INFSCI 1022
Database Management Systems

Today's Evil Plan

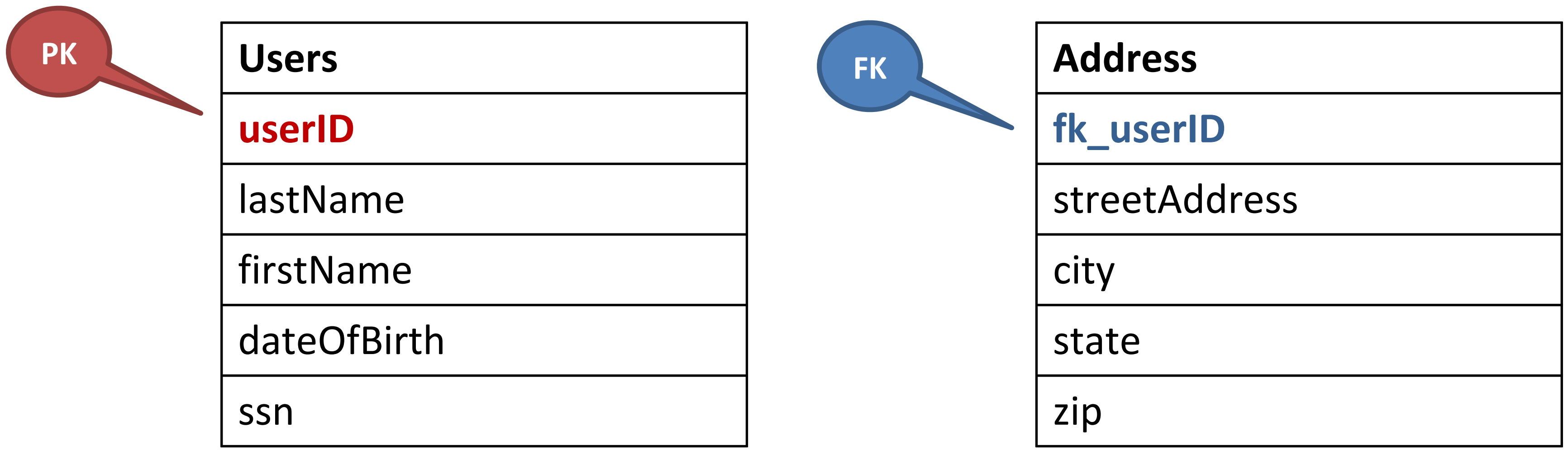
- More on JOINs
- **IN** predicate
- Subqueries and nested queries

