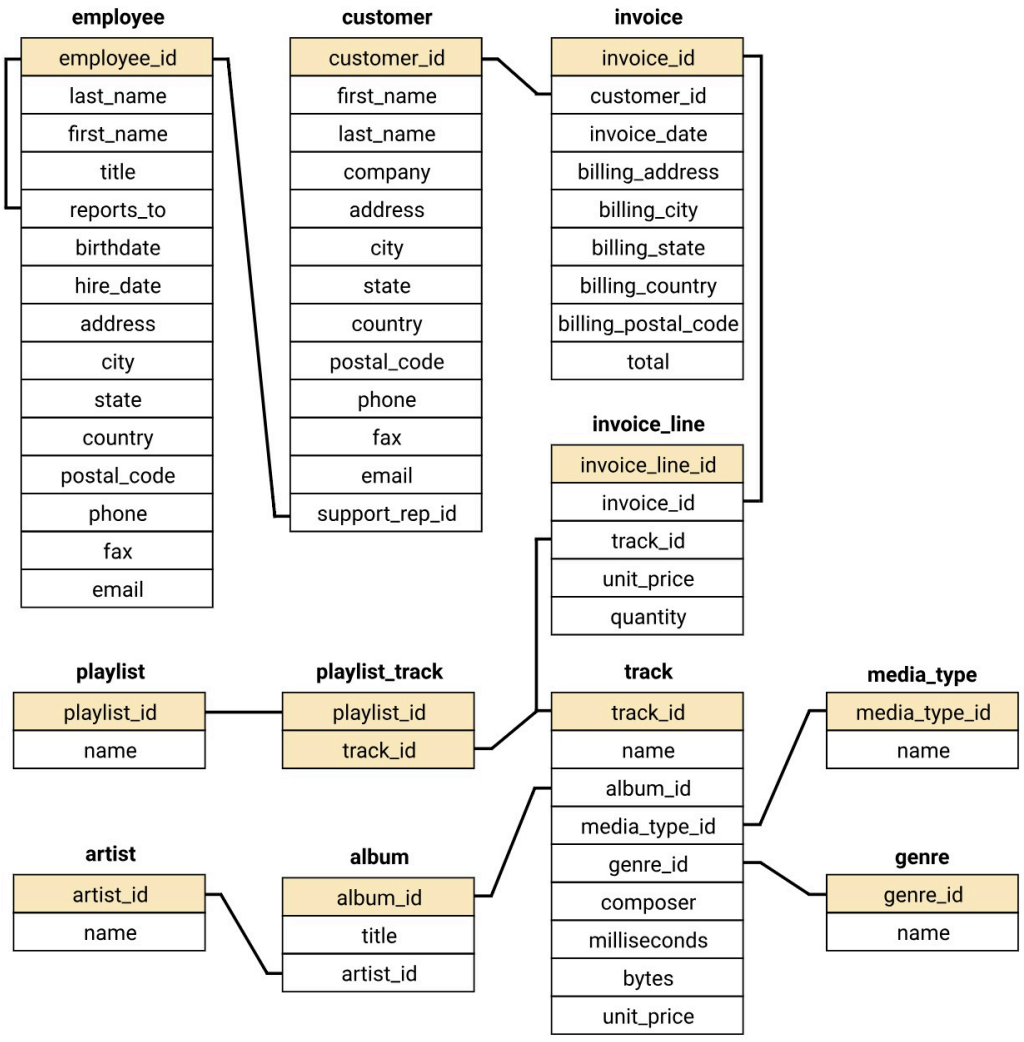
In [1]:
```
import pandas as pd
import sqlite3 as sql
```

In [2]:
```
database = "chinook.db"
conn = sql.connect(database)
```

In [3]:
```
def read_query(q):
    return pd.read_sql_query(q, conn)
```



1 - Call all tables from master where type == 'table'

In [4]:
```
read_query('''select * from sqlite_master where type == "table"''')
```

