Complex Joins in SQL

Why is this important?

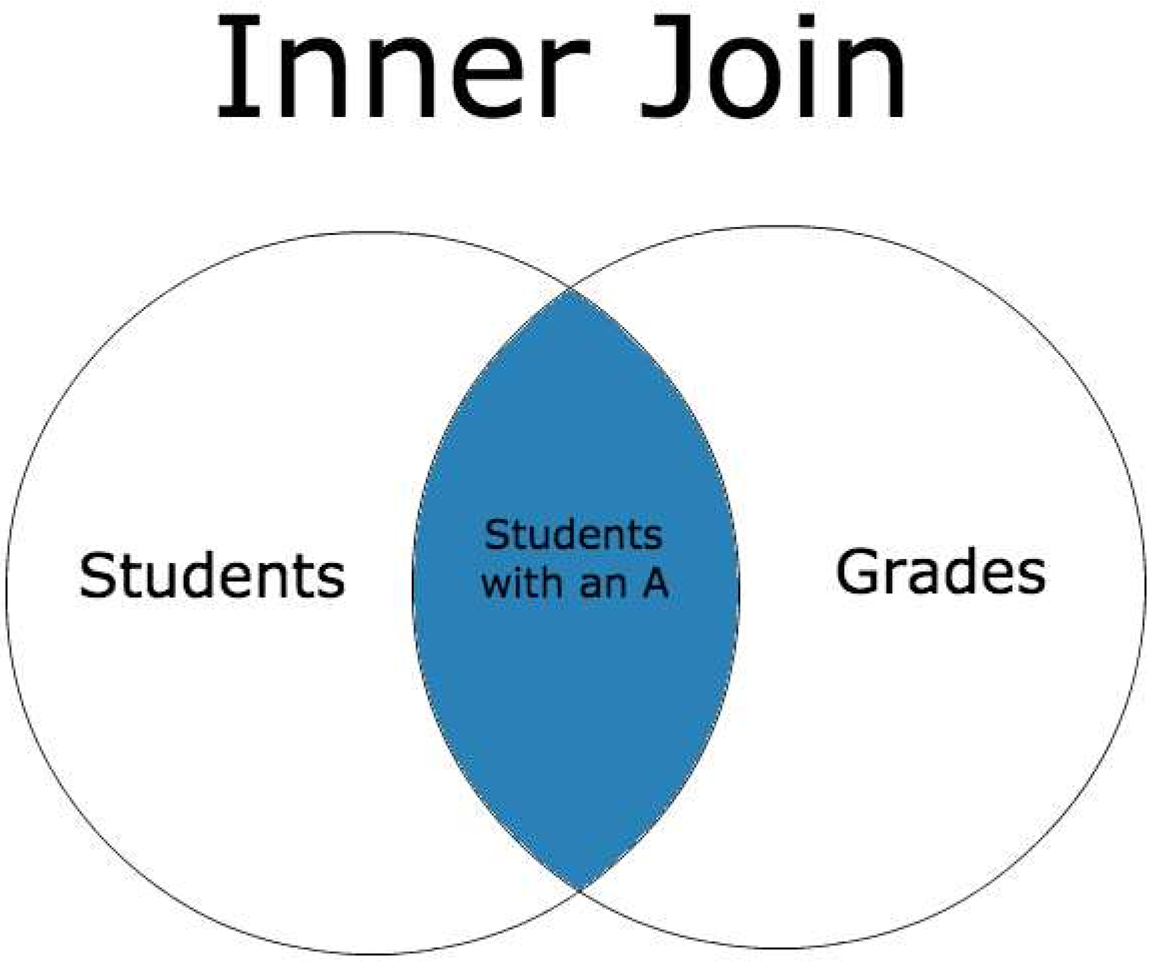
Grade Example (Inner Join)

Imagine you want to get a list of all the students with an "A" in the class. We only want those students in the class with top grades, ignoring the other students in the class.