More on JOINs

IN predicate

Subqueries and nested queries

INNER JOIN IN A 1:m RELATIONSHIP

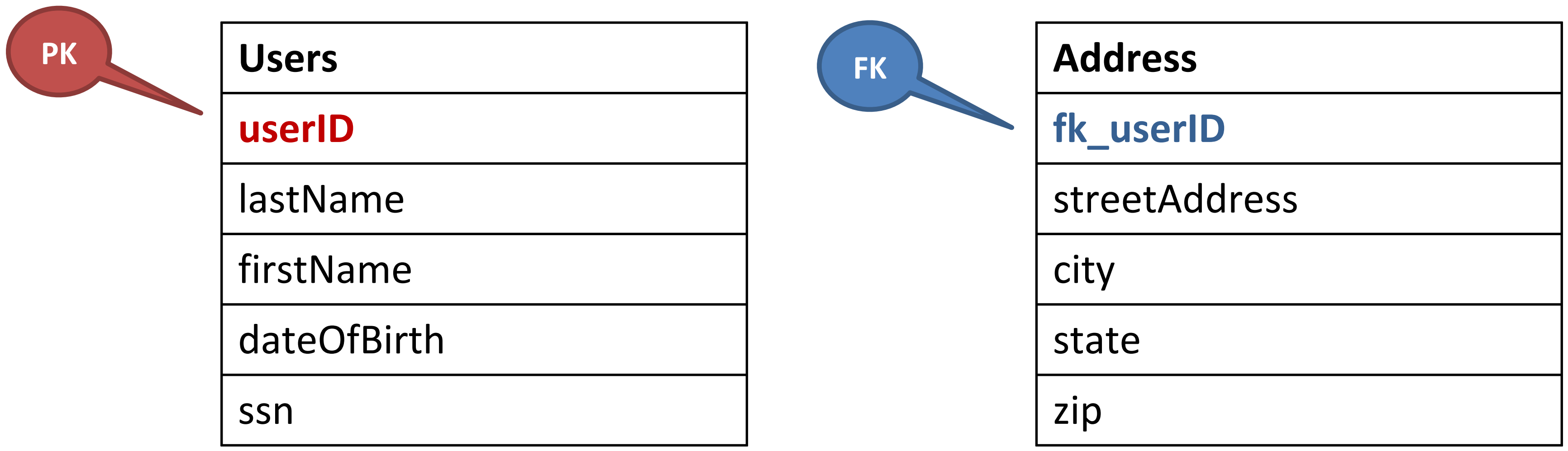


Users
userID
lastName
firstName
dateOfBirth
ssn

Address
fk_userID
streetAddress
city
state
zip

```
SELECT lastName, firstName, streetAddress, city, state, zip
FROM Users u JOIN Address a
ON u.userID = a.fk_userID;
```

INNER JOIN IN A 1:m RELATIONSHIP



Users
userID
lastName
firstName
dateOfBirth
ssn

Address
fk_userID
streetAddress
city
state
zip

SQL JOINS

userID	lastName	firstName	dateOfBirth	ssn
1	Doe	John	04/01/2001	111-11-1111
2	Brown	Michael	01/02/1986	222-22-2222
3	Green	Evelyn	03/14/1976	333-33-3333

addressID	fk_userID	streetAddress	city	state	zip
1335235	1	101 Phillips Avenue	Pittsburgh	PA	15217
5436543	1	325 Hobart Street	Pittsburgh	PA	15217
3675476	3	722 Darlington Avenue	Pittsburgh	PA	15217

SQL JOINS

lastName	firstName	streetAddress	city	state	zip
Doe	John	101 Phillips Avenue	Pittsburgh	PA	15217
Doe	John	325 Hobart Street	Pittsburgh	PA	15217
Green	Evelyn	722 Darlington Avenue	Pittsburgh	PA	15217

Columns from ***Users*** table

Columns from ***Address*** table

SQL JOINS

userID (pk)	lastName	firstName	dateOfBirth	ssn
1	Doe	John	04/01/2001	111-11-1111
2	Brown	Michael	01/02/1986	222-22-2222
3	Green	Evelyn	03/14/1976	333-33-3333

addressID	fk_userID (fk)	streetAddress	city	state	zip
1335235	1	101 Phillips Avenue	Pittsburgh	PA	15217
5436543	1	325 Hobart Street	Pittsburgh	PA	15217
3675476	3	722 Darlington Avenue	Pittsburgh	PA	15217

JOIN – Concatenating Data From Multiple Tables

Fields from ***Users*** table

Fields from ***Address*** table

```
SELECT lastName, firstName, streetAddress, city, state, zip
FROM Users JOIN Address
ON userID = fk_userID;
```

Primary key from **users** table

Foreign key from **addresses** table

INNER JOIN

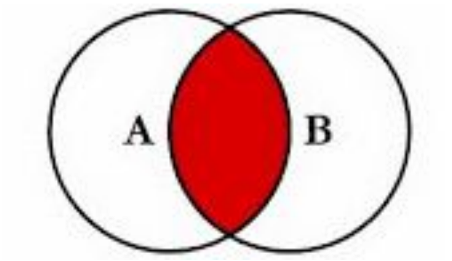
- This join returns rows when there is at least one match in both the tables.
- Inner join is the default join in SQL language.