Out[4]:

	type	name	tbl_name	rootpage	sql
0	table	album	album	2	CREATE TABLE [album]\n(\n [album_id] INTEGE...
1	table	artist	artist	3	CREATE TABLE [artist]\n(\n [artist_id] INTE...
2	table	customer	customer	4	CREATE TABLE [customer]\n(\n [customer_id] ...
3	table	employee	employee	5	CREATE TABLE [employee]\n(\n [employee_id] ...
4	table	genre	genre	6	CREATE TABLE [genre]\n(\n [genre_id] INTEGE...
5	table	invoice	invoice	7	CREATE TABLE [invoice]\n(\n [invoice_id] IN...
6	table	invoice_line	invoice_line	8	CREATE TABLE [invoice_line]\n(\n [invoice_l...
7	table	media_type	media_type	9	CREATE TABLE [media_type]\n(\n [media_type_...
8	table	playlist	playlist	10	CREATE TABLE [playlist]\n(\n [playlist_id] ...
9	table	playlist_track	playlist_track	11	CREATE TABLE [playlist_track]\n(\n [playlis...
10	table	track	track	13	CREATE TABLE [track]\n(\n [track_id] INTEGE...
11	table	wishlist_track	wishlist_track	261	CREATE TABLE wishlist_track(\nwishlist_id Inte...
12	table	wishlist	wishlist	260	CREATE TABLE "wishlist" (\n\t"whishlist_id"\tI...

2 - Explore the invoice table for invoice_id = 1

In [5]:
```
read_query('''select * from invoice where invoice_id = 1''')
```

Out[5]:

	invoice_id	customer_id	invoice_date	billing_address	billing_city	billing_st
0	1	18	2017-01-03 00:00:00	627 Broadway	New York	

3 - Now check the same invoice_id i.e. 1 in the invoice_line table

In [6]:
```
read_query('''select * from invoice_line where invoice_id = 1''')
```

Out[6]:

	invoice_line_id	invoice_id	track_id	unit_price	quantity
0	1	1	1158	0.99	1
1	2	1	1159	0.99	1
2	3	1	1160	0.99	1
3	4	1	1161	0.99	1
4	5	1	1162	0.99	1
5	6	1	1163	0.99	1
6	7	1	1164	0.99	1
7	8	1	1165	0.99	1
8	9	1	1166	0.99	1
9	10	1	1167	0.99	1
10	11	1	1168	0.99	1
11	12	1	1169	0.99	1
12	13	1	1170	0.99	1
13	14	1	1171	0.99	1
14	15	1	1172	0.99	1
15	16	1	1173	0.99	1

`You may have notice that for every invoice, multiple tracks have been sold. Consider an invoice as a restaurant bill where you have order multiple food items.`

4 - For invoice_id = 1, calculate the total cost (unit_price) of all the tracks sold

In [7]:
```
read_query('select sum(unit_price) from invoice_line where invoice_id = 1')
```

Out[7]:

	sum(unit_price)
0	15.84

Multi Joins

5 - In the above schema, find the following columns:

- Invoice_id, Track_id, Track_Name, Track_type name, Quantity, Unit_Price

Notice that they all belong to multiple tables. We have learned that in order to bring data from two table you have to join them based on your common key. Using the same concept return the above columns by using joins for between the tables.

In [8]:
```
read_query('''select il.invoice_id, t.track_id, t.name Track_name, mt.name m
            from invoice_line il inner join track t on il.track_id = t.track
            inner join media_type mt on t.media_type_id = mt.media_type_id''
```

Out[8]:

	invoice_id	track_id	Track_name	media_type_name	quantity	unit_price
0	1	1158	Right Next Door to Hell	Protected AAC audio file	1	0.99
1	1	1159	Dust N' Bones	Protected AAC audio file	1	0.99
2	1	1160	Live and Let Die	Protected AAC audio file	1	0.99
3	1	1161	Don't Cry (Original)	Protected AAC audio file	1	0.99
4	1	1162	Perfect Crime	Protected AAC audio file	1	0.99
...	...	...	...	...	...	..
4752	614	2659	Every Breath You Take	MPEG audio file	1	0.99
4753	614	2660	King Of Pain	MPEG audio file	1	0.99
4754	614	2661	Wrapped Around Your Finger	MPEG audio file	1	0.99
4755	614	2662	Don't Stand So Close to Me '86	MPEG audio file	1	0.99
4756	614	2663	Message in a Bottle (new classic rock mix)	MPEG audio file	1	0.99

4757 rows × 6 columns

Self Join

6 - In the above schema, notice the employee table is connected to itself from `employee_id` to `reports_to` column.

In an office scenario, your manager is also an employee for the company just like you, hence the `reports_to` columns specifies the `employee_id` of your manager.