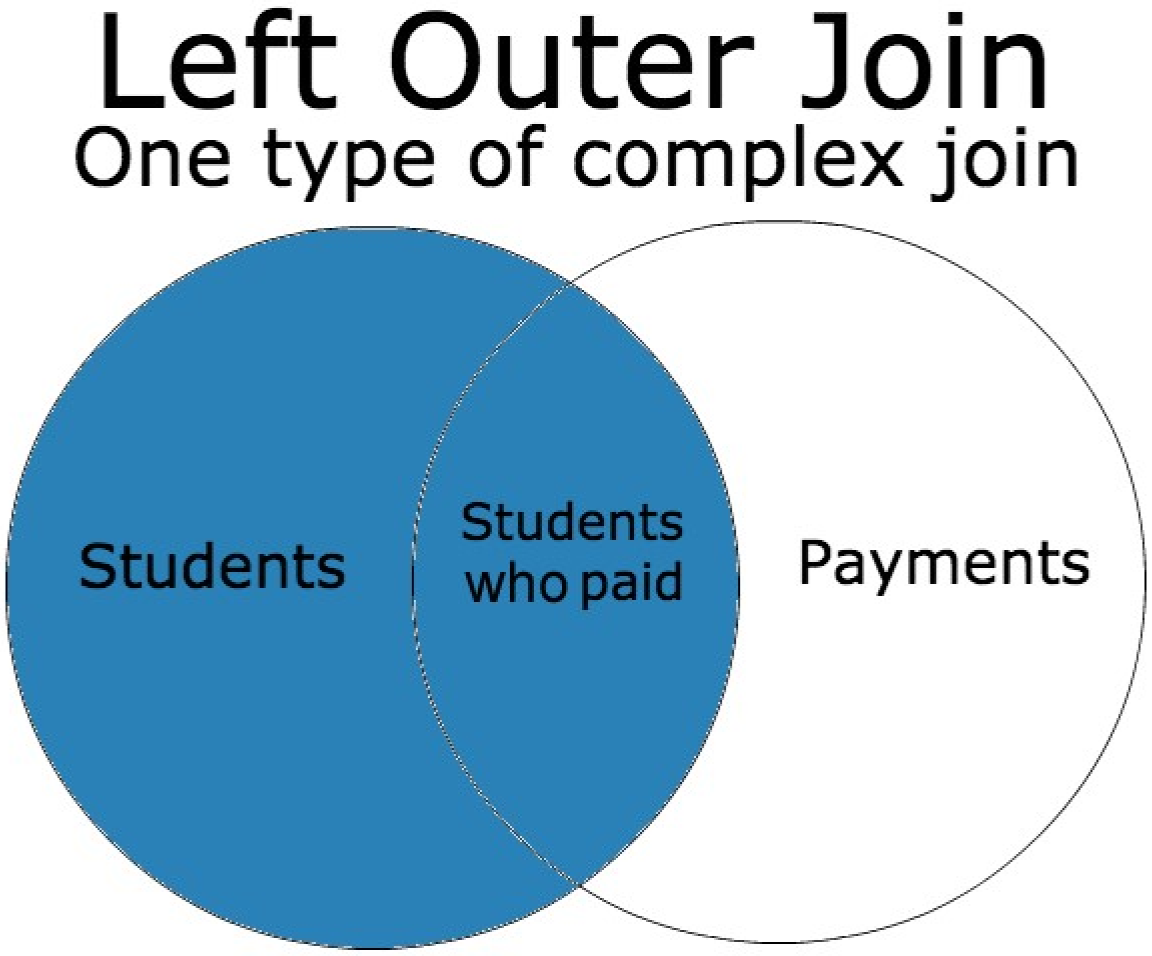
Field Trip Example (Complex/Outer Join)

Now imagine another scenario where the class is going on a field trip. The cost of the field trip is $10 per student. As a teacher, we want to keep track of which students have paid AND which students still need to pay.

Everything we've done up until this point looks like the Grade Example. This is an inner join. We only want the students with a certain grade. You can imagine a Venn Diagram where one circle is "Grades" and another circle is "Students". We only want the overlapping (or "inner") parts of the two circles.



Complex joins are useful and important when it comes to situations like the Field Trip Example. Sticking with the Venn Diagrams, we can think about "Students" as one circle and "Payments" as another circle. A complex join (or outer join) will return the overlap between the two circles AND the rest (or the "outer" part) of the "Students" circle as well.



We'll elaborate more on visualizing joins in the Venn Diagrams section below.

Overview

A complex join in SQL is also referred to as an outer join. It is not necessarily more complex than an inner join. It is referred to as "complex" simply because SQL is conducting an inner join in addition to gathering a little more information from one or more tables. What is that extra bit of information? We will discover this by looking at the difference between outer and inner joins.

