COMS W 4111-02, H02, V02
Introduction to Databases
Spring 2019 Take Home Midterm

Exam Overview and Instructions

- Homework assignments and exams have point values. Final grade depends on total point value. The range of total possible points for a semester is 0 to 100 points (not including extra-credit).
    - The grade for this midterm is in the range 0 to 100.
    - **This midterm exam is worth 20 points** for your final semester points total. So, divide your score on this exam by 5 to determine point contribution.

- Submission:
    - The exam is due on 27-Oct-2019 at 11:59PM. **You may not use grace days.**
    - Submission:
        - Submission format a copy of this Jupyter Notebook with your solutions entered into the code cells or Markdown cells for each question. If you embed images or diagrams, you may need to use a zip file to include the images in the notebook.
        - You submit your homework on CourseWorks under "Midterm Examination" assignment.
        - No other formats are allowed.
        - If the notebook format is incorrect or the notebook is corrupted, the grade is 0.
        - Submissions after the due date and time are not allowed. Submissions not received on time receive a grade of 0.
    - Respect for the individual is paramount. We will accommodate special circumstances, but we must be notified and discuss **in advance.**

- Exam Rules:
    - Please read and review the Academic Integrity policy and guidelines (https://www.college.columbia.edu/academics/academicintegrity), including subsections and details. This material defines the rules for this exam regardless of your school.
    - No collaboration of any form is permitted. You may not share material of any form, including links to on-line information, **information from the preparatory recitation section,** suggestions or ideas, etc.
    - You **MAY** use material from office hours or recitations.
    - You MAY use any on-line information you find, but may not directly use code you find. You must cite any on-line sources in the comments Markdown cell for each questions.
    - You must privately send comments or questions to Professor Ferguson via email. If questions or comments demonstrate a need for clarification or correction, Prof. Ferguson will update this document and send an email notification.
    - You may not discuss the exam on Piazza. The CA or professor may post clarifying comment on Piazza.

- Completing the exam:
    - Environment:
        - You must install iPython-SQL (https://github.com/catherinedevlin/ipython-sql). Lecture notebooks have included iPython-SQL since lecture 1. There are installation instructions on the iPython-SQL web site and in lecture notes. **Note:** You must install into your Anaconda environment, and not other system Python environments.
        - You will need to use several databases/schemas for the homework. You will need to use lahman2019clean, which you had to install for HW2. You will also need classicmodels (http://www.mysqltutorial.org/mysql-sample-database.aspx). You need to install both databases, if you have not. You MUST use the schema/database names lahman2019clean and classmodels.
        - You must have a user ID dbuser with password dbuserdbuser and use in any connections you make completing questions.
        - Section 2 tests the setup of your environment. You **MUST USE** dbuser:dbuserdbuser@localhost/lahman2019.
        - Your database **MUST HAVE** a user `dbuser` with pasword `dbuserdbuser`
    - Each question starts with an explanation of the structure of the answer, e.g. prose, diagram, SQL, etc.
    - Questions requiring SQL or code have empty text cells where you enter your statements. Some also contain sample answers to help you determine if your answer is correct. You must show the execution of your SQL in your submission. You may use LIMIT 10 to reduce the size of results.

You may include diagrams in text/Markdown cells when answering a question. You can include a diagram using an HTML tag of the form:

```
<img src="./filename.file_extension">
```

The example below between the horizontal lines includes an image. You can click on the cell to see the Markdown code for including the image. **You will have to submit a zip file containing the notebook and you image if you include diagrams or images in your submission.**

- Double click in between lines to see the Markdown example for including a diagram.

Note:

- Not all questions with the same point value are equally hard.
- One five point question might be much easier than another five point question.
- A ten point question is not always exactly twice as hard as a five point question.
- Students sometimes complain about the discrepancy in difficulty and value.
- I am sensitive to the concern. I am happy to resolve this complaint by redoing the exam to make all five point questions as hard as the hardest five point question, and all ten point questions as hard as the hardest ten point question.
- Just let me know.

Environment Test

This section tests the environment. You must change the "userid:pw" to the correct user ID and password for your MySQL instance. Please change back to "userid:pw" before submitting your exam. Unless you have received an exception, you **MUST USE dbuser:dbuserdbuser.**

SQL Magic Plugin

In [3]:

```
%load_ext sql
%sql mysql+pymysql://userid:pw@localhost/lahman2019clean

%sql select * from people where playerid='willite01'
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
 * mysql+pymysql://dbuser:***@localhost/lahman2019clean
1 rows affected.
```

Out[3]:

playerID	birthYear	birthMonth	birthDay	birthCountry	birthState	birthCity	deathYear	deathM
willite01	1918	8	30	USA	CA	San Diego	2002	

In [4]:
```
%sql select * from classicmodels.customers where customerNumber=103
```

```
1 rows affected.
```

Out[4]:

customerNumber	customerName	contactLastName	contactFirstName	phone	addressLin
103	Atelier graphique	Schmitt	Carine	40.32.2555	54, ru Roya

Python Connection

In [7]:
```
import json
import pymysql
import logging

logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger()
logger.setLevel(logging.DEBUG)

midterm_conn = pymysql.connect(
    host="localhost",
    user="dbuser",
    password="dbuserdbuser",
    cursorclass=pymysql.cursors.DictCursor)
```

