In [3]:
```
%load_ext sql
%sql mysql+pymysql://dbuser:dbuserdbuser@localhost/lahman2019clean

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

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
1 rows affected.
```

Out[3]:

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

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

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
1 rows affected.
```

Out[4]:

customerNumber	customerName	contactLastName	contactFirstName	phone	addressLine1
103	Atelier graphique	Schmitt	Carine	40.32.2555	54, rue Royale

Python Connection

In [29]:
```
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 [30]:
```
import logging


def run_q(sql, args=None, fetch=True, cur=None, conn=midterm_conn, commi
    '''
    Helper function to run an SQL statement.
```

```
    This is a modification that better supports HW1. An RDBDataTable MUS
    the connection information. This means that this implementation of r
    a defailt connection.

    :param sql: SQL template with placeholders for parameters. Canno be
    :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, un
        DO NOT PASS CURSORS for HW1.
    :param cur: The cursor to use. This is wizard stuff. Do not worry ab
        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). There
        the fetch parameter is True. 'execute response' is the return fr
        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 Non

        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 [31]:
```
q = "select playerID, nameLast, nameFirst from lahman2019clean.people wh
print(q)
res,d = run_q(q, args=('Williams', 'San Diego'))

print("Data = ", json.dumps(d, indent=2))
```

```
DEBUG:root:Executing SQL = select playerID, nameLast, nameFirst from l
ahman2019clean.people where nameLast='Williams' and birthCity='San Die
go'

select playerID, nameLast, nameFirst from lahman2019clean.people where
nameLast=%s and birthCity=%s
Data =  [
  {
    "playerID": "willite01",
    "nameLast": "Williams",
    "nameFirst": "Ted"
  },
  {
    "playerID": "willitr01",
    "nameLast": "Williams",
    "nameFirst": "Trevor"
  }
]
```

Your query and execution

In [152]:
```
%%sql
use lahman2019clean;

SELECT distinct people.playerID, people.nameLast, people.bats, batting.H
(batting.H - batting.2B - batting.3B - batting.HR) AS '1B', batting.2B,
round(if(batting.AB=0, null, batting.H/batting.AB),3) AS 'AVG',
round(if (batting.BB + batting.AB =0, null,(batting.H + batting.BB)/ (ba
round(if (batting.AB =0, null, ((batting.H - batting.2B - batting.3B - b
FROM PEOPLE JOIN BATTING using(playerID) WHERE batting.teamID='BOS' and
ORDER BY SLG DESC LIMIT 10;
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
0 rows affected.
10 rows affected.
```

Out[152]:

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

My Answer

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 [153]:
```
%%sql select playerID, nameFirst, nameLast from
People join (SELECT distinct playerID from halloffame JOIN
             (SELECT DISTINCT playerID from appearances JOIN (SELECT pla
            USING(playerID)) MPi USING(playerID)) AMPi USING(playerID))
            using(playerID) order by nameLast DESC LIMIT 10
```

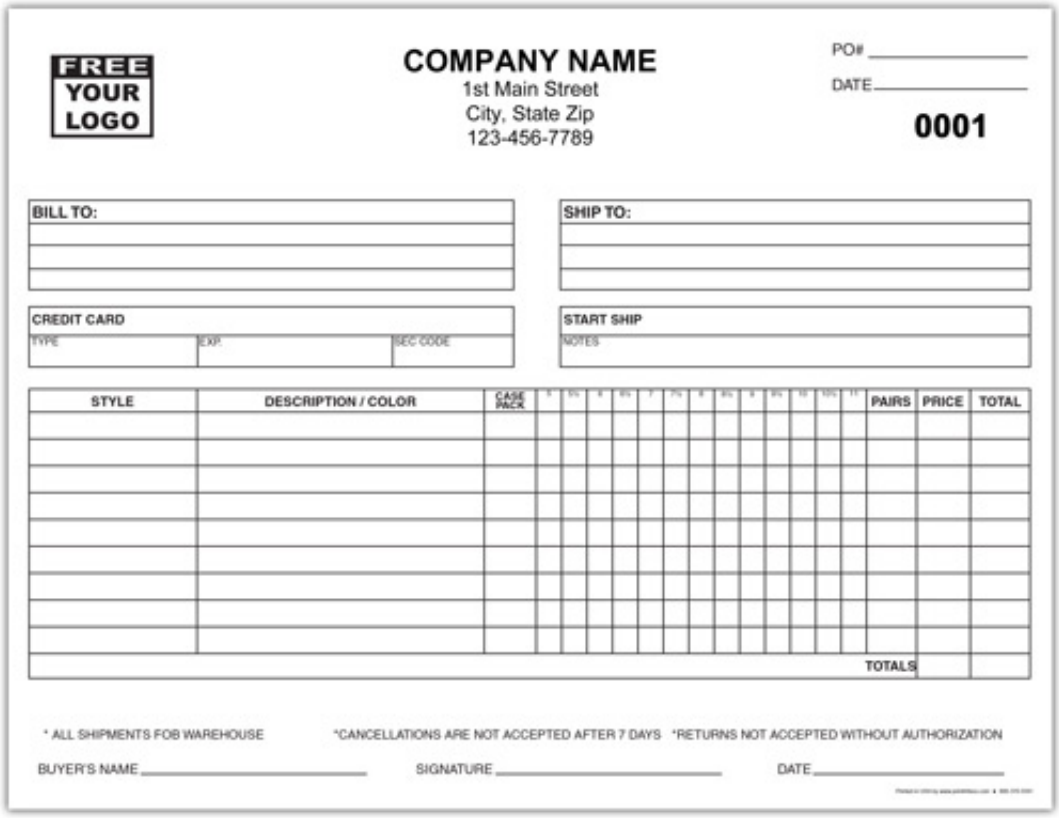
```
 * mysql+pymysql://root:***@localhost/lahman2019clean
10 rows affected.
```

Out[153]:

playerID	nameFirst	nameLast
youngcy01	Cy	Young
wrighha01	Harry	Wright
wrighge01	George	Wright
willite01	Ted	Williams
whitewi01	Will	White
whitede01	Deacon	White
wardjo01	John	Ward
waltebu01	Bucky	Walters
walshed01	Ed	Walsh
wallabo01	Bobby	Wallace

Complex Insert (10 points)

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



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 [32]:
```
import pandas as pd
def create_order(order_info):

