=(neo4j_user, neo4j_password))
    q = "MATCH (p:Person)-[:ACTED_IN]-(m:Movie), (p)-[:DIRECTED]-(m) " \
        "RETURN p.name as Name, m.title as Title"
    res = graph.run(q)
    rels = []
    for r in res:
        rels.append(r)
    return rels
```

In [32]:
```
acted_directed_relations = [dict(r) for r in get_acted_directed_relations()]
acted_directed_relations_df = pd.DataFrame(acted_directed_relations)
acted_directed_relations_df
```

Out[32]:

	Name	Title
0	Tom Hanks	That Thing You Do
1	Clint Eastwood	Unforgiven
2	Danny DeVito	Hoffa

In [47]:
```
# Desired output
```

Out[47]:

	Name	Title
0	Tom Hanks	That Thing You Do
1	Clint Eastwood	Unforgiven
2	Danny DeVito	Hoffa

MongoDB

- Run the following code using your Atlas MongoDB.

In [37]:
```
import json

client = pymongo.MongoClient(
    mongodb_url
)

with open("./episodes.json") as e_file:
    episodes = json.load(e_file)

for e in episodes['episodes']:
    client['w4111_final']['episodes'].insert_one(e)
```

In [38]:
```
ratings_df = pd.read_csv("./got_title_ratings.csv")
ratings_info = ratings_df[['tconst', 'averageRating', 'numVotes']]
r_dict = ratings_info.to_dict("records")

for r in r_dict:
    client['w4111_final']['ratings'].insert_one(r)
```

Question:

Write Python code that uses an aggregation pipeline and operations to produce the following table.

In [39]:
```
# Requires the PyMongo package.
# https://api.mongodb.com/python/current
#
# Write the query/aggregation that produces result
result = client['w4111_final']['episodes'].aggregate([
            {
                '$project':{
                    "seasonNum": 1,
                    "episodeNum": 1,
                    "episodeLink": 1,
                    "episodeTitle": 1,
                    'episodeLink': {
                        '$substr': ["$episodeLink", 7, 9]
                    }
                }
            },
            {
```

```
                '$lookup' : {
                    'from': 'ratings',
                    'localField': 'episodeLink',
                    'foreignField': 'tconst',
                    'as': 'ratingInfo'
                }
            },
            {
                '$project': {
                    "seasonNum": 1,
                    "episodeNum": 1,
                    "episodeLink": 1,
                    "episodeTitle": 1,
                    'ratingInfo': {
                        '$arrayElemAt': [
                            '$ratingInfo', 0
                        ]
                    }
                }
            },
            {
                '$project': {
                    "_id": 0,
                    "seasonNum": 1,
                    "episodeNum": 1,
                    "episodeLink": 1,
                    "episodeTitle": 1,
                    'avgRating': '$ratingInfo.averageRating',
                    'numVotes': '$ratingInfo.numVotes'
                }
            }
])
```

In [40]:
```
info_df = pd.DataFrame(list(result))
info_df = info_df[['seasonNum', 'episodeNum', 'episodeLink', 'episodeTitle', 'avgRating', 'numVotes']]
info_df
```

Out[40]:

	seasonNum	episodeNum	episodeLink	episodeTitle	avgRating	numVotes
0	1	1	tt1480055	Winter Is Coming	8.9	48686
1	1	2	tt1668746	The Kingsroad	8.6	36837
2	1	3	tt1829962	Lord Snow	8.5	34863
3	1	4	tt1829963	Cripples, Bastards, and Broken Things	8.6	33136
4	1	5	tt1829964	The Wolf and the Lion	9.0	34436
...	...	...	...	...	...	...
68	8	2	tt6027908	A Knight of the Seven Kingdoms	7.9	130844
69	8	3	tt6027912	The Long Night	7.5	215995
70	8	4	tt6027914	The Last of the Starks	5.5	165067
71	8	5	tt6027916	The Bells	6.0	192449
72	8	6	tt6027920	The Iron Throne	4.0	248318

73 rows × 6 columns

In [62]:

```
# Desired output
info_df = pd.DataFrame(list(result))
info_df = info_df[['seasonNum', 'episodeNum', 'episodeLink', 'episodeTitle', 'avgRating', 'numVotes']]
info_df
```

Out[62]:

	seasonNum	episodeNum	episodeLink	episodeTitle	avgRating	numVotes
0	1	1	tt1480055	Winter Is Coming	8.9	48686
1	1	2	tt1668746	The Kingsroad	8.6	36837
2	1	3	tt1829962	Lord Snow	8.5	34863
3	1	4	tt1829963	Cripples, Bastards, and Broken Things	8.6	33136
4	1	5	tt1829964	The Wolf and the Lion	9.0	34436
...	...	...	...	...	...	...
68	8	2	tt6027908	A Knight of the Seven Kingdoms	7.9	130844
69	8	3	tt6027912	The Long Night	7.5	215995
70	8	4	tt6027914	The Last of the Starks	5.5	165067
71	8	5	tt6027916	The Bells	6.0	192449
72	8	6	tt6027920	The Iron Throne	4.0	248318

73 rows × 6 columns

Data Modeling and Schema Definition

- This is an exciting, interesting problem that involves:
  - Using Crow's Foot Notation
  - Relational approaches to implementing specialization, aggregation, quaternary relations, composite attributes and multi-valued attributes.
  - Foreign keys, check constraints and triggers.
- I did the answer and it took 3 hours to do all the work. My normal rule of thumb is that students require about 15 times as much time as I need to produce an answer.
- I giggled like the Riddler in Batman about how much fun we were going to have working on this question, and then the following happened.

- So, there will not be any data modeling question on the exam. Darn!

Module II Questions

- The questions require brief, written answers.

Q1

Question:

Briefly explain:

- Functional Dependency
- Lossy Decomposition
- Normalization

Answer:

- **Functional Dependency**: Relationship between two attributes in a table so that one attribute's value is uniquely dependent on the other attriute's value.
- **Lossy Decomposition**: When data is lost in the process of decomposing a table into smaller relations—for example if you decompose a table into two tables, one with the primary key and one without, data in one table has lost the primary key.
- **Normalization**: Codd's normal forms for minimizing redundancy and dependencies, ensuring consistency, and enforcing constraints—such as with the use of primary and foreign keys, and atomic domains.

Q2

Question:

Briefly explain:

- Serializability
- Conflict Serializability

- Deadlock
- Cascading Abort
- Two Phase Locking

Answer:

- **Serializability**: Property such that executing SQL statements would produce the same result as executing one statement after the other, allowing for data to be manipulated concurrently and maintain consistency.
- **Conflict Serializability**: Property such that swapping non-conflicting operations in a serial schedule would produce same result—each combination is equivalent, and the transaction can be executed in any order such that conflicting statements are not executed at the same time.
- **Deadlock**: When two transactions acquire a lock due to conflicting updates and resources, and one is locking the other—one statement is then waiting on the lock to be released to perform the transaction.
- **Cascading Abort**: When a transaction fails and causes a rollback which causes other dependent transactions to fail, which may occur due to deadlocks.
- **Two Phase Locking**: Mechanism to ensure concurrent updating and serializibility in two phases: (1) growing phase, where transactions can acquire locks but can't release, and (2) when transactions can release locks but not acquire them. Used to help prevent deadlocks.

Q3

Question:

Briefly explain:

- Logical block addressing, CHS addressing
- RAID-0, RAID-1, RAID-5
- Fixed length records, variable length records.

Answer:

- Logical block addressing, CHS addressing:
  - **LBA**: Used to idenfity location of data blocks on storage devices with a linear addressing scheme, where blocks are indexed by integers, incrementing from 0 and on. The disk controller and implementation translates this logical block adddress into the physical block address.

    - **CHS Addressing**: Identification of physical blocks of data on hard disks using cylinder positions on a specific track, sector, and head number where the data is located.
- RAID-0, RAID-1, RAID-5:
    - **RAID-0**: Consists of striping (splits data evenly across two or more disks) without any redundancy (mirroring or parity)—it doesn't use parity information, redundancy, or account for fault tolerance.
    - **RAID-1**: Consists of mirroring, without parity or striping—data is copied, or mirrored across two or more disks.
    - **RAID-5**: Consists of block-level striping with distributed parity using three or more disks.
- Fixed length records, variable length records.
    - **Fixed length records**: Where record length is fixed, or constant. Record i is stored from byte $n * (i - 1)$, where n is the size of each record.
    - **Variable length records**: When record length is not constant, and allows for storage of multiple record types in a file, such as varchars, and allows for repeating fields.

Q4

Question:

Briefly explain:

- Clustered Index
- Sparse Index
- Covering Index

Answer:

- **Clustered Index**: A type of index on which a table's ordering is dependent on—tables are reordered based on this index, where there can be only one clustered index.
- **Sparse Index**: A type of index that stores a portion of rows in a table for querying, optimizing storage for NULL values.
- **Covering Index**: A type of index that contains all the columns needed to satisfy a query, as well all the rows.

Q5

Question:

Briefly explain:

- Equivalent queries
- Hash Join
- Materialization, Pipelining

Answer:

- Equivalent queries: Two or more queries that produce the same results, used to optimize querying.
- Hash Join: Join operation that uses a hash table from two tables to merge them by comparing matching records from both hash tables.
- Materialization, Pipelining:
  - Materialization: Use of copy of query results as a temporary table or cache to allow faster querying and access.
  - Pipelining: Does not use temporary data to query, but merges operations into a pipeline, where each execution passes output into another operation sequentially—often quicker than materialization.

Source: https://www.javatpoint.com/pipelining-in-query-processing