In [8]:
```
import logging


def run_q(sql, args=None, fetch=True, cur=None, conn=midterm_conn, com
mit=True):
    '''
    Helper function to run an SQL statement.

    This is a modification that better supports HW1. An RDBDataTable M
UST have a connection specified by
    the connection information. This means that this implementation of
run_q MUST NOT try to obtain
    a defailt connection.

    :param sql: SQL template with placeholders for parameters. Canno b
e NULL.
    :param args: Values to pass with statement. May be null.
    :param fetch: Execute a fetch and return data if TRUE.
    :param conn: The database connection to use. This cannot be NULL,
unless a cursor is passed.
        DO NOT PASS CURSORS for HW1.
    :param cur: The cursor to use. This is wizard stuff. Do not worry
about it for now.
        DO NOT PASS CURSORS for HW1.
    :param commit: This is wizard stuff. Do not worry about it.
    :return: A pair of the form (execute response, fetched data). Ther
e will only be fetched data if
        the fetch parameter is True. 'execute response' is the return
from the connection.execute, which
        is typically the number of rows effected.
    '''

    cursor_created = False
    connection_created = False

    try:

        if conn is None:
            raise ValueError("In this implementation, conn cannot be N
one.")

        if cur is None:
            cursor_created = True
            cur = conn.cursor()

        if args is not None:
            log_message = cur.mogrify(sql, args)
        else:
            log_message = sql

        logger.debug("Executing SQL = " + log_message)

        res = cur.execute(sql, args)

        if fetch:
            data = cur.fetchall()
        else:
            data = None

        # Do not ask.
        if commit == True:
            conn.commit()

    except Exception as e:
        raise(e)

    return (res, data)
```

In [16]:

```
q = "select playerID, nameLast, nameFirst from lahman2019clean.people
where nameLast=%s and birthCity=%s"
res,d = run_q(q, args=('Williams', 'San Diego'))

print("Data =\n{}".format(json.dumps(d, indent=2)))
```

```
DEBUG:root:Executing SQL = select playerID, nameLast, nameFirst from
lahman2019clean.people where nameLast='Williams' and birthCity='San
Diego'

Data =
[
  {
    "playerID": "willite01",
    "nameLast": "Williams",
    "nameFirst": "Ted"
  },
  {
    "playerID": "willitr01",
    "nameLast": "Williams",
    "nameFirst": "Trevor"
  }
]
```

Written Questions

Each question is worth 5 points.

Benefits of Database Management Systems

- Prior to database management systems (DBMS), user relied on application programs that directly access files to create, retrieve and update shared data.
- Give five benefits of using a database management system to provide data access for applications.
- One or two sentences is sufficient for each answer.
- Double click on the number to open the Markdown cell.

Answer

1. A DBMS vastly simplifies the process of accessing data in a file or group of files by providing a common interface.

1. Because a DBMS is not file or project specific, it is highly opimized for fast information access regardless of where it is in a properly formated file. On the other hand, it might not be worth optimizing to such levels an application program that is written for just one file or set of files.

1. A DBMS provides an interface that does not change, regardless of the data. An application program is brittle when facing files with different data than the one they were written for; therefore, it would require modifications to the program's code if the questions change.

1. A DBMS enforces certain semantics automatically. This means that it can greatly simplify managing a variety of logic and structures for applying integrity constraints, indexing, among other things.

1. If an application is designed and implemented with the proper encapsulation, i.e. using Data Access Objects, a change of DBMS for a project would require changes only to the code of the DAO that uses the DBMS interface.

Relational Concepts

Briefly explain *Cartesian product, equijoin, natural join,* and *theta join.*

Answer

Cartesian Product:

Equijoin:

Natural Join: When two tables have common attributes, the natural join will match the rows where the values of those common attributes are the same for both tables.

Theta Join:

Relational Algebra

Use the following tables when answering this question.

Name	Month
Don	September
Meghna	June
Aly	January
Ara	September
Kirit	May

BirthInfo

Month	Sign
January	Acquarius
September	Virgo
June	Gemini
July	Leo

AstrologicalInfo

Give the result of each of the relational algebra statements. You can provide your answer in text in the form:

column name, column name, ..., column name
value, value, ..., value
value, value, ..., value
value, value, ..., value

1. $\sigma_{Month="September"}(BirthInfo) \ \bowtie \ \pi_{Sign}(\sigma_{Month<"September"}))$

Answer

2. $\pi_{Name}(\sigma_{Month="December"}(AstrologicalSign))$

Answer

3. $BirthInfo \bowtie_{BirthInfo.Month=AstrologicalSign.Month} AtrologicalInfo$

Answer

4. $\pi_{Month}(BirthInfo) \wedge \pi_{Month}(AstrologicalInfo)$

Answer

5. Produce an SQL statement that is equivalent to $Student(\underline{UNI}, last_{n}ame, first_{n}ame, email)$

Answer

Relational Semantics

Provide a short (at most five sentences) answer to the following questions.

`1.` **Codd's Twelve Rules** define what it means for a DBMS to be relational. Briefly explain Rule 3, "Systematic treatment of null values."

Answer

When a value is not known or not applicable, it should be entered as null into the database. A relational database should be able to accept this null value regardless of the data type of the field it was entered into. The relational database then should be able to work with these null values in its operations without having to resort to treating unknowns in different ways for different data types. Having null defined in this way also allows for it to be used for integrity constraints purposes. Source: https://computing.derby.ac.uk/c/codds-twelve-rules/ (https://computing.derby.ac.uk/c/codds-twelve-rules/)

2. In a relational model, the domain for an attribute must be *atomic.* Briefly explain what this means. Given an example of a domain that is not atomic.

Answer

It means that any given value in a table must represent a single element. This does not mean that the value itself cannot be a complex construct; rather, it means that the value in an on itself must represent one and only piece of data that makes sense on its own. An example of a domain that is not atomic is one belonging to a fullName column. If we use my name as an example, fullName='Luigi Alessandro Pastore Pica', the domain is not atomic because we can decompose it into individually meaningful parts. In this case, properly atomic attribute domains would be nameFirst='Luigi', nameMiddle='Alessandro', nameLast='Pastore Pica' (this example is particularly useful because my last name is composed of two words, and still, the range of the attribute nameLast is atomic).

3. Briefly explain *super key, candidate key,* and *primary key.*

Answer

Super Key: it is an attribute or set of attributes that uniquely identify a row (tuple). In other words, a specific set of values belonging to these attributes can be mapped to one, and only one tuple.

Candidate Key: It is a super key for which none of its attribute subsets are super keys themselves. In other words, a combination of values belonging to a subset of the candidate key can map to more than one row, but a combination of values belonging to the whole candidate key can be mapped to only one row.

Primary Key: It is a candidate key that has been "arbitrarily" chosen to be the main key. This means that any candidate key is a potential primary key, but then only one of those is chosen to be the main (but not only) way of identifying a unique tuple.

4. Briefly (two or three sentences) explain the following concepts: *domain constraint, table integrity constraints, referential integrity constraints.*

Answer

Domain Constraint: Restricts the domain (type and limitation) of the values that can be contained in a column; e.g. a value cannot be null, a value has to be an integer, etc.

Table Integrity Constraints: Limit the actions that can be performed on a table so the data in it contained does not become inconsistent; e.g. prevent deleting a column belonging to the primary key.

Referential Integrity Constraints: Makes sure that the values appearing in one set of attributes for one table appear in another table related to it; e.g. if a foreign key is associated to a given table, then the constraint checks that the data is consistent on the referencing table and the referenced table.

5. What are referential integrity *cascading deletes* and *cascading updates?*

Answer

They are one of two options when an integrity constraint is violated by an action on a relation (the other option is simply aborting the action). When one of these actions (delete or update) would violate a referential integrity constraint, the cascading option would make changes to the referenced / referencing table in order to maintain the referential constraint. For instance, if all unique rows on Table_A must be mapped to at least row on Table_B, cascading a row deletion on Table_A would delete the corresponging row(s) on Table_B; in this way, the constraint remains valid on the reference between tableslp.

SQL Data Manipulation Language Questions

Batter Performance for Red Sox in 1960 (5 points)

- This query requires the following columns from `Lahman2019clean:`
  - `people.playerid, people.nameLast, people.bats`
  - `batting.playerid, batting.ab, batting.h, batting.bb, batting.hr, batting.teamid, batting.yearid, batting.2b, batting.3b, batting.HR`
- The formula for on-base percentage is (H + BB)/(H + AB). We will denote on-base percentage as `OBP.`
- Batting average is H/AB. We will denote this as AVG.
- Slugging Percentage:
  - In the `Batting` table, `H` is total hits.
  - The table lists three types of hits 2B`` is doubles, 3B is triples and HR is homeruns. There is a fourth type of hit, _singles_ that contributes to total hits but is not in the table. We will call this 1B```
  - The formulate for slugging percentage, which we will denote as SLG, is

$$\frac{1B + 2 * 2B + 3 * 3B + 4 * HR}{AB}$$

- The following table summarizes batting performance for BOS in 1960 for the top ten hitters, ordered by SLG. Write and execute the SQL to produce the table.

Your query and execution

```
In [ ]:
```

My Answer

In [21]:

```
10 rows affected.
```

Out[21]:

playerid	nameLast	bats	H	AB	1B	2B	3B	HR	RBI	AVG	OBP	SLG
willite01	Williams	L	98	310	54.0	15	0	29	72	0.316	0.449	0.645
pagliji01	Pagliaroni	R	19	62	10.0	5	2	2	9	0.306	0.427	0.548
geigega01	Geiger	L	74	245	49.0	13	3	9	33	0.302	0.362	0.49
wertzvi01	Wertz	L	125	443	84.0	22	0	19	103	0.282	0.338	0.46
thomsbo01	Thomson	R	30	114	21.0	3	1	5	20	0.263	0.328	0.439
nixonru01	Nixon	L	81	272	56.0	17	3	5	33	0.298	0.33	0.438
fornimi01	Fornieles	R	6	15	6.0	0	0	0	1	0.4	0.4	0.4
malzofr01	Malzone	R	161	595	115.0	30	2	14	79	0.271	0.312	0.398
runnepe01	Runnels	L	169	528	136.0	29	2	2	35	0.32	0.401	0.394
tasbywi01	Tasby	R	108	385	83.0	17	1	7	37	0.281	0.365	0.384

Set Membership (5 points)

- This query involves the lahman2019clean tables `halloffame`, `people`, `appearances`, `pitching`, `managers`.
- Return the `playerID`, `nameLast`, `nameFirst` for every person that is in all of the tables.

Your query and execution

In [ ]:

My Answer

In [22]:

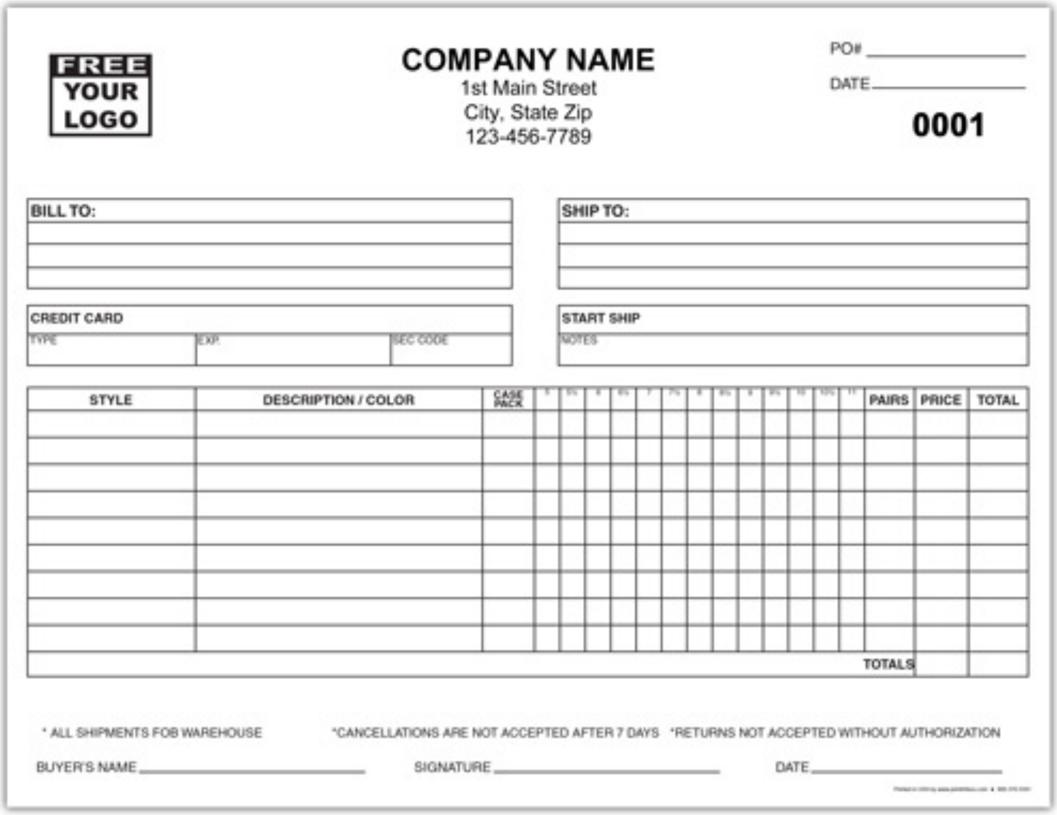
```
10 rows affected.
```

Out[22]:

playerid	nameFirst	nameLast
zimmech01	Chief	Zimmer
yorkru01	Rudy	York
wilsoji01	Jimmie	Wilson
willsma01	Maury	Wills
willima04	Matt	Williams
willidi02	Dick	Williams
westrwe01	Wes	Westrum
weisswa01	Walt	Weiss
wathajo01	John	Wathan
walkeha01	Harry	Walker

Complex Insert (10 points)

- Use `classicmodels` for this question.
- An order form typically looks something like:



FREE YOUR LOGO

COMPANY NAME
1st Main Street
City, State Zip
123-456-7789

PO# ____________
DATE ____________

0001

BILL TO:

SHIP TO:

CREDIT CARD
TYPE EXP. SEC CODE

START SHIP
NOTES

STYLE	DESCRIPTION / COLOR	CASE PACK	5	5½	6	6½	7	7½	8	8½	9	9½	10	10½	11	PAIRS	PRICE	TOTAL
																TOTALS		

* ALL SHIPMENTS FOB WAREHOUSE *CANCELLATIONS ARE NOT ACCEPTED AFTER 7 DAYS *RETURNS NOT ACCEPTED WITHOUT AUTHORIZATION

BUYER'S NAME ____________ SIGNATURE ____________ DATE ____________

Example Order Form

- For `classicmodels` the application user interface would `POST` of the form.

```
{
            "orderNumber": 10123,
            "orderDate": "2003-05-20",
            "requiredDate": "2003-05-29",
            "shippedDate": "2003-05-22",
            "status": "Shipped",
            "comments": null,
            "customerNumber": 103,
            "orderdetails": [
                {
                    "orderNumber": 10123,
                    "productCode": "S18_1589",
                    "quantityOrdered": 26,
                    "priceEach": "120.71",
                    "orderLineNumber": 2
                },
                {
                    "orderNumber": 10123,
                    "productCode": "S18_2870",
                    "quantityOrdered": 46,
                    "priceEach": "114.84",
                    "orderLineNumber": 3
                },
                {
                    "orderNumber": 10123,
                    "productCode": "S18_3685",
                    "quantityOrdered": 34,
                    "priceEach": "117.26",
                    "orderLineNumber": 4
                },
                {
                    "orderNumber": 10123,
                    "productCode": "S24_1628",
                    "quantityOrdered": 50,
                    "priceEach": "43.27",
                    "orderLineNumber": 1
                }
            ]
        }
```

- This data structure maps to two tables in `classicmodels: orders` and `ordersdetails`

- Complete the implementation of the Python function below that takes a data structure (dict) of the form above and inserts that data into `classicmodels`.

Answer

In [23]:

```python
def create_order(order_info):
    """
    Creates (Inserts) the data associated with an order. The order information goes into orders table and each
    and line item/order detail item goes into the ordersdetails table.
    :param order_info: A dictionary. There are top-level elements for the order. There is an orderdetails element
        that is a list of dictionary for the orderdetails elements.
    :param cnx: The database connection to use.
    :return: A tuple of the form (order_insert_count, orderdetals_insert_count), where the values are the number
        of rows inserted into each table.
    """

    # Your code goes here.
    pass
```

Complex Query/View — Player Performance Statistics by Year (10 points)

- Use the `lahman2019clean` database/schema.
- Create performance summary views. Create five views:
  - `batting_summary:` yearID, teamID, AB, H, HR, RBI
  - `appearances summary:` yearID, teamID, G_all, GS
  - `pitching summary:` yearID, teamID, W, L, IPouts
  - `fielding_summary` yearID, teamID, PO, A, E, POS
  - `annual_summary,` which combines the views above.
  - `career_summary,` which contains the totals/summaries for the entire career.
- **NOTE:** You will need to do aggregation on some of the views to get annual values.
- **Note:** Your query must produce the correct results for any `playerID`.

Answer

- `batting_summary`

```
Put your create view statement here.
```

In [10]:

```
%sql select * from batting_summary where playerID='willite01'
```

```
19 rows affected.
```

Out[10]:

playerid	teamid	yearid	ab	h	hr	rbi
willite01	BOS	1939	565.0	185.0	31.0	145.0
willite01	BOS	1940	561.0	193.0	23.0	113.0
willite01	BOS	1941	456.0	185.0	37.0	120.0
willite01	BOS	1942	522.0	186.0	36.0	137.0
willite01	BOS	1946	514.0	176.0	38.0	123.0
willite01	BOS	1947	528.0	181.0	32.0	114.0
willite01	BOS	1948	509.0	188.0	25.0	127.0
willite01	BOS	1949	566.0	194.0	43.0	159.0
willite01	BOS	1950	334.0	106.0	28.0	97.0
willite01	BOS	1951	531.0	169.0	30.0	126.0
willite01	BOS	1952	10.0	4.0	1.0	3.0
willite01	BOS	1953	91.0	37.0	13.0	34.0
willite01	BOS	1954	386.0	133.0	29.0	89.0
willite01	BOS	1955	320.0	114.0	28.0	83.0
willite01	BOS	1956	400.0	138.0	24.0	82.0
willite01	BOS	1957	420.0	163.0	38.0	87.0
willite01	BOS	1958	411.0	135.0	26.0	85.0
willite01	BOS	1959	272.0	69.0	10.0	43.0
willite01	BOS	1960	310.0	98.0	29.0	72.0

- Pitching summary

Put create view statement here.

In [11]:

```
%sql select * from pitching_summary where playerid='willite01';
```

```
1 rows affected.
```

Out[11]:

playerID	teamID	yearID	w	l	g_p	IPouts
willite01	BOS	1940	0.0	0.0	1.0	6.0

- `fielding summary`

```
Put create view statement here.
```

```
In [12]: %sql select * from fielding_summary where playerid='willite01'
```

```
19 rows affected.
```

Out[12]:

playerid	teamid	yearid	po	a	e	group_concat(pos)
willite01	BOS	1939	318.0	11.0	19.0	OF
willite01	BOS	1940	302.0	15.0	13.0	OF,P
willite01	BOS	1941	262.0	11.0	11.0	OF
willite01	BOS	1942	312.0	15.0	4.0	OF
willite01	BOS	1946	325.0	7.0	10.0	OF
willite01	BOS	1947	347.0	10.0	9.0	OF
willite01	BOS	1948	289.0	9.0	5.0	OF
willite01	BOS	1949	337.0	12.0	6.0	OF
willite01	BOS	1950	165.0	7.0	8.0	OF
willite01	BOS	1951	315.0	12.0	4.0	OF
willite01	BOS	1952	4.0	0.0	0.0	OF
willite01	BOS	1953	31.0	1.0	1.0	OF
willite01	BOS	1954	213.0	5.0	4.0	OF
willite01	BOS	1955	170.0	5.0	2.0	OF
willite01	BOS	1956	174.0	7.0	5.0	OF
willite01	BOS	1957	215.0	2.0	1.0	OF
willite01	BOS	1958	154.0	3.0	7.0	OF
willite01	BOS	1959	94.0	4.0	3.0	OF
willite01	BOS	1960	131.0	6.0	1.0	OF

- `appearances_summary`

```
Put create view statement here.
```

In [13]:

```
%sql select * from appearances_summary where playerid = 'willite01'
```

```
19 rows affected.
```

Out[13]:

playerid	teamid	yearid	G_all	GS
willite01	BOS	1939	149	149
willite01	BOS	1940	144	143
willite01	BOS	1941	143	133
willite01	BOS	1942	150	150
willite01	BOS	1946	150	150
willite01	BOS	1947	156	156
willite01	BOS	1948	137	134
willite01	BOS	1949	155	155
willite01	BOS	1950	89	86
willite01	BOS	1951	148	147
willite01	BOS	1952	6	2
willite01	BOS	1953	37	26
willite01	BOS	1954	117	113
willite01	BOS	1955	98	93
willite01	BOS	1956	136	110
willite01	BOS	1957	132	125
willite01	BOS	1958	129	114
willite01	BOS	1959	103	75
willite01	BOS	1960	113	87

- `annual_summary`

```
Put create view satement here.
```

In [14]:
```
%sql select * from annual_summary where playerid='willite01'
```

```
19 rows affected.
```

Out[14]:

playerid	teamid	yearid	G_all	GS	ab	h	hr	rbi	w	l	g_p	IPouts	p
willite01	BOS	1939	149	149	565.0	185.0	31.0	145.0	None	None	None	None	318
willite01	BOS	1940	144	143	561.0	193.0	23.0	113.0	0.0	0.0	1.0	6.0	302
willite01	BOS	1941	143	133	456.0	185.0	37.0	120.0	None	None	None	None	262
willite01	BOS	1942	150	150	522.0	186.0	36.0	137.0	None	None	None	None	312
willite01	BOS	1946	150	150	514.0	176.0	38.0	123.0	None	None	None	None	325
willite01	BOS	1947	156	156	528.0	181.0	32.0	114.0	None	None	None	None	347
willite01	BOS	1948	137	134	509.0	188.0	25.0	127.0	None	None	None	None	289
willite01	BOS	1949	155	155	566.0	194.0	43.0	159.0	None	None	None	None	337
willite01	BOS	1950	89	86	334.0	106.0	28.0	97.0	None	None	None	None	165
willite01	BOS	1951	148	147	531.0	169.0	30.0	126.0	None	None	None	None	315
willite01	BOS	1952	6	2	10.0	4.0	1.0	3.0	None	None	None	None	4
willite01	BOS	1953	37	26	91.0	37.0	13.0	34.0	None	None	None	None	31
willite01	BOS	1954	117	113	386.0	133.0	29.0	89.0	None	None	None	None	213
willite01	BOS	1955	98	93	320.0	114.0	28.0	83.0	None	None	None	None	170
willite01	BOS	1956	136	110	400.0	138.0	24.0	82.0	None	None	None	None	174
willite01	BOS	1957	132	125	420.0	163.0	38.0	87.0	None	None	None	None	215
willite01	BOS	1958	129	114	411.0	135.0	26.0	85.0	None	None	None	None	154
willite01	BOS	1959	103	75	272.0	69.0	10.0	43.0	None	None	None	None	94
willite01	BOS	1960	113	87	310.0	98.0	29.0	72.0	None	None	None	None	131

- `career_summary`

`Put create view statement here.`

In [16]:
```
%sql select * from career_summary limit 10;
```

```
10 rows affected.
```

Out[16]:

playerid	g_all	gs	ab	h	hr	rbi	w	l	IPouts	po	a	e	pos
aardsda01	331	0	4	0	0	0	16	18	1011	11	29	3	
aaronha01	3298	3173	12364	3771	755	2297	None	None	None	7436	429	144	1B,2B,
aaronto01	437	206	944	216	13	94	None	None	None	1317	113	22	1B,2B,
aasedo01	448	91	5	0	0	0	66	60	3328	67	135	13	
abadan01	15	4	21	2	0	0	None	None	None	37	1	1	
abadfe01	363	6	9	1	0	0	8	27	953	7	37	2	
abadijo01	12	0	49	11	0	5	None	None	None	129	3	13	
abbated01	857	357	3044	772	11	324	None	None	None	1873	2368	315	2B,3B
abbeybe01	79	0	225	38	0	17	22	40	1704	17	134	22	
abbeych01	452	0	1756	493	19	280	0	0	6	920	92	100	

Update Statement (5 points)

- Make copies of the `orders` table in `classicmodels.`
- The following statements will accomplish that.

In [60]:
```
%%sql
use classicmodels;
drop table if exists orders_copy;
create table orders_copy as select * from orders;
```

```
0 rows affected.
0 rows affected.
326 rows affected.
```

Out[60]:
```
[]
```

- You can test of your copy worked by producing the same results as the following query.

In [59]:

```
%sql select * from orders_copy join orderdetails using(orderNumber) where orderNumber=10100;
```

```
4 rows affected.
```

Out[59]:

orderNumber	orderDate	requiredDate	shippedDate	status	comments	customerNumber	p
10100	2003-01-06	2003-01-13	2003-01-10	Shipped	None	363	
10100	2003-01-06	2003-01-13	2003-01-10	Shipped	None	363	
10100	2003-01-06	2003-01-13	2003-01-10	Shipped	None	363	
10100	2003-01-06	2003-01-13	2003-01-10	Shipped	None	363	

- Write a single `UPDATE` statement that sets the status of all orders for customers to 'EMBARGOED' if:
  - The customer's address is in Australia And
  - The order's status is not SHIPPED or CANCELLED.

- Before the update, run the following query. You should get results that match the example.

In [61]:

```
%%sql
select
        customers.customerNumber, customers.country, orders_copy.orderNumber, orders_copy.status from
    customers join orders_copy
    using (customerNumber)
    where country = 'Australia'
order by status;
```

```
19 rows affected.
```

Out[61]:

customerNumber	country	orderNumber	status
471	Australia	10415	Disputed
282	Australia	10420	In Process
114	Australia	10120	Shipped
114	Australia	10125	Shipped
282	Australia	10139	Shipped
276	Australia	10148	Shipped
333	Australia	10152	Shipped
276	Australia	10169	Shipped
333	Australia	10174	Shipped
471	Australia	10193	Shipped
114	Australia	10223	Shipped
471	Australia	10265	Shipped
282	Australia	10270	Shipped
114	Australia	10342	Shipped
114	Australia	10347	Shipped
282	Australia	10361	Shipped
276	Australia	10370	Shipped
333	Australia	10374	Shipped
276	Australia	10391	Shipped

Answer Your update statement

In [ ]:

```
%%sql
```

- After running your update, run the following query to produce the same output as the example.

In [62]:

```
%%sql
select
        customers.customerNumber, customers.country, orders_copy.orderNumber, orders_copy.status from
    customers join orders_copy
    using (customerNumber)
    where country = 'Australia'
order by status;
```

```
19 rows affected.
```

Out[62]:

customerNumber	country	orderNumber	status
471	Australia	10415	EMBARGOED
282	Australia	10420	EMBARGOED
114	Australia	10120	Shipped
114	Australia	10125	Shipped
282	Australia	10139	Shipped
276	Australia	10148	Shipped
333	Australia	10152	Shipped
276	Australia	10169	Shipped
333	Australia	10174	Shipped
471	Australia	10193	Shipped
114	Australia	10223	Shipped
471	Australia	10265	Shipped
282	Australia	10270	Shipped
114	Australia	10342	Shipped
114	Australia	10347	Shipped
282	Australia	10361	Shipped
276	Australia	10370	Shipped
333	Australia	10374	Shipped
276	Australia	10391	Shipped

Data Modeling, Cleanup and Implementation

Classicmodels Orders (5 points)

- There is a glaringly obvious design problem that could compromise data integrity in the table `classicmodels.orders`.

- The current schema is:

  ```
  CREATE TABLE `orders` (
  `orderNumber` int(11) NOT NULL,
  `orderDate` date NOT NULL,
  `requiredDate` date NOT NULL,
  `shippedDate` date DEFAULT NULL,
  `status` varchar(15) NOT NULL,
  `comments` text,
  `customerNumber` int(11) NOT NULL,
  PRIMARY KEY (`orderNumber`),
  KEY `customerNumber` (`customerNumber`),
  CONSTRAINT `orders_ibfk_1` FOREIGN KEY (`customerNumber`) REFERENCES `
  customers` (`customerNumber`)
  ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
  ```

- Alter the schema to correct the issue, and test your correction.

Answer

In [ ]:

In [ ]:

Data Cleanup (10 Points)

- There are international standards for two letter country codes, e.g. ISO 3166-1.

- datahub.io has down-loadable versions (https://datahub.io/core/country-list) of the information.

- The first part of answering this question is downloading the country code, country name information, and loading into a table in classicmodels.

- After a successful load, a sample query produces.

```
In [66]: %sql SELECT * FROM classicmodels.countrycodes limit 10;
```

```
10 rows affected.
```

Out[66]:

Name	Code
Afghanistan	AF
Åland Islands	AX
Albania	AL
Algeria	DZ
American Samoa	AS
Andorra	AD
Angola	AO
Anguilla	AI
Antarctica	AQ
Antigua and Barbuda	AG

- Allowing people to enter country names as free form text is an extremely bad idea. People will enter things like 'USA,' 'US', 'U.S. of A.', 'United States,' ...
- We are going to modify a copy of `classicmodels.customers` to have better integrity.
- The first step is to create a copy of `classicmodels.customers`.

```
In [67]: %sql create table classicmodels.customers_clean as select * from classicmodels.customers;
```

```
122 rows affected.
```

```
Out[67]: []
```

In [72]:

```
%sql select customerNumber, customerName, country from classicmodels.c
ustomers_clean limit 10;
```

```
10 rows affected.
```

Out[72]:

customerNumber	customerName	country
103	Atelier graphique	France
112	Signal Gift Stores	USA
114	Australian Collectors, Co.	Australia
119	La Rochelle Gifts	France
121	Baane Mini Imports	Norway
124	Mini Gifts Distributors Ltd.	USA
125	Havel & Zbyszek Co	Poland
128	Blauer See Auto, Co.	Germany
129	Mini Wheels Co.	USA
131	Land of Toys Inc.	USA

- You must produce a table that looks like the following, and implements referential integrity.

In [73]:

```
%sql select * from customers_clean limit 10;
```

```
10 rows affected.
```

Out[73]:

customerNumber	customerName	contactLastName	contactFirstName	phone	addressLir
103	Atelier graphique	Schmitt	Carine	40.32.2555	54, Roy
112	Signal Gift Stores	King	Jean	7025551838	8489 Strc
114	Australian Collectors, Co.	Ferguson	Peter	03 9520 4555	636 St Ki Rc
119	La Rochelle Gifts	Labrune	Janine	40.67.8555	67, rue c Cinqua Otag
121	Baane Mini Imports	Bergulfsen	Jonas	07-98 9555	Erl Skakkes g
124	Mini Gifts Distributors Ltd.	Nelson	Susan	4155551450	5677 Strc
125	Havel & Zbyszek Co	Piestrzeniewicz	Zbyszek	(26) 642-7555	ul. Filtrowa
128	Blauer See Auto, Co.	Keitel	Roland	+49 69 66 90 2555	Lyonerstr.
129	Mini Wheels Co.	Murphy	Julie	6505555787	5557 Nc Pend Str
131	Land of Toys Inc.	Lee	Kwai	2125557818	897 Lc Airp Aver

In [75]:

```
try:
    %sql update customers_clean set country_code = 'XX' where customer
Number=103
    print("Getting here is bad.")
except Exception as e:
    print("This is OK, e = ", e)
```

```
This is OK, e =  (pymysql.err.IntegrityError) (1452, 'Cannot add or
update a child row: a foreign key constraint fails (`classicmodels`.
`customers_clean`, CONSTRAINT `cc` FOREIGN KEY (`country_code`) REFE
RENCES `country_codes` (`code`))')
[SQL: update customers_clean set country_code = 'XX' where customerN
umber=103]
(Background on this error at: http://sqlalche.me/e/gkpj)
```

This one is really unpleasant!

Answer

- This one is really unpleasant.

- The character sets might be a problem.

E-R Diagrams (5 points)

- **Note:** Please use Crow's Foot notation for this diagram.

- The model has the following entity types:
  - Student(uni, last_name, first_name)
  - Course(course_id, course_name)
  - Section(section_number, semester, year, course_id)

- Draw a logical ER diagram representing the data model. You do not have to worry about column types.

- The model MUST represent student enrollments.

- You may need to create an additional table.

Answer

Inheritance and Stored Procedures (10 points)

- The two following table definitions are a simple model for people at a university.

```
CREATE TABLE `student` (
  `student` varchar(12) NOT NULL,
  `last_name` varchar(64) NOT NULL,
  `first_name` varchar(64) NOT NULL,
  `graduation_year` year(4) NOT NULL,
  PRIMARY KEY (`student`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;


CREATE TABLE `faculty` (
  `uni` varchar(12) NOT NULL,
  `last_name` varchar(64) NOT NULL,
  `first_name` varchar(64) NOT NULL,
  `title` enum('Professor','Assistant Professor','Associate Professor','Ad
junct Professor') NOT NULL,
  PRIMARY KEY (`uni`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

- Implement a view `People` that supports `SELECT` for the following columns:
  - UNI
  - last_name
  - first_name
  - Type is 'S' if the person is a student and 'F' if the person is a faculty.
  - 'NA' for graduation year if the person is not a student.
  - 'NA' for title if the person is not a faculty.

- Write a stored procedure that:
  - Inserts the data in the proper table based on the type.
  - Generates a unique UNI for a newly inserted person.

- You do not need to worry about error checking parameters, types, etc.

Answer

Putting Some Pieces Together (5 points)

- The following is the current definition for `lahman2019.salaries.`

  ```
  CREATE TABLE `salaries` (
    `yearID` text,
    `teamID` text,
    `lgID` text,
    `playerID` text,
    `salary` text
  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
  ```

- Create a copy of the data into `salaries_clean.`
- Transform the definition to improve integrity, making whatever changes you think necessary. The changes will require modifying column types, check constraints/triggers to ensure values are valid, and foreign key constraints.

Answer

Graph Data — Game of Thrones

- The GitHub repository https://github.com/melaniewalsh/sample-social-network-datasets/tree/master/sample-datasets/game-of-thrones (https://github.com/melaniewalsh/sample-social-network-datasets/tree/master/sample-datasets/game-of-thrones) contains data for a graph of relationships between characters in *Game of Thrones.*
  - The file *got-nodes.csv* contains simple information about characters.
  - The file *got-edges.csv* contains information about relationships between characters.
- The README (https://github.com/melaniewalsh/sample-social-network-datasets/blob/master/sample-datasets/game-of-thrones/README.md) explains the meaning of the files and fields.
- Load the data:
  - Create a new database *W4111Midterm* in your MySQL instance.
  - Use the Table Data import tool to load the CSV files into tables named:
    - _got*nodes*
    - _got*edges*
- After loading, your sample data should look like the examples below.

In [78]:

```
%sql select * from W4111Midterm.got_nodes limit 10;
```

```
10 rows affected.
```

Out[78]:

Id	Label
Aegon	Aegon
Aemon	Aemon
Aerys	Aerys
Alliser	Alliser
Amory	Amory
Anguy	Anguy
Arya	Arya
Balon	Balon
Barristan	Barristan
Belwas	Belwas

In [79]:

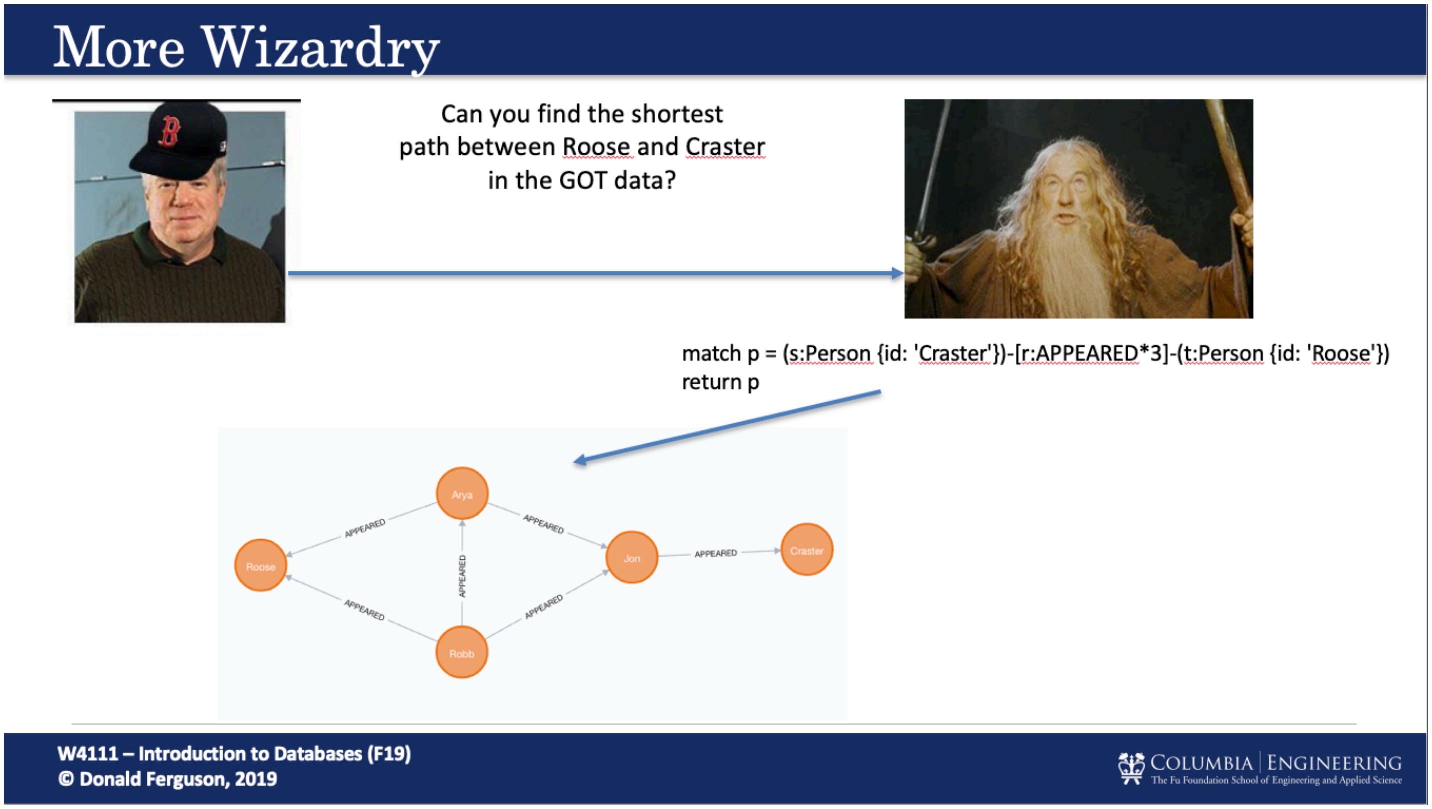
```
%sql select * from W4111Midterm.got_edges limit 10;
```

```
10 rows affected.
```

Out[79]:

Source	Target	Weight
Aemon	Grenn	5
Aemon	Samwell	31
Aerys	Jaime	18
Aerys	Robert	6
Aerys	Tyrion	5
Aerys	Tywin	8
Alliser	Mance	5
Amory	Oberyn	5
Arya	Anguy	11
Arya	Beric	23

- Wanted to know the shortest path in the data between two obscure characters: Roose, Craster.
- So, I asked my wizard friend.



More Wizardry

Can you find the shortest path between Roose and Craster in the GOT data?

```
match p = (s:Person {id: 'Craster'})-[r:APPEARED*3]-(t:Person {id: 'Roose'})
return p
```

W4111 – Introduction to Databases (F19)
© Donald Ferguson, 2019

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Advanced Magic

- I did not understand the spell. I have been teaching SQL.
- But, I know giant class of wizards comfortable with SQL magic. So, I decided to ask them to show me the spell.

- So, the exam question is, "Show me an SQL spell that returns the information."
- There is a spell that is a single SQL statement, but the aliasing will drive you nuts.
- You may create tables that compute partial results.

Answer

In [ ]:

In [81]:

```
4 rows affected.
```

Out[81]:

one_source	one_target	two_source	two_target	one_source_1	one_target_1
Craster	Jon	Jon	Arya	Arya	Roose
Roose	Arya	Arya	Jon	Jon	Craster
Roose	Robb	Robb	Jon	Jon	Craster
Craster	Jon	Jon	Robb	Robb	Roose