By calling the employee table twice and using aliases return an employee's full name, title and it's supervisor/manager's full name

In [9]:
```
read_query('''select * from employee''')
```

Out[9]:

	employee_id	last_name	first_name	title	reports_to	birthdate	hire_da
0	1	Adams	Andrew	General Manager	NaN	1962-02-18 00:00:00	2016- 00:00
1	2	Edwards	Nancy	Sales Manager	1.0	1958-12-08 00:00:00	2016- 00:00
2	3	Peacock	Jane	Sales Support Agent	2.0	1973-08-29 00:00:00	2017- 00:00
3	4	Park	Margaret	Sales Support Agent	2.0	1947-09-19 00:00:00	2017- 00:00
4	5	Johnson	Steve	Sales Support Agent	2.0	1965-03-03 00:00:00	2017- 00:00
5	6	Mitchell	Michael	IT Manager	1.0	1973-07-01 00:00:00	2016- 00:00
6	7	King	Robert	IT Staff	6.0	1970-05-29 00:00:00	2017- 00:00
7	8	Callahan	Laura	IT Staff	6.0	1968-01-09 00:00:00	2017- 00:00

In [10]:
```
read_query('''select e1.first_name || " " || e1.last_name Employee_Name,
                e1.title,
                e2.first_name || " " || e2.last_name Supervisor_name
```

```
                from employee e1 inner join employee e2 on
                e1.reports_to = e2.employee_id''')
```

Out[10]:

	Employee_Name	title	Supervisor_name
0	Nancy Edwards	Sales Manager	Andrew Adams
1	Jane Peacock	Sales Support Agent	Nancy Edwards
2	Margaret Park	Sales Support Agent	Nancy Edwards
3	Steve Johnson	Sales Support Agent	Nancy Edwards
4	Michael Mitchell	IT Manager	Andrew Adams
5	Robert King	IT Staff	Michael Mitchell
6	Laura Callahan	IT Staff	Michael Mitchell

Left Join

7 - In the above schema, notice the customer and invoice table.

We have information for every customer in the customer table and whenever the customer has bought something an invoice is generated and saved in the invoice table.

We would like to know every customer's full name and the total number of times he/she has bought from our store (using the quantity of invoices generated) and also would like to know the total amount spend by that customer in the visits.

Use Left Join keeping the customer table on the left in this scenario. The reason for left join is that what if the customer data is available in customer table but the customer has never bought anything yet so no invoice would be generated, and in this case left join would return that customer but zero values in number of boughts and the amount spend.

If we use Inner Join then only those customers will come which have bought atleast one time but that is not our objective

In [11]:

```
read_query('''select c.first_name || " " || c.last_name Customer_Name,
                count(i.invoice_id) No_of_Visits,
                sum(i.subtotal) Total_Spending
                from customer c left join invoice i on
                c.customer_id = i.customer_id
                group by c.customer_id''')
```

Out[11]:

	Customer_Name	No_of_Visits	Total_Spending
0	Luís Gonçalves	13	108.90
1	Leonie Köhler	11	82.17
2	François Tremblay	9	99.99
3	Bjørn Hansen	9	72.27
4	František Wichterlová	18	144.54
5	Helena Holý	12	128.70
6	Astrid Gruber	9	69.30
7	Daan Peeters	7	60.39
8	Kara Nielsen	10	37.62
9	Eduardo Martins	12	60.39
10	Alexandre Rocha	10	69.30
11	Roberto Almeida	11	82.17
12	Fernanda Ramos	15	106.92
13	Mark Philips	10	29.70
14	Jennifer Peterson	9	66.33
15	Frank Harris	8	74.25
16	Jack Smith	12	98.01
17	Michelle Brooks	8	79.20
18	Tim Goyer	9	54.45
19	Dan Miller	12	95.04
20	Kathy Chase	11	91.08
21	Heather Leacock	12	92.07
22	John Gordon	10	66.33
23	Frank Ralston	8	71.28
24	Victor Stevens	10	76.23
25	Richard Cunningham	12	86.13
26	Patrick Gray	9	84.15
27	Julia Barnett	10	72.27
28	Robert Brown	4	40.59
29	Edward Francis	13	91.08
30	Martha Silk	11	62.37
31	Aaron Mitchell	8	70.29
32	Ellie Sullivan	12	75.24