userID	lastName	firstName	dateOfBirth	ssn
1	Doe	John	04/01/2001	111-11-1111
2	Brown	Michael	01/02/1986	222-22-2222
3	Green	Evelyn	03/14/1976	333-33-3333
4	Smith	Jake	05/02/1922	444-44-4444

addressID	fk_userID	streetAddress	city	state	zip
1335235	1	101 Phillips Avenue	Pittsburgh	PA	15217
5436543	1	325 Hobart Street	Pittsburgh	PA	15217
3675476	3	722 Darlington Avenue	Pittsburgh	PA	15217

INNER JOIN (same as JOIN)

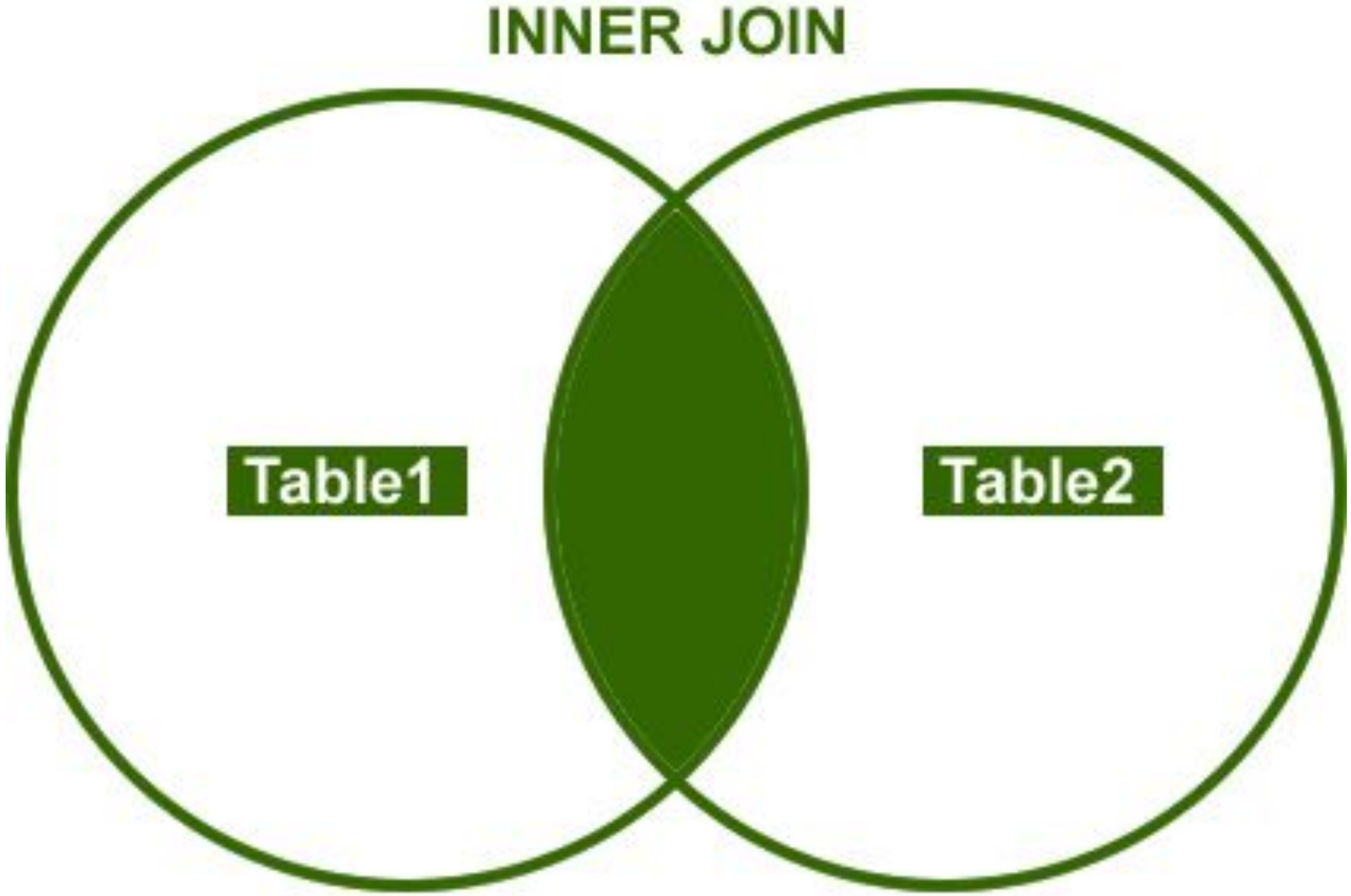
The **INNER JOIN** keyword selects all rows from both tables as long as there is a match between the columns in both tables.



```
SELECT column_name(s)
FROM table1
[INNER] JOIN table2
ON table1.column_name=table2.column_name;
```