Difference between Inner Join and Outer Join

Inner Join

As you may recall, an inner join is going to return only the rows from the database that match the query. For example, imagine we have the following tables:

```
TEACHERS TABLE              STUDENTS TABLE
id                          student_id   teacher_id
---------------             ------------------------
1                           1            NULL
2                           2            1
3                           3            NULL
```

Note: If you would like to follow along in the terminal, first run `sqlite3` to open the Command Line Shell for SQLite3. Once opened, run the following queries to create the two tables:

```
CREATE TABLE Teachers(id);
CREATE TABLE Students(student_id, teacher_id);
INSERT INTO Teachers VALUES (1);
INSERT INTO Teachers VALUES (2);
INSERT INTO Teachers VALUES (3);
INSERT INTO Students VALUES (1, NULL);
INSERT INTO Students VALUES (2, 1);
INSERT INTO Students VALUES (3, NULL);
```

First, let's look at an inner join:

```
SELECT *
FROM Teachers
INNER JOIN Students
ON Teachers.id = Students.teacher_id;
```

This query returns only the teacher with the `id = 1` because student 2 is in the first teacher's class.

```
id  |  student_id |  teacher_id
--------------------------------
1   |  2          |  1
```

Note: Since we're *joining* tables, running this example SQL command will return a result with both an *id* and a *teacher_id*, even though they are the same.

Outer Join

Outer Joins, on the other hand, will return all of the matching rows AND all of the additional rows from the specified table. Which table/additional rows are determined by the type of outer join. There are three types of outer joins: Left Outer Join, Right Outer Join, and Full Outer Join.

NOTE: SQLite, the database management system that we've been using to explore SQL, does not implement the full SQL standard. Specifically, SQLite does not implement RIGHT OUTER JOIN or FULL OUTER JOIN. However, the concepts underlying these joins are still important to understand (and other databases, like PostgreSQL, do implement them), so you'll need to know about these JOINs even if you won't be using them right away.

Left Outer Join

This is the most common outer join and the one you'll use most often. This returns the normal inner join result and also ***returns all of the rows from the left-most (i.e. first mentioned) table***.

We also make use of the `as` keyword which allows us to specify how our returned data shows up. You'll see how the `as` name helps make the output easier to read.

```
SELECT
  Teachers.id as teacher_id,
  Students.student_id
FROM Teachers
LEFT OUTER JOIN Students
ON Teachers.id = Students.teacher_id;
```

```
teacher_id  |  student_id
--------------------------
1           |  2
2           |  NULL
3           |  NULL
```

Notice that every row from the teacher's table is returned whether there is a corresponding student or not.

Right Outer Join

As you might imagine, this is the same as the Left Outer Join with the minor difference being that it ***returns all of the rows from the right-most (i.e. last-mentioned) table***. Sticking with our example:

```
SELECT
  Teachers.id as teacher_id,
  Students.id as student_id
FROM Teachers
RIGHT OUTER JOIN Students
ON Teachers.id = Students.teacher_id;
```

```
teacher_id     |  student_id
-----------------------------
NULL           |  1
1              |  2
NULL           |  3
```

Full Outer Join

The full ***returns all of the rows from all the tables***.