	Customer_Name	No_of_Visits	Total_Spending
33	João Fernandes	13	102.96
34	Madalena Sampaio	16	82.17
35	Hannah Schneider	11	85.14
36	Fynn Zimmermann	10	94.05
37	Niklas Schröder	9	73.26
38	Camille Bernard	9	79.20
39	Dominique Lefebvre	9	72.27
40	Marc Dubois	9	64.35
41	Wyatt Girard	11	99.99
42	Isabelle Mercier	12	73.26
43	Terhi Hämäläinen	11	79.20
44	Ladislav Kovács	10	78.21
45	Hugh O'Reilly	13	114.84
46	Lucas Mancini	9	50.49
47	Johannes Van der Berg	10	65.34
48	Stanisław Wójcik	10	76.23
49	Enrique Muñoz	11	98.01
50	Joakim Johansson	10	75.24
51	Emma Jones	8	68.31
52	Phil Hughes	11	98.01
53	Steve Murray	9	79.20
54	Mark Taylor	10	81.18
55	Diego Gutiérrez	5	39.60
56	Luis Rojas	13	97.02
57	Manoj Pareek	13	111.87
58	Puja Srivastava	8	71.28

8 - Modify the above query to only bring the top 3 big spending customers

In [12]:
```
read_query('''select c.first_name || " " || c.last_name Customer_Name,
                count(i.invoice_id) No_of_Visits,
                sum(i.subtotal) Total_Spending
                from customer c left join invoice i on
                c.customer_id = i.customer_id
                group by c.customer_id
                order by Total_Spending DESC
                limit 3''')
```

Out[12]:

	Customer_Name	No_of_Visits	Total_Spending
0	František Wichterlová	18	144.54
1	Helena Holý	12	128.70
2	Hugh O'Reilly	13	114.84

9 - Modify the query from task 7 to bring only those customers which have spent atleast 100 dollars at our store

In [13]:
```
read_query('''select c.first_name || " " || c.last_name Customer_Name,
                count(i.invoice_id) No_of_Visits,
                sum(i.subtotal) Total_Spending
                from customer c left join invoice i on
                c.customer_id = i.customer_id
                group by c.customer_id
                having Total_Spending >= 100
                ''')
```

Out[13]:

	Customer_Name	No_of_Visits	Total_Spending
0	Luís Gonçalves	13	108.90
1	František Wichterlová	18	144.54
2	Helena Holý	12	128.70
3	Fernanda Ramos	15	106.92
4	João Fernandes	13	102.96
5	Hugh O'Reilly	13	114.84
6	Manoj Pareek	13	111.87

Case - End Case

10 - We want to categorise each customer such that if the customer has spent less than 40 then he/she is a small spender, if the customer has spent more than 100 then he/she is a big spender, else the customer is a regular. Make this category a seperate column called Customer_Category. Modify the query from task 7 and use Case for this, the syntax is below:

Case When _Condition_ Then _Statement_ When _Second Condition_ Then _Statement_ else _Statement_ End _Column name you want to give_

In [14]:
```
read_query('''select c.first_name || " " || c.last_name Customer_Name,
                count(i.invoice_id) No_of_Visits,
                sum(i.subtotal) Total_Spending,
                CASE WHEN sum(i.subtotal) > 100 THEN "Big Spender"
                    When sum(i.subtotal) < 40 THEN "Small Spender"
                    ELSE "Regular" END Customer_Category
                from customer c left join invoice i on
                c.customer_id = i.customer_id
                group by c.customer_id
                order by Total_Spending''')
```

Out[14]:

	Customer_Name	No_of_Visits	Total_Spending	Customer_Category
0	Mark Philips	10	29.70	Small Spender
1	Kara Nielsen	10	37.62	Small Spender
2	Diego Gutiérrez	5	39.60	Small Spender
3	Robert Brown	4	40.59	Regular
4	Lucas Mancini	9	50.49	Regular
5	Tim Goyer	9	54.45	Regular
6	Daan Peeters	7	60.39	Regular
7	Eduardo Martins	12	60.39	Regular
8	Martha Silk	11	62.37	Regular
9	Marc Dubois	9	64.35	Regular
10	Johannes Van der Berg	10	65.34	Regular
11	Jennifer Peterson	9	66.33	Regular
12	John Gordon	10	66.33	Regular
13	Emma Jones	8	68.31	Regular
14	Astrid Gruber	9	69.30	Regular
15	Alexandre Rocha	10	69.30	Regular
16	Aaron Mitchell	8	70.29	Regular
17	Frank Ralston	8	71.28	Regular
18	Puja Srivastava	8	71.28	Regular
19	Julia Barnett	10	72.27	Regular
20	Dominique Lefebvre	9	72.27	Regular
21	Bjørn Hansen	9	72.27	Regular
22	Niklas Schröder	9	73.26	Regular
23	Isabelle Mercier	12	73.26	Regular
24	Frank Harris	8	74.25	Regular
25	Joakim Johansson	10	75.24	Regular
26	Ellie Sullivan	12	75.24	Regular
27	Stanisław Wójcik	10	76.23	Regular
28	Victor Stevens	10	76.23	Regular
29	Ladislav Kovács	10	78.21	Regular
30	Michelle Brooks	8	79.20	Regular
31	Camille Bernard	9	79.20	Regular
32	Terhi Hämäläinen	11	79.20	Regular

	Customer_Name	No_of_Visits	Total_Spending	Customer_Category
33	Steve Murray	9	79.20	Regular
34	Mark Taylor	10	81.18	Regular
35	Leonie Köhler	11	82.17	Regular
36	Roberto Almeida	11	82.17	Regular
37	Madalena Sampaio	16	82.17	Regular
38	Patrick Gray	9	84.15	Regular
39	Hannah Schneider	11	85.14	Regular
40	Richard Cunningham	12	86.13	Regular
41	Edward Francis	13	91.08	Regular
42	Kathy Chase	11	91.08	Regular
43	Heather Leacock	12	92.07	Regular
44	Fynn Zimmermann	10	94.05	Regular
45	Dan Miller	12	95.04	Regular
46	Luis Rojas	13	97.02	Regular
47	Jack Smith	12	98.01	Regular
48	Enrique Muñoz	11	98.01	Regular
49	Phil Hughes	11	98.01	Regular
50	François Tremblay	9	99.99	Regular
51	Wyatt Girard	11	99.99	Regular
52	João Fernandes	13	102.96	Big Spender
53	Fernanda Ramos	15	106.92	Big Spender
54	Luís Gonçalves	13	108.90	Big Spender
55	Manoj Pareek	13	111.87	Big Spender
56	Hugh O'Reilly	13	114.84	Big Spender
57	Helena Holý	12	128.70	Big Spender
58	František Wichterlová	18	144.54	Big Spender

11 - In the schema, notice the three tables - playlist, playlist_track and track. The playlist_track table is serving as the basis/bridge to connect playlist and track and to remove repetition.

Return playlist_id, playlist_name, every track_id of the track existing in the specific playlist (there can be multiple songs/tracks in a single playlist) and also the duration of each track which is in milliseconds. Make sure to convert the duration in seconds by dividing by 1000.

In [15]:
```
read_query('''select ply.playlist_id, t.track_id, t.milliseconds/1000 Seconc
            from playlist ply left join playlist_track plt
            on ply.playlist_id = plt.playlist_id
            inner join track t
            on plt.track_id = t.track_id''')
```

Out[15]:

	playlist_id	track_id	Seconds
0	1	3402	294
1	1	3389	252
2	1	3390	217
3	1	3391	260
4	1	3392	230
...	...	...	...
8710	17	2094	312
8711	17	2095	295
8712	17	2096	290
8713	17	3290	332
8714	18	597	197

8715 rows × 3 columns

Common Table Expressions (CTE)

`A common table expression serves a virtual table existing between the main table and your query and can only be accessible by your query. CTEs can help simplify, shorten, and organize your code.`

12 - Notice the query results in task 11, for every playlist we have a track and its relative duration. Return the total number of tracks in each playlist and the total duration of each playlist.

If you were to directly perform this task without performing task 11 and making that query a CTE, it would have been more complex. So basically you are breaking your query in two parts, first by returning data from main table (task 11) then using the data that is returned to perform your aggregation (task 12).