    ordertable="classicmodels.orders"
    orderdetailstable="classicmodels.orderdetails"

    def find_by_template(template, field_list=None, limit=None, offset=N
        """

        :param template: A dictionary of the form { "field1" : value1, "
        :param field_list: A list of request fields of the form, ['field
        :param limit: Do not worry about this for now.
        :param offset: Do not worry about this for now.
        :param order_by: Do not worry about this for now.
        :return: A list containing dictionaries. A dictionary is in the
            that matches the template. The dictionary only contains the
        """

        result = None

        try:
            sql, args = create_select(ordertable, template=template, fie
            res, data = run_q(sql=sql, args=args, conn=self._cnx, commit
        except Exception as e:
            print("Exception e = ", e)
            raise e

        return list(data)



    def create_insert(table_name, new_row):
        sql = "insert into " + table_name + " "
        cols = list(new_row.keys())
        cols = ", ".join(cols)
        col_clause = "(" + cols + ") "
        args = list(new_row.values())
        s_stuff = ["%s"]*len(args)
        s_clause = ", ".join(s_stuff)
        v_clause = "values(" + s_clause + ")"
        sql += col_clause + v_clause
        return sql, args

    def insert(table_name, new_record):
        """
```

```
        :param new_record: A dictionary representing a row to add to the
        :return: None
        """
        # Get the list of columns.
        sql, args = create_insert(table_name, new_record)

        res, d = run_q(sql, args=args)
        return res

    data = %sql SELECT max(orderNumber) FROM classicmodels.orderdetails;
    df = pd.DataFrame(data)
    maxi = df.values[0][0] + 1

    order = order_info
    details = order["orderdetails"]
    del order["orderdetails"]

    order['orderNumber'] = str(maxi)
    for i in details:
        i['orderNumber'] = str(maxi)
        print(order)   #Note: Ive printed the queries to show the user w
    tup1 = insert(ordertable, order)
    tup2 = 0
    for i in details:
        res = insert(orderdetailstable, i)
        tup2 += res
    return(tup1, tup2)


    """
    Creates (Inserts) the data associated with an order. The order infor
    and line item/order detail item goes into the ordersdetails table.
    :param order_info: A dictionary. There are top-level elements for th
        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
        of rows inserted into each table.
    """
```

In [34]:
```
#Note: Ive printed the order detail queries to show the user the new ord
#I label the order number as the maximum order number plus 1 because it
#added.

null = None

orderinfo = {
            "orderNumber": 10888,
            "orderDate": "2003-05-20"
```

```
            "orderDate": "2003-05-20",
            "requiredDate": "2003-05-29",
            "shippedDate": "2003-05-22",
            "status": "Shipped",
            "comments": null,
            "customerNumber": 103,
            "orderdetails": [
                {
                    "orderNumber": 10888,
                    "productCode": "S18_1589",
                    "quantityOrdered": 26,
                    "priceEach": "120.71",
                    "orderLineNumber": 2
                },
                {
                    "orderNumber": 10888,
                    "productCode": "S18_2870",
                    "quantityOrdered": 46,
                    "priceEach": "114.84",
                    "orderLineNumber": 3
                },
                {
                    "orderNumber": 10888,
                    "productCode": "S18_3685",
                    "quantityOrdered": 34,
                    "priceEach": "117.26",
                    "orderLineNumber": 4
                },
                {
                    "orderNumber": 10888,
                    "productCode": "S24_1628",
                    "quantityOrdered": 50,
                    "priceEach": "43.27",
                    "orderLineNumber": 1
                }
            ]
        }

create_order(orderinfo)
```

```
DEBUG:root:Executing SQL = insert into classicmodels.orders (orderNumb
er, orderDate, requiredDate, shippedDate, status, comments, customerNu
mber) values('11010', '2003-05-20', '2003-05-29', '2003-05-22', 'Shipp
ed', NULL, 103)
DEBUG:root:Executing SQL = insert into classicmodels.orderdetails (ord
erNumber, productCode, quantityOrdered, priceEach, orderLineNumber) va
lues('11010', 'S18_1589', 26, '120.71', 2)
DEBUG:root:Executing SQL = insert into classicmodels.orderdetails (ord

erNumber, productCode, quantityOrdered, priceEach, orderLineNumber) va
```

```
lues('11010', 'S18_2870', 46, '114.84', 3)
DEBUG:root:Executing SQL = insert into classicmodels.orderdetails (ord
erNumber, productCode, quantityOrdered, priceEach, orderLineNumber) va
lues('11010', 'S18_3685', 34, '117.26', 4)
DEBUG:root:Executing SQL = insert into classicmodels.orderdetails (ord
erNumber, productCode, quantityOrdered, priceEach, orderLineNumber) va
lues('11010', 'S24_1628', 50, '43.27', 1)
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
1 rows affected.
{'orderNumber': '11010', 'orderDate': '2003-05-20', 'requiredDate': '2
003-05-29', 'shippedDate': '2003-05-22', 'status': 'Shipped', 'comment
s': None, 'customerNumber': 103}
{'orderNumber': '11010', 'orderDate': '2003-05-20', 'requiredDate': '2
003-05-29', 'shippedDate': '2003-05-22', 'status': 'Shipped', 'comment
s': None, 'customerNumber': 103}
{'orderNumber': '11010', 'orderDate': '2003-05-20', 'requiredDate': '2
003-05-29', 'shippedDate': '2003-05-22', 'status': 'Shipped', 'comment
s': None, 'customerNumber': 103}
{'orderNumber': '11010', 'orderDate': '2003-05-20', 'requiredDate': '2
003-05-29', 'shippedDate': '2003-05-22', 'status': 'Shipped', 'comment
s': None, 'customerNumber': 103}
```

Out[34]: (1, 4)

- batting_summary

  Put your create view statement here.

In [158]:

```
%%sql
DROP VIEW IF EXISTS batting_summary;

CREATE VIEW batting_summary(playerID, yearID, teamID, AB, H, HR, RBI)
AS SELECT playerID, yearID, teamID, AB, H, HR, RBI FROM BATTING;
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
0 rows affected.
0 rows affected.
```

Out[158]: []

In [159]:
```
#TEST FOR BATTING_SUMMARY
%sql select * from batting_summary where playerID='willite01'
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
19 rows affected.
```

Out[159]:

playerID	yearID	teamID	AB	H	HR	RBI
willite01	1939	BOS	565	185	31	145
willite01	1940	BOS	561	193	23	113
willite01	1941	BOS	456	185	37	120
willite01	1942	BOS	522	186	36	137
willite01	1946	BOS	514	176	38	123
willite01	1947	BOS	528	181	32	114
willite01	1948	BOS	509	188	25	127
willite01	1949	BOS	566	194	43	159
willite01	1950	BOS	334	106	28	97
willite01	1951	BOS	531	169	30	126
willite01	1952	BOS	10	4	1	3
willite01	1953	BOS	91	37	13	34
willite01	1954	BOS	386	133	29	89
willite01	1955	BOS	320	114	28	83
willite01	1956	BOS	400	138	24	82
willite01	1957	BOS	420	163	38	87
willite01	1958	BOS	411	135	26	85
willite01	1959	BOS	272	69	10	43
willite01	1960	BOS	310	98	29	72

In [160]:
```
%%sql DROP VIEW IF EXISTS pitching_summary;
CREATE VIEW pitching_summary(playerID, yearID, teamID, W, L, IPouts) AS
playerID, yearID, teamID, W, L, IPouts FROM PITCHING;
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
0 rows affected.
0 rows affected.
```

Out[160]:
```
[]
```

In [161]:
```
#TEST FOR PITCHING SUMMARY
%sql select * from pitching_summary where playerid='willite01';
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
1 rows affected.
```

Out[161]:

playerID	yearID	teamID	W	L	IPouts
willite01	1940	BOS	0	0	6

In [26]:
```
%%sql DROP VIEW IF EXISTS fielding_summary;
CREATE VIEW fielding_summary AS SELECT playerID, teamID, yearID, sum(PO)
group_concat(pos) AS 'POS'  FROM lahman2019clean.FIELDING group by playe
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
0 rows affected.
0 rows affected.
```

Out[26]:
```
[]
```

In [27]:
```
#TEST FOR FIELDING SUMMARY
%sql select * from lahman2019clean.fielding_summary where playerid='will
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
19 rows affected.
```

Out[27]:

playerID	teamID	yearID	PO	A	E	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

In [164]:
```
%%sql DROP VIEW IF EXISTS appearances_summary;
CREATE VIEW appearances_summary(playerID, yearID, G_all, GS) AS SELECT p
yearID, G_all, GS FROM lahman2019clean.APPEARANCES;
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
0 rows affected.
0 rows affected.
```

Out[164]:
```
[]
```

In [165]:
```
#TEST FOR APPEARANCE SUMMARY
%sql select * from appearances_summary where playerid = 'willite01'
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
19 rows affected.
```

Out[165]:

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

In [171]:
```
%%sql
DROP VIEW IF EXISTS annual_summary;

CREATE VIEW annual_summary(playerID, teamID, yearID, G_all, GS, AB, H, H
AS
SELECT appearances_summary.playerID, teamID, appearances_summary.yearID,
FROM
lahman2019clean.appearances_summary
LEFT JOIN
(SELECT batting_summary.playerID, batting_summary.teamID, batting_summar
FROM
lahman2019clean.batting_summary LEFT JOIN
(SELECT fielding_summary.playerID, fielding_summary.teamID, fielding_sum
FROM lahman2019clean.fielding_summary LEFT JOIN lahman2019clean.pitching
ON lahman2019clean.fielding_summary.playerID = lahman2019clean.pitching_
lahman2019clean.fielding_summary.yearID = lahman2019clean.pitching_summa
ON
lahman2019clean.batting_summary.playerID = PF.playerID and
lahman2019clean.batting_summary.yearID = PF.yearID) BPF
ON
lahman2019clean.appearances_summary.playerID = BPF.playerID and
lahman2019clean.appearances_summary.yearID = BPF.yearID;
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
0 rows affected.
0 rows affected.
```

Out[171]:
```
[]
```

In [172]:
```
%sql select * from annual_summary where playerid='willite01'
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
19 rows affected.
```

Out[172]:

playerID	teamID	yearID	G_all	GS	AB	H	HR	RBI	W	L	IPouts	po	a	e
willite01	BOS	1939	149	149	565	185	31	145	None	None	None	318.0	11.0	19.0
willite01	BOS	1940	144	143	561	193	23	113	0	0	6	302.0	15.0	13.0
willite01	BOS	1941	143	133	456	185	37	120	None	None	None	262.0	11.0	11.0
willite01	BOS	1942	150	150	522	186	36	137	None	None	None	312.0	15.0	4.0
willite01	BOS	1946	150	150	514	176	38	123	None	None	None	325.0	7.0	10.0
willite01	BOS	1947	156	156	528	181	32	114	None	None	None	347.0	10.0	9.0
willite01	BOS	1948	137	134	509	188	25	127	None	None	None	289.0	9.0	5.0
willite01	BOS	1949	155	155	566	194	43	159	None	None	None	337.0	12.0	6.0
willite01	BOS	1950	89	86	334	106	28	97	None	None	None	165.0	7.0	8.0
willite01	BOS	1951	148	147	531	169	30	126	None	None	None	315.0	12.0	4.0
willite01	BOS	1952	6	2	10	4	1	3	None	None	None	4.0	0.0	0.0
willite01	BOS	1953	37	26	91	37	13	34	None	None	None	31.0	1.0	1.0
willite01	BOS	1954	117	113	386	133	29	89	None	None	None	213.0	5.0	4.0
willite01	BOS	1955	98	93	320	114	28	83	None	None	None	170.0	5.0	2.0
willite01	BOS	1956	136	110	400	138	24	82	None	None	None	174.0	7.0	5.0
willite01	BOS	1957	132	125	420	163	38	87	None	None	None	215.0	2.0	1.0
willite01	BOS	1958	129	114	411	135	26	85	None	None	None	154.0	3.0	7.0
willite01	BOS	1959	103	75	272	69	10	43	None	None	None	94.0	4.0	3.0
willite01	BOS	1960	113	87	310	98	29	72	None	None	None	131.0	6.0	1.0

In [173]:
```
%%sql
drop view if exists career_summary;


create view career_summary as
SELECT playerID, G_all, GS, AB, H, HR, RBI, w, l, IPouts, po, a, e, pos
FROM

(SELECT playerID, sum(PO) as 'PO', sum(A) as 'A', sum(E) as 'E',
group_concat(distinct pos) AS 'POS'  FROM lahman2019clean.FIELDING group

JOIN

(SELECT playerID, sum(G_all) as 'G_all', sum(GS) as 'GS', sum(AB) as 'AB
sum(RBI) as 'rbi', sum(W) as 'w', sum(L) as 'l', sum(IPouts) as 'IPouts'
FROM lahman2019clean.annual_summary group by playerID) CAR

using(playerID);
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
0 rows affected.
0 rows affected.
```

Out[173]:
```
[]
```

In [174]:
```
%sql select * from career_summary limit 10;
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
10 rows affected.
```

Out[174]:

playerID	G_all	GS	AB	H	HR	RBI	w	l	IPouts	po	a	
aardsda01	331.0	0.0	4.0	0.0	0.0	0.0	16.0	18.0	1011.0	11.0	29.0	
aaronha01	3298.0	3173.0	12364.0	3771.0	755.0	2297.0	None	None	None	7436.0	429.0	1
aaronto01	437.0	206.0	944.0	216.0	13.0	94.0	None	None	None	1317.0	113.0	
aasedo01	448.0	91.0	5.0	0.0	0.0	0.0	66.0	60.0	3328.0	67.0	135.0	
abadan01	15.0	4.0	21.0	2.0	0.0	0.0	None	None	None	37.0	1.0	
abadfe01	762.0	6.0	16.0	1.0	0.0	0.0	15.0	69.0	1933.0	7.0	37.0	
abadijo01	48.0	0.0	196.0	44.0	0.0	20.0	None	None	None	129.0	3.0	
abbated01	1025.0	492.0	3587.0	904.0	11.0	354.0	None	None	None	1873.0	2368.0	3
abbeybe01	142.0	0.0	379.0	80.0	0.0	24.0	57.0	61.0	2964.0	17.0	134.0	
abbeych01	452.0	0.0	1756.0	493.0	19.0	280.0	0.0	0.0	6.0	920.0	92.0	1

In [177]:
```
%%sql
use classicmodels;
drop table if exists orders_copy;
create table orders_copy as select * from orders;
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
0 rows affected.
0 rows affected.
337 rows affected.
```

Out[177]:
```
[]
```

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

In [178]:
```
%sql select * from orders_copy join orderdetails using(orderNumber) wher
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
4 rows affected.
```

Out[178]:

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

- 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 [179]:
```
%%sql
select
    customers.customerNumber, customers.country, orders_copy.orderNumber
    customers join orders_copy
    using (customerNumber)
    where country = 'Australia'
order by status;
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
19 rows affected.
```

Out[179]:

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 [180]:
```
%%sql UPDATE  customers join orders_copy using(customerNumber) set order
WHERE customers.country='Australia' and orders_copy.status !='Shipped' a
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
2 rows affected.
```

Out[180]:
```
[]
```

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

In [181]:
```
%%sql
select
    customers.customerNumber, customers.country, orders_copy.orderNumber
    customers join orders_copy
    using (customerNumber)
    where country = 'Australia'
order by status;
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
19 rows affected.
```

Out[181]:

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

In [191]:
```
%%sql

ALTER TABLE classicmodels.orders ADD CONSTRAINT CK_Status1
    CHECK (status IN ('Shipped', 'Resolved', 'Cancelled', 'On Hold','Dis
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
337 rows affected.
```

Out[191]:
```
[]
```

In [68]:
```
#Lets try add a status called 'Sipping Shipped' which isnt valid
try:
    %sql INSERT into classicmodels.orders values ('19999', '2003-01-06',
    print("Getting here is bad.")
except Exception as e:
    print("This is OK, e = ", e)
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
This is OK, e =  (pymysql.err.InternalError) (3819, "Check constraint
'CK_Status' is violated.")
[SQL: INSERT into classicmodels.orders values ('19999', '2003-01-06',
'2003-01-13', '2003-01-10', 'Sipping Shipped', 'None', '999')]
(Background on this error at: http://sqlalche.me/e/2j85)
(http://sqlalche.me/e/2j85))
```

Type *Markdown* and LaTeX: α^2

Data Cleanup (10 Points)

In [92]:
```
%sql SELECT * FROM classicmodels.countrycodes limit 10;
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
10 rows affected.
```

Out[92]:

Name	Code
Andorra	AD
United Arab Emirates	AE
Afghanistan	AF
Antigua and Barbuda	AG
Anguilla	AI
Albania	AL
Armenia	AM
Angola	AO
Antarctica	AQ
Argentina	AR

In [93]:
```
%%sql
DROP TABLE IF EXISTS classicmodels.customers_clean;
create table classicmodels.customers_clean as select * from classicmodel
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
0 rows affected.
122 rows affected.
```

Out[93]:
```
[]
```

In [94]:
```
%sql select customerNumber, customerName, country from classicmodels.cus
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
10 rows affected.
```

Out[94]:

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

In [95]:
```
#Setting Up the table for data cleaning. Steps are as follows:
# 1. Changing data type of Code column in countrycode table to varchar(2
# 2. Altering country_code table first by making country code the primar
# 3. Adding country_code column in customers_clean
# 4. Adding foriegn key between code and country_code columns

%sql ALTER TABLE classicmodels.countrycodes MODIFY COLUMN Code varchar(2

try:
    %sql ALTER TABLE classicmodels.countrycodes ADD CONSTRAINT customers

    print("Primary key installed.")
except Exception as e:
    print("Primary Key was already installed")


%sql ALTER TABLE classicmodels.customers_clean ADD country_code VARCHAR(


%sql ALTER TABLE classicmodels.customers_clean ADD FOREIGN KEY (country_
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
0 rows affected.
 * mysql+pymysql://root:***@localhost/lahman2019clean
Primary Key was already installed
 * mysql+pymysql://root:***@localhost/lahman2019clean
0 rows affected.
 * mysql+pymysql://root:***@localhost/lahman2019clean
122 rows affected.
```

Out[95]:
```
[]
```

In [96]:
```
import pandas as pd
import numpy as np
data = %sql SELECT * FROM classicmodels.countrycodes
df1 = pd.DataFrame(data, columns = ['Country', 'Code'])
country_codes = df1.set_index('Country').T.to_dict('list')
def country_to_code(country, dicte):
    return dicte[country][0]
# country_to_code('Andorra', country_codes)

data2 = %sql SELECT * FROM classicmodels.customers_clean order by custom
df2 = pd.DataFrame(data2)
df2.head()

list_of_countries = np.asarray(df2[10].tolist(), dtype=object)
list_of_indices = list(np.asarray(df2[0].tolist(), dtype=object))



for i in range(len(list_of_countries)):
    if list_of_countries[i] == 'UK':
        list_of_countries[i]= 'United Kingdom'
    elif list_of_countries[i]== 'United State' or list_of_countries[i]==
        list_of_countries[i]='United States'
    elif list_of_countries[i]== 'Norway  ':
        list_of_countries[i]='Norway'
    elif list_of_countries[i]== 'Russia':
        list_of_countries[i]='Russian Federation'




for i in range(len(list_of_countries)):
    code = country_to_code(list_of_countries[i], country_codes) #convert
    list_of_countries[i] = code


(list(list_of_countries), list_of_indices)

# (len(list(list_of_countries)), len(list_of_indices))
list_of_countries = list(list_of_countries)
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
249 rows affected.
 * mysql+pymysql://root:***@localhost/lahman2019clean
122 rows affected.
```

In [97]:
```
for i in range(len(list_of_countries)):
    code = list_of_countries[i]
    number = list_of_indices[i]
    %sql update classicmodels.customers_clean SET country_code= :code WH
```

```
1 rows affected.
 * mysql+pymysql://root:***@localhost/lahman2019clean
1 rows affected.
 * mysql+pymysql://root:***@localhost/lahman2019clean
1 rows affected.
 * mysql+pymysql://root:***@localhost/lahman2019clean
1 rows affected.
 * mysql+pymysql://root:***@localhost/lahman2019clean
1 rows affected.
 * mysql+pymysql://root:***@localhost/lahman2019clean
1 rows affected.
 * mysql+pymysql://root:***@localhost/lahman2019clean
1 rows affected.
 * mysql+pymysql://root:***@localhost/lahman2019clean
1 rows affected.
 * mysql+pymysql://root:***@localhost/lahman2019clean
1 rows affected.
 * mysql+pymysql://root:***@localhost/lahman2019clean
1 rows affected.
 * mysql+pymysql://root:***@localhost/lahman2019clean
```

In [98]:
```
%sql ALTER TABLE classicmodels.customers_clean DROP country;
%sql select * from classicmodels.customers_clean limit 10;
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
0 rows affected.
 * mysql+pymysql://root:***@localhost/lahman2019clean
10 rows affected.
```

Out[98]:

customerNumber	customerName	contactLastName	contactFirstName	phone	addressLine1
103	Atelier graphique	Schmitt	Carine	40.32.2555	54, rue Royale
112	Signal Gift Stores	King	Jean	7025551838	8489 Strong St.
114	Australian Collectors, Co.	Ferguson	Peter	03 9520 4555	636 St Kilda Road
119	La Rochelle Gifts	Labrune	Janine	40.67.8555	67, rue des Cinquante Otages
121	Baane Mini Imports	Bergulfsen	Jonas	07-98 9555	Erling Skakkes gate 78
124	Mini Gifts Distributors Ltd.	Nelson	Susan	4155551450	5677 Strong St.
125	Havel & Zbyszek Co	Piestrzeniewicz	Zbyszek	(26) 642-7555	ul. Filtrowa 68
128	Blauer See Auto, Co.	Keitel	Roland	+49 69 66 90 2555	Lyonerstr. 34
129	Mini Wheels Co.	Murphy	Julie	6505555787	5557 North Pendale Street
131	Land of Toys Inc.	Lee	Kwai	2125557818	897 Long Airport Avenue

In [99]:
```
try:
    %sql update classicmodels.customers_clean set country_code = 'XX' wh
    print("Getting here is bad.")
except Exception as e:
    print("This is OK, e = ", e)
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
This is OK, e =  (pymysql.err.IntegrityError) (1452, 'Cannot add or up
date a child row: a foreign key constraint fails (`classicmodels`.`cus
tomers_clean`, CONSTRAINT `customers_clean_ibfk_1` FOREIGN KEY (`count
ry_code`) REFERENCES `countrycodes` (`Code`))')
[SQL: update classicmodels.customers_clean set country_code = 'XX' whe
re customerNumber=103]
(Background on this error at: http://sqlalche.me/e/gkpj)
(http://sqlalche.me/e/gkpj))
```

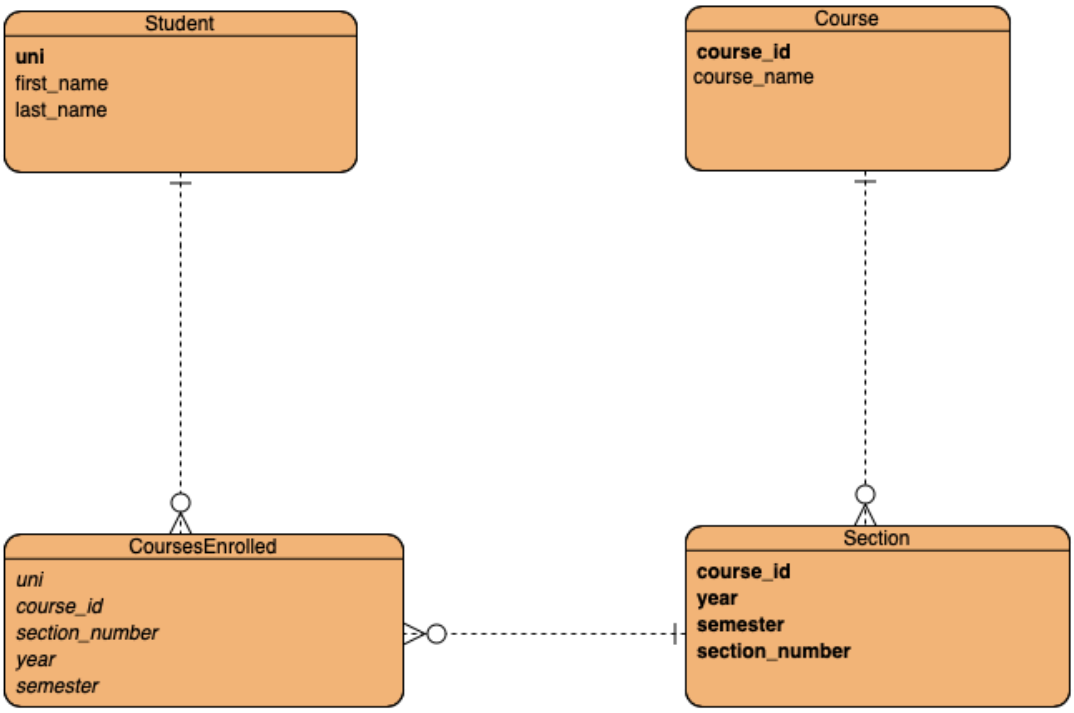
This one is really unpleasant!

E-R Diagrams Competency

Answer

Note: Bolded words (representing columns) represent primary keys and italicized words represent foriegn keys.

Explanation: If we group by UNI in the CoursesEnrolled table, we get a representation of student enrollment in each unique class.



Type *Markdown* and LaTeX: α^2

Inheritance and Stored Procedures (10 points)

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