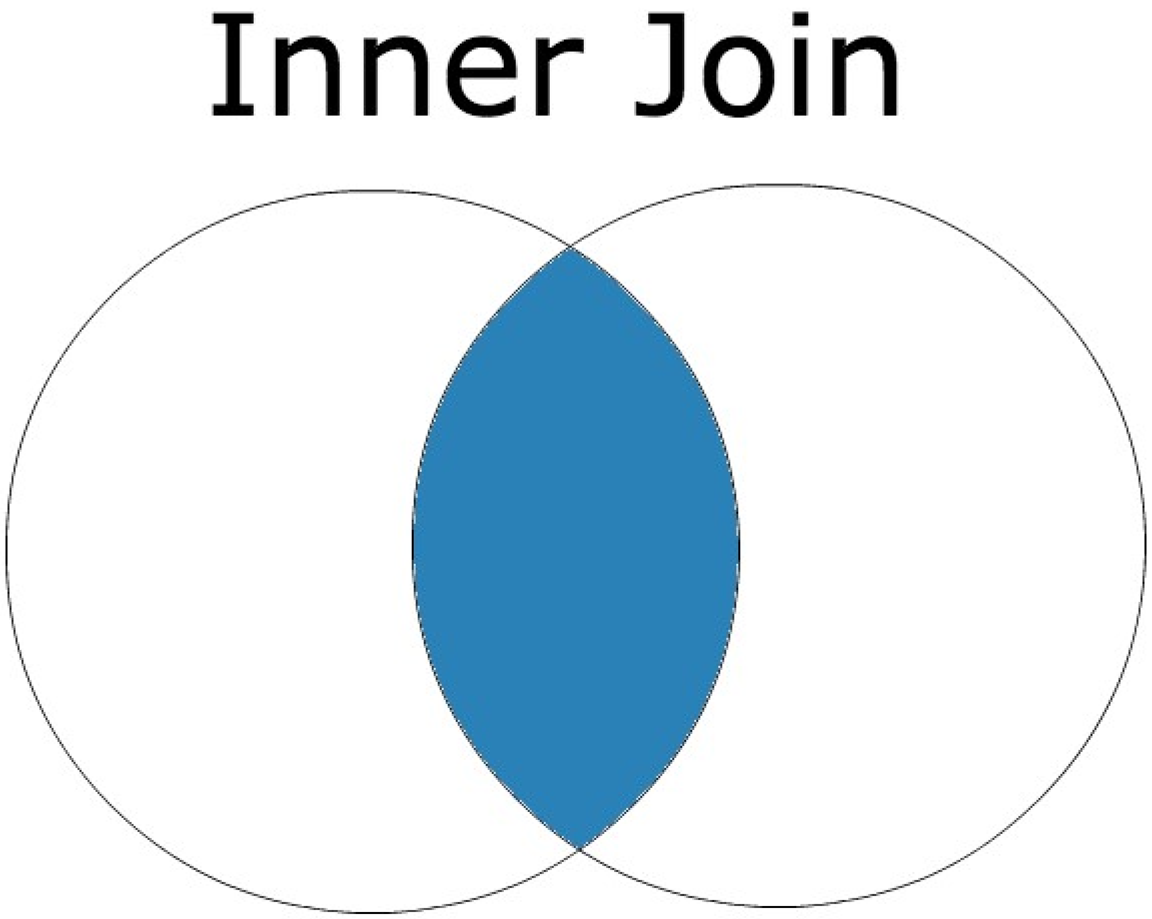
```
SELECT
  Teachers.id as teacher_id,
  Students.id as student_id
FROM Teachers
FULL OUTER JOIN Students
ON Teachers.id = Students.teacher_id;
```

```
teacher_id     |  student_id
-----------------------------
NULL           |  1
1              |  2
NULL           |  3
2              |  NULL
3              |  NULL
```

Venn Diagrams

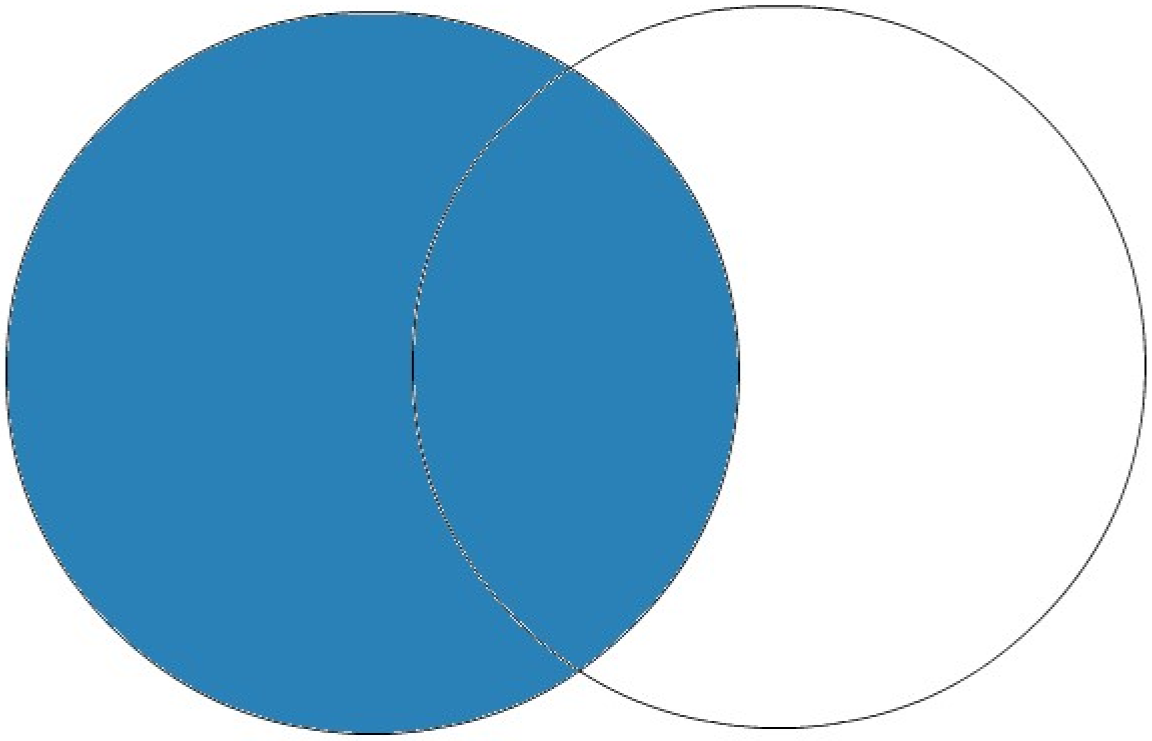
It is helpful to think about our queries as a Venn Diagram. Each table can be represented by a circle.

An Inner Join just returns the overlapping areas of the Venn Diagram.



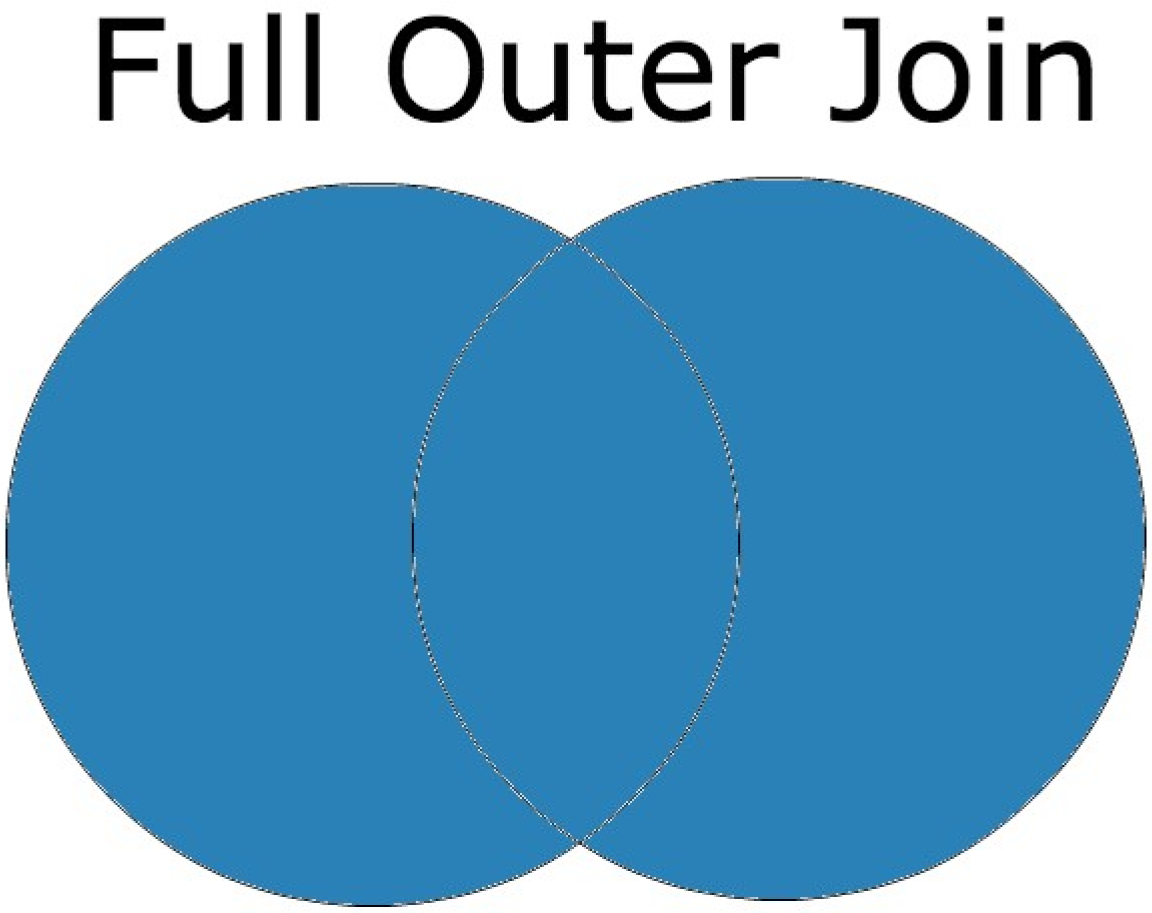
A Left Outer Join returns all the the data from the left circle, and it includes the overlapping information from the right circle.

Left Outer Join



A Right Outer Join returns all the the data from the right circle, and it includes the overlapping information from the left circle.

A Full Outer Join returns all the data from all the tables, including the overlapping data.



Examples

Create Our Students Table

```
CREATE TABLE students (
    id INTEGER PRIMARY KEY,
    name TEXT,
    teacher_id INTEGER);
```

Insert Students

```
INSERT INTO students (name, teacher_id)
    VALUES ("Dave", 1);
INSERT INTO students (name, teacher_id)
    VALUES ("Jessie", 1);
INSERT INTO students (name, teacher_id)
    VALUES ("Bob", 1);
INSERT INTO students (name, teacher_id)
    VALUES ("Sara", 2);
INSERT INTO students (name, teacher_id)
    VALUES ("George",  2);
```

```
INSERT INTO students (name, teacher_id)
    VALUES ("Alexis",  NULL);
```

Students Schema

```
id               name        teacher_id
---------------  ----------  ----------
1                Dave           1
2                Jessie         1
3                Bob            1
4                Sara           2
5                George         2
6                Alexis
```

Create Our Teachers Table

```
CREATE TABLE teachers (
    id INTEGER PRIMARY KEY,
    name TEXT);
```

Insert Into Teachers

```
INSERT INTO teachers (name)
    VALUES ("Joe");
INSERT INTO teachers (name)
    VALUES ("Steven");
INSERT INTO teachers (name)
    VALUES ("Jeff");
```

Teachers Schema

```
id               name
---------------  ---------
1                Joe
2                Steven
3                Jeff
```

Left Outer Join

```
SELECT * FROM Teachers
   LEFT OUTER JOIN Students on Teachers.id = Students.teacher_id;
```

This query will return all of the records in the left table (teachers) regardless if any of those records have a match in the right table (students). It will also return any matching records from the right table. So for our example, it first returns all of the teachers followed by any student that has a `teacher_id`. You can see that Alexis was not returned because her `teacher_id` column is `NULL`.

Results

```
id  teacher_name    id      name     teacher_id
--- ------------   ----    ------    -----------
1     Joe          3       Bob          1
1     Joe          1       Dave         1
1     Joe          2       Jessie       1
2     Steven       5       George       2
2     Steven       4       Sara         2
3     Jeff         NULL    NULL         NULL
```

Right Outer Join

```
SELECT * from Teachers
   RIGHT OUTER JOIN Students on Teachers.id = Students.teacher_id;
```

This query will return all of the records in the right table (students) regardless if any of those records have a match in the left table (teachers). It will also return any matching records from the left table. You can see that all of the students were returned, but this time Jeff was left out.

Results

```
id    teacher_name   id      name     teacher_id
---   ------------  ----    ------    -----------
1       Joe         3       Bob          1
1       Joe         1       Dave         1
1       Joe         2       Jessie       1
2       Steven      5       George       2
2       Steven      4       Sara         2
NULL    NULL        6       Alexis       NULL
```

Full Join

```
SELECT * from Teachers
   FULL OUTER JOIN Students on Teachers.id = Students.teacher_id;
```

This Join can be referred to as a FULL OUTER JOIN or a FULL JOIN. This query will return all of the records from both tables, joining records from the left table (`Teachers`) that match records from the right table (`Students`).

```
id    teacher_name   id      name     teacher_id
---   ------------  ----    ------    -----------
1       Joe          3       Bob         1
1       Joe          1       Dave        1
1       Joe          2       Jessie      1
2       Steven       5       George      2
2       Steven       4       Sara        2
3       Jeff         NULL    NULL        NULL
NULL    NULL         6       Alexis      NULL
```