Adapted from Robert Perkoski's slides and from
http://www.codinghorror.com/blog/2007/10/a-visual-explanation-of-sql-joins.html

http://blog.sqlauthority.com/2009/04/13/sql-server-introduction-to-joins-basic-of-joins/

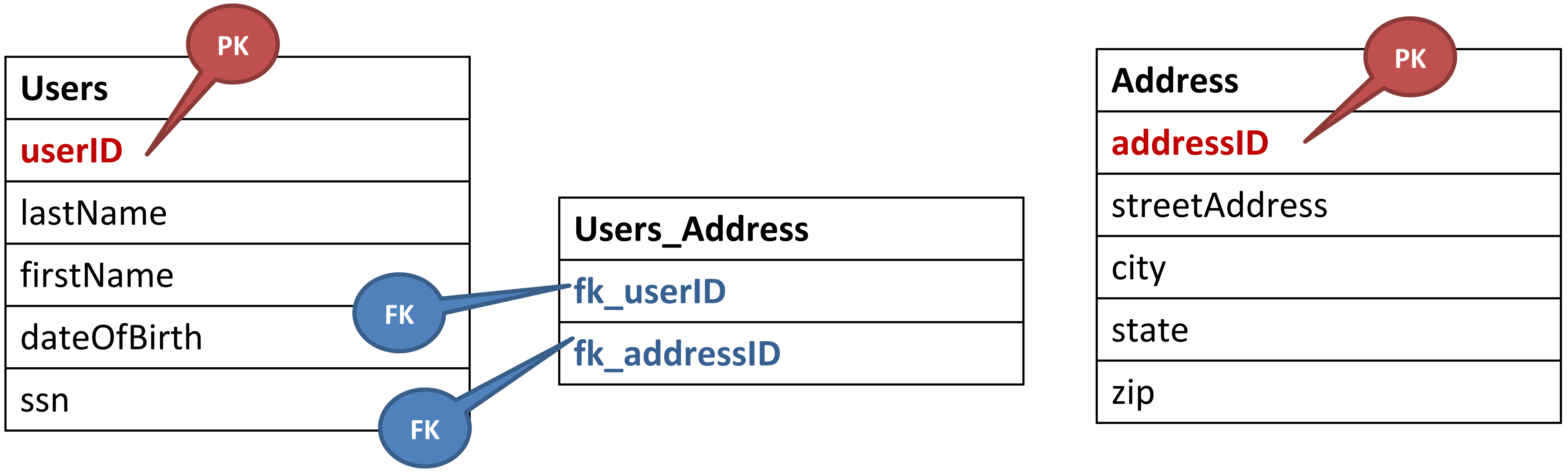


AIN’T NO DIFFERENCE

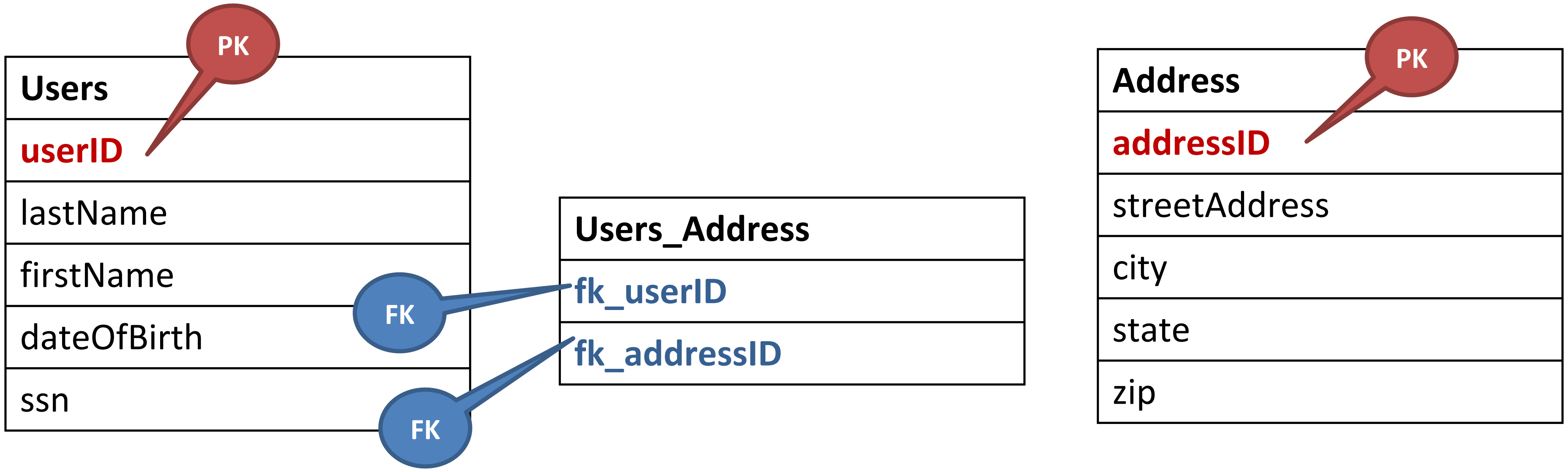
```
SELECT lastName, firstName, streetAddress, city, state, zip
FROM Users INNER JOIN Address
ON userID = fk_userID;

SELECT lastName, firstName, streetAddress, city, state, zip
FROM Users JOIN Address
ON userID = fk_userID;
```

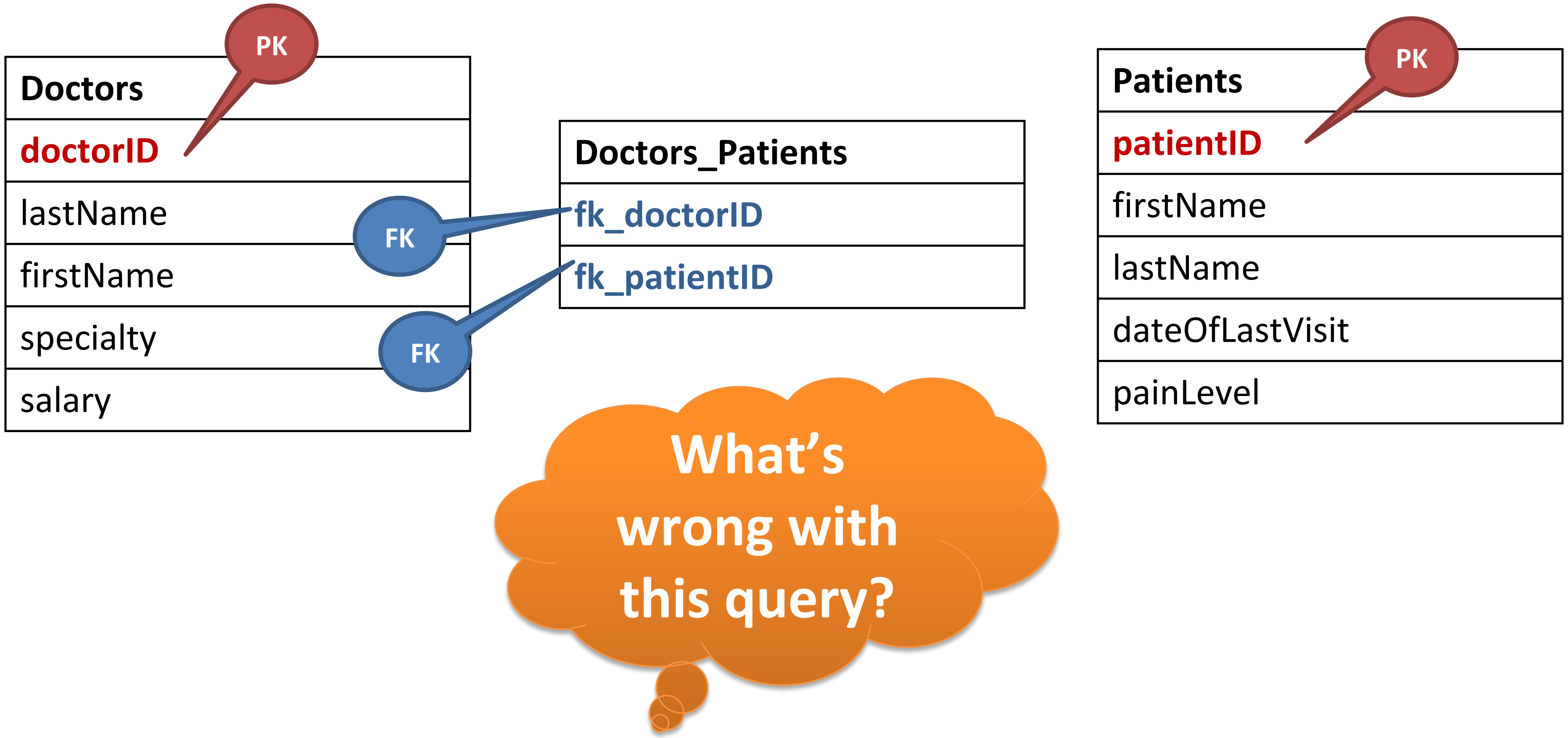
INNER JOIN IN A m:n RELATIONSHIP



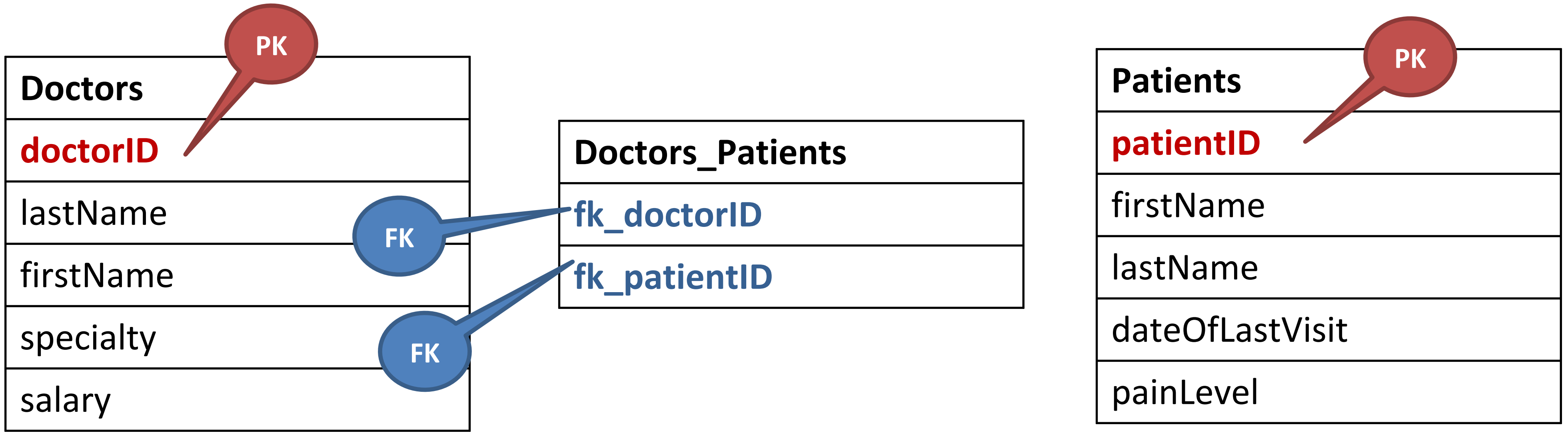
INNER JOIN IN A m:n RELATIONSHIP



```
SELECT lastName, firstName, streetAddress, city, state, zip
FROM Users JOIN Users_Address ON userID = fk_userID
JOIN Address ON fk_addressID = addressID ;
```



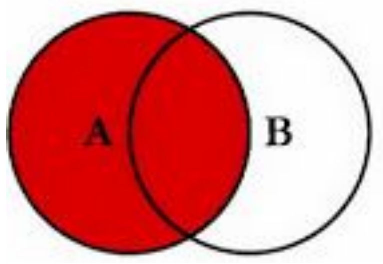
```
SELECT lastName, firstName, dateOfLastVisit, painLevel
FROM Doctors JOIN Doctors_Patients ON doctorID = fk_doctorID
JOIN Patients ON fk_patientID = patientID;
```



```
SELECT d.lastName AS doctorLastName, d.firstName AS doctorFirstName,
p.lastName AS patientLastName, p.firstName AS patientFirstName,
dateOfLastVisit, painLevel
FROM Doctors d JOIN Doctors_Patients dp ON doctorID = fk_doctorID
JOIN Patients p ON fk_patientID = patientID;
```

LEFT JOIN (same as LEFT OUTER JOIN)

The **LEFT JOIN** keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.



```
SELECT column_name(s)
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column_name=table2.column_name;
```

Note: **RIGHT JOIN** works in similar fashion but in reverse

Adapted from Robert Perkoski's slides and from
http://www.codinghorror.com/blog/2007/10/a-visual-explanation-of-sql-joins.html

LEFT [OUTER] JOIN

userID	lastName	firstName	dateOfBirth	ssn
1	Doe	John	04/01/2001	111-11-1111
2	Brown	Michael	01/02/1986	222-22-2222
3	Green	Evelyn	03/14/1976	333-33-3333

addressID	fk_userID	streetAddress	city	state	zip
1335235	1	101 Phillips Avenue	Pittsburgh	PA	15217
5436543	1	325 Hobart Street	Pittsburgh	PA	15217
3675476	3	722 Darlington Avenue	Pittsburgh	PA	15217

Note that user with ***userID = 2*** does not have an address record in table ***Addresses***

LEFT [OUTER] JOIN

```
SELECT lastName, firstName, streetAddress, city, state, zip
FROM Users LEFT JOIN Address
ON userID = fk_userID;
```

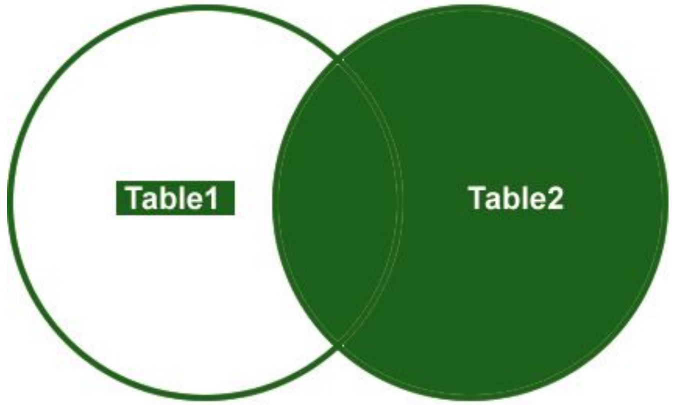
LEFT [OUTER] JOIN

lastName	**firstName**	**streetAddress**	**city**	**state**	**zip**
Doe	John	101 Phillips Avenue	Pittsburgh	PA	15217
Brown	Michael	**NULL**	**NULL**	**NULL**	**NULL**
Green	Evelyn	325 Hobart Street	Pittsburgh	PA	15217
Green	Evelyn	722 Darlington Avenue	Pittsburgh	PA	15217

Columns from ***Users*** table

Columns from ***Addresses*** table

RIGHT OUTER JOIN



The **RIGHT OUTER JOIN** returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.

```
SELECT column_name(s)
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column_name=table2.column_name;
```

RIGHT [OUTER] JOIN

```
SELECT lastName, firstName, streetAddress, city, state, zip
FROM Address RIGHT OUTER JOIN User
ON fk_userID = userID;
```

Note that **ORDER MATTERS!!!**

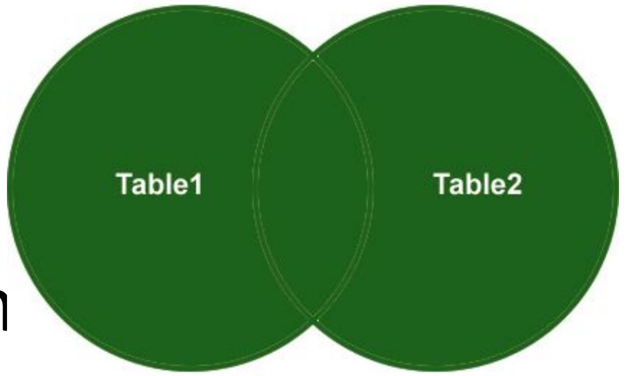
RIGHT [OUTER] JOIN

lastName	**firstName**	**streetAddress**	**city**	**state**	**zip**
Doe	John	101 Phillips Avenue	Pittsburgh	PA	15217
Brown	Michael	**NULL**	**NULL**	**NULL**	**NULL**
Green	Evelyn	325 Hobart Street	Pittsburgh	PA	15217
Green	Evelyn	722 Darlington Avenue	Pittsburgh	PA	15217

These results are the same as from the LEFT OUTER JOIN query –
we simply reversed the order of tables in the RIGHT OUTER JOIN query.

Note: There is almost never a good reason to use RIGHT OUTER JOIN

FULL OUTER JOIN



The **FULL OUTER JOIN** combines left outer join and right outer join. It returns row from either table when the conditions are met and returns null value when there is no match.

SELECT *column_name(s)*
FROM *table1*
FULL OUTER JOIN *table2*
ON *table1.column_name=table2.column_name*;

More on JOINs

IN predicate

Subqueries and nested queries

IN Predicate

- Limits results to a set of rows where a specified value matches any value in a subquery or a **list**.

- Ex: WHERE lastName **IN** (‘Smith’, ‘Jones’, ‘Brown’, ‘Doe’);
- Ex: WHERE accountNumber **IN** (1, 2, 4, 7);

More on JOINs

IN predicate

Subqueries and nested queries

Subqueries

- A Subquery or Inner query or Nested query is a query within another SQL query and is **usually** embedded within the WHERE clause.
- A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.

http://www.tutorialspoint.com/sql/sql-sub-queries.htm

Subqueries

Parent Query

```
SELECT * FROM City
WHERE CountryCode IN
(SELECT Code FROM Country WHERE Name = 'Afghanistan');
```

Subquery

Subquery Rules!

- Subqueries must be enclosed within parentheses.
- A subquery can have only **one column** in the SELECT clause
- An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY.
- Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- The BETWEEN operator cannot be used with a subquery; however, the BETWEEN operator can be used within the subquery.

Subqueries must be enclosed within parentheses.

```
SELECT * FROM City
WHERE CountryCode IN
(SELECT Code FROM Country WHERE Name = 'Afghanistan');
```

A subquery can have only one column in the SELECT clause

```
SELECT * FROM City
WHERE CountryCode IN
(SELECT Code FROM Country WHERE Name = 'Afghanistan');
```

A subquery can have only one column in the SELECT clause

```
SELECT a.Name,
(SELECT AVG(b.Population) FROM City b WHERE a.Code = b.CountryCode)
FROM Country a;
```

Subquery

An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY.

```
SELECT * FROM City
WHERE CountryCode IN
(SELECT Code FROM Country WHERE Name = 'Afghanistan')
ORDER BY City.Name;
```

Note that subquery does not have an ORDER BY clause

Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.

```
SELECT * FROM City
WHERE CountryCode IN
(SELECT Code FROM Country WHERE Name = 'Afghanistan');
```

Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.

```
SELECT * FROM City
WHERE CountryCode =
(SELECT Code FROM Country WHERE Name = 'Afghanistan' AND
District = 'Herat');
```

This subquery will return only one value

Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.

SELECT * FROM City

WHERE CountryCode **=**

(SELECT Code FROM Country WHERE Name = 'Afghanistan' **AND District = 'Herat'**);

This subquery will return only one value