```
CREATE TABLE `student` (
  `uni` varchar(12) NOT NULL,
  `last_name` varchar(64) NOT NULL,
  `first_name` varchar(64) NOT NULL,
  `graduation_year` year(4) NOT NULL,
  PRIMARY KEY (`uni`)
) 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 Prof
essor','Adjunct 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

1.

```
CREATE VIEW PEOPLE AS
SELECT uni, last_name, first_name, 'S' as 'type', graduation_yea
r, 'NA' as title FROM
(SELECT * FROM student) as s
UNION
SELECT uni, last_name, first_name, 'F' as 'type', 'NA' as 'gradu
ation_year', title FROM
(SELECT * FROM faculty) as f;
```

2. a

```
Delimiter //
CREATE DEFINER=`root`@`localhost` FUNCTION `generate_uni` (first
_name varchar(32), last_name varchar(32)) RETURNS varchar(12) CH
ARSET utf8 deterministic
    BEGIN
        declare f_prefix varchar(2);
        declare l_prefix varchar(2);
        declare prefix_count int;
        declare full_prefix varchar(5);
        declare result varchar(12);
        set f_prefix = lower(substr(first_name, 1, 2));
        set l_prefix = lower(substr(last_name, 1, 2));
        set full_prefix = concat(f_prefix, l_prefix, '%');
        set prefix_count = (select count(*) as count from person
3 where uni like(full_prefix));
        set result = concat(f_prefix, l_prefix, prefix_count+1);
        return result;
    END
```

b.

```
CREATE PROCEDURE insert_based_on_type
(
     in last_name varchar(64),
    in first_name varchar(64),
    in type enum('F', 'S'),
    in title varchar(32)
)
BEGIN
        declare uni varchar(12);

        if type != 'S' and  type != 'F' then
            signal sqlstate '50001'
            set message_text = 'incorrect type';
        END IF;

        SET uni = generate_uni(last_name, first_name);
        If type = 'S' then
            insert into student(uni, last_name, first_name, grad
uation_year)
            values (uni, last_name, first_name, graduation_year)
;

        ELSE
            insert into faculty(uni, last_name, first_name, titl
e) values(uni, first_name, last_name, title)
        End IF;
END
```

Reference for PROCEDURE: Professor's recitation

In [23]:
```
%%sql
call insert_based_on_type(
    "Harry",
    "Johnson",
    "S",
    2022,
    'NA'
)
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
1 rows affected.
```

In [15]:
```
%sql select * FROM W4111Midterm.people;
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
1 rows affected.
```

Out[15]:

uni	last_name	first_name	graduation_year	title	kind
hajo1	Harry	Johnson	2022	NA	S

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.

In [144]:
```
%sql select * from W4111Midterm.got_nodes limit 10;
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
10 rows affected.
```

Out[144]:

Id	Label
Id	Label
Aemon	Aemon
Grenn	Grenn
Samwell	Samwell
Aerys	Aerys
Jaime	Jaime
Robert	Robert
Tyrion	Tyrion
Tywin	Tywin
Alliser	Alliser

In [145]:
```
%sql select * from W4111Midterm.got_edges limit 10;
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
10 rows affected.
```

Out[145]:

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

Shortest path

In [146]:
```
%%sql
use W4111Midterm;
drop table if exists jump_one;
create table jump_one as
select source as one_source, target as one_target from got_edges
union
select target as one_source, source as one_target from got_edges;

drop table if exists two_jump;
create table two_jump as select a.*,jump_one.one_source as two_source, j
from (select * from jump_one where one_source in ('Craster','Roose')) as
    join jump_one on a.one_target = jump_one.one_source;
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
0 rows affected.
0 rows affected.
704 rows affected.
0 rows affected.
127 rows affected.
```

Out[146]:
```
[]
```

In [147]:
```
%%sql
select
    two_jump.*, jump_one.*
from two_jump join jump_one
    on two_jump.two_target = jump_one.one_source
    where jump_one.one_target in ('Roose', 'Craster') and two_jump.one_s
```

```
 * mysql+pymysql://root:***@localhost/lahman2019clean
4 rows affected.
```

Out[147]:

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