Syntax for CTE is below:

with _CTE name_ as (your query in task 11 serving as intermediate) _Your main query but this time give the name of your CTE when writing FROM for your query_

In [16]:
```
read_query('''
WITH PLAYLIST_INFO  AS (
       select ply.playlist_id, t.track_id, t.milliseconds/1000 Seconds
```

```
        from playlist ply left join playlist_track plt on
        ply.playlist_id = plt.playlist_id inner join track t
        on plt.track_id = t.track_id
        )
SELECT playlist_id, count(track_id), SUM(Seconds)
FROM PLAYLIST_INFO
GROUP BY playlist_id
        ''')
```

Out[16]:

	playlist_id	count(track_id)	SUM(Seconds)
0	1	3290	876049
1	3	213	500987
2	5	1477	397970
3	8	3290	876049
4	9	1	294
5	10	213	500987
6	11	39	9464
7	12	75	21736
8	13	25	6742
9	14	25	7565
10	15	25	7429
11	16	15	4114
12	17	26	8189
13	18	1	197

Views

`A view is the result set of a stored query that presents a limited perspective of the database to a user. They are like a virtual table.`

The difference between a CTE and a view is that a CTE is a temporary result which is only used in the scope of that specific query. However, Views are permanent objects created in your database, which helps in better maintainability and reusability.

If you are writing a query which is being used frequently again and again then consider making it a view. But if you are writing a query which is being used as a virtual table/intermediate only for a specific query then go for CTE.

13 - Return all the information of customers from customer table and their total spend from invoice such that customer belongs to USA with total spend more

than 90 dollars.

Run the commands below as it is after your query

In [17]:
```
read_query('''select c.*,
                sum(i.subtotal) Total_Spending
                from customer c left join invoice i on
                c.customer_id = i.customer_id
                WHERE c.country = 'USA'
                group by c.customer_id
                having Total_Spending > 90
            ''')
```

Out[17]:

	customer_id	first_name	last_name	company	address	city	state	c
0	17	Jack	Smith	Microsoft Corporation	1 Microsoft Way	Redmond	WA	
1	20	Dan	Miller	None	541 Del Medio Avenue	Mountain View	CA	
2	21	Kathy	Chase	None	801 W 4th Street	Reno	NV	
3	22	Heather	Leacock	None	120 S Orange Ave	Orlando	FL	

In [18]:
```
q = """drop view CUSTOMER_USA_90"""

# read_query(q)
conn.execute(q)
```

Out[18]:
```
<sqlite3.Cursor at 0x27852ee8f80>
```

In [19]:
```
q = """select * from sqlite_master where type == "view" """

read_query(q)
```

Out[19]:

	type	name	tbl_name	rootpage	sql

Now Put your query in a view the syntax is below:

Create View _View Name_ as _Your Query_

In [20]:
```
read_query('''select c.*, sum(i.subtotal) Total_Spending
                from customer c left join invoice i on c.customer_id = i.cus
                where country = "USA"
```

```
                group by c.customer_id
                having Total_Spending > 90''')
```

Out[20]:

	customer_id	first_name	last_name	company	address	city	state
0	17	Jack	Smith	Microsoft Corporation	1 Microsoft Way	Redmond	WA
1	20	Dan	Miller	None	541 Del Medio Avenue	Mountain View	CA
2	21	Kathy	Chase	None	801 W 4th Street	Reno	NV
3	22	Heather	Leacock	None	120 S Orange Ave	Orlando	FL

In [21]:
```
conn.execute('''CREATE VIEW CUSTOMER_USA_90 AS
                SELECT c.*, SUM(i.subtotal) AS Total_Spending
                FROM customer c
                LEFT JOIN invoice i ON c.customer_id = i.customer_id
                WHERE country = 'USA'
                GROUP BY c.customer_id
                HAVING Total_Spending > 90''')
```

Out[21]:
```
<sqlite3.Cursor at 0x27852f76420>
```

Now run the following commands below

In [22]:
```
q = """select * from sqlite_master where type == "view" """

read_query(q)
```

Out[22]:

	type	name	tbl_name	rootpage	sql
0	view	CUSTOMER_USA_90	CUSTOMER_USA_90	0	CREATE VIEW CUSTOMER_USA_90 AS\n ...

In [23]:
```
q = "select * from CUSTOMER_USA_90"

read_query(q)
```

Out[23]:

	customer_id	first_name	last_name	company	address	city	state	c
0	17	Jack	Smith	Microsoft Corporation	1 Microsoft Way	Redmond	WA	
1	20	Dan	Miller	None	541 Del Medio Avenue	Mountain View	CA	
2	21	Kathy	Chase	None	801 W 4th Street	Reno	NV	
3	22	Heather	Leacock	None	120 S Orange Ave	Orlando	FL	

In [ ]: