IT125 SQL: MULTI-TABLE QUERIES

Bill Barry

TONIGHT

- Understand the concept of a "join" and the most common types of joins
- Learn how to write SQL queries that bring data together from two tables
- Extend that knowledge to data from *more* than two tables
- Use knowledge from single-table queries; all of that applies here, too!

JOIN CONCEPTS

WHAT IS A JOIN?

- Remember that exercise you did where you took data from one table and traced it through to other tables, collecting related information?
- That was a manual join you did
- You don't want to ever have to do that again, right?
- SQL will do that work for you, using SELECT's JOIN syntax

Hint: having an EER in front of you is helpful (almost *required*)

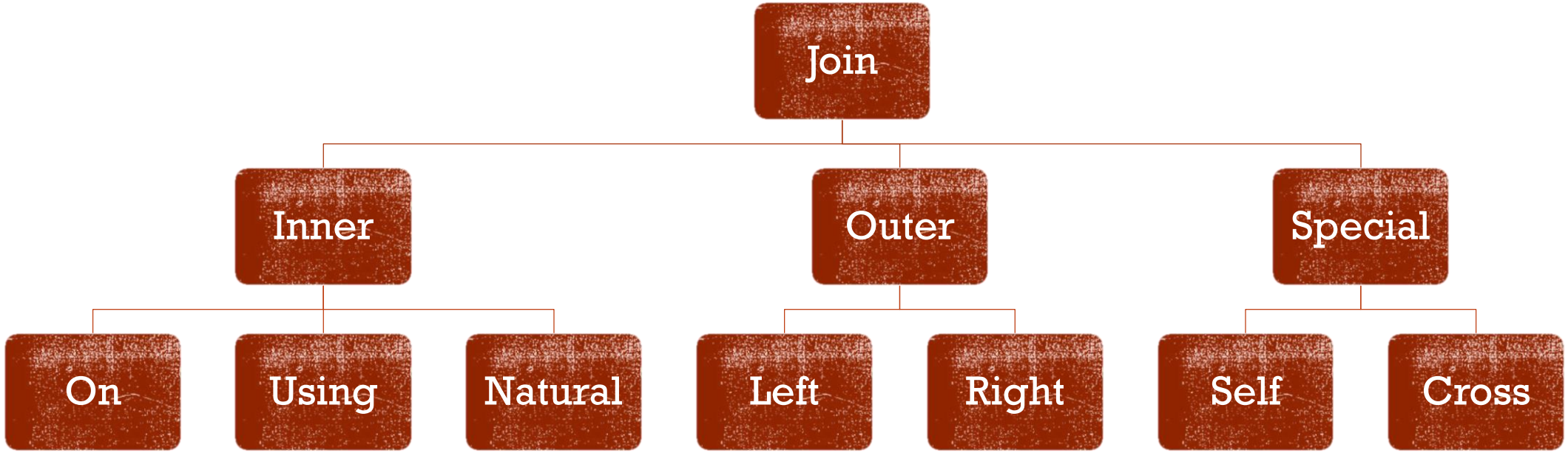
WHAT IS A JOIN? MEMBER/DONATION EXAMPLE

MemberId*	MemberName	MemberPhone
1	Pat Johnson	206-555-1234
2	Lupe Valdez	206-555-4321
3	Chris Sanada	425-555-2345

MemberId*	DonationDate*	DonationAmt
1	5/15/16	$25
2	6/1/16	$15
1	7/3/16	$10
3	7/3/16	$30

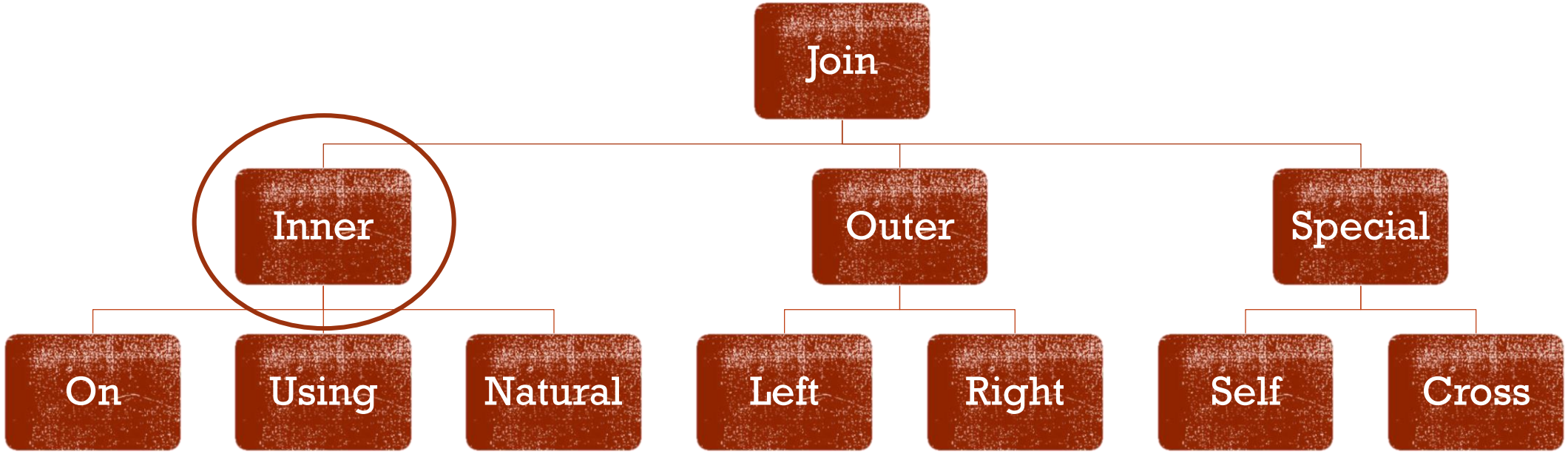
MemberId	MemberName	DonationDate	DonationAmt
1	Pat Johnson	5/15/16	$25
1	Pat Johnson	7/3/16	$10
2	Lupe Valdez	6/1/16	$15
3	Chris Sanada	7/3/16	$30

JOIN HIERARCHY



WRITING JOINS IN SQL

JOIN HIERARCHY



WHAT IS AN INNER JOIN? MEMBER/DONATION EXAMPLE

MemberId*	MemberName	MemberPhone
1	Pat Johnson	206-555-1234
2	Lupe Valdez	206-555-4321
3	Chris Sanada	425-555-2345
4	Dana Smith	206-555-7654

MemberId*	DonationDate*	DonationAmt
1	5/15/16	$25
2	6/1/16	$15
1	7/3/16	$10
3	7/3/16	$30
52	6/15/16	$50

MemberId	MemberName	DonationDate	DonationAmt
1	Pat Johnson	5/15/16	$25
1	Pat Johnson	7/3/16	$10
2	Lupe Valdez	6/1/16	$15
3	Chris Sanada	7/3/16	$30

EXAMPLE INTRO: PET ADOPTION

- You don't have this database; we'll create it together in a future lesson

Customer

CustId	CustName	CustPhone	CustBalance
1	Judd Jetson	2068881414	34.45
2	Harriett Hanson	4258882626	0.00
3	Betty Beaumont	2068884747	75.00

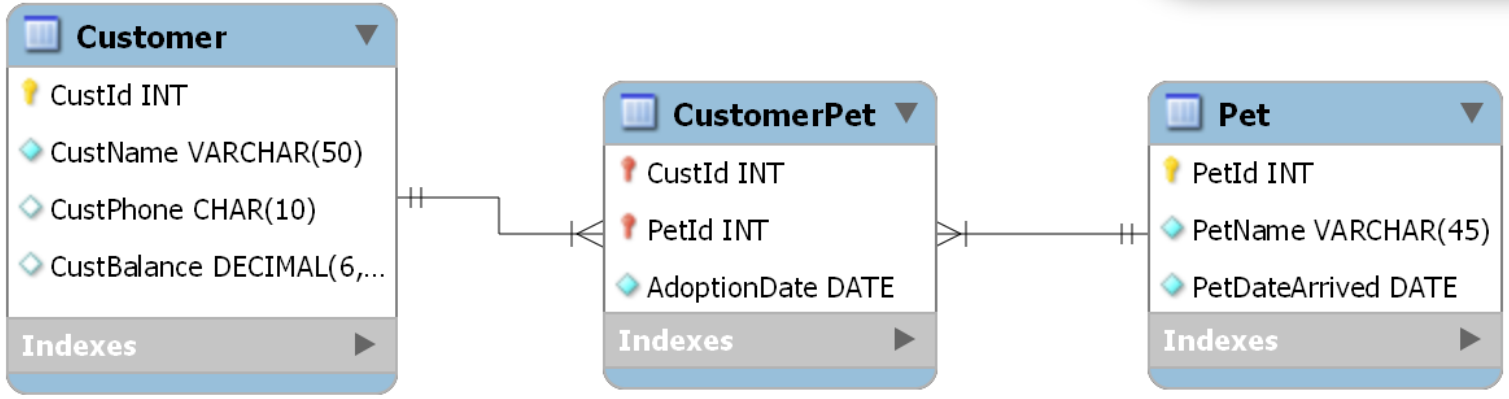
CustomerPet

CustId	PetId	AdoptionDate
2	2	2017-11-03
3	1	2017-11-06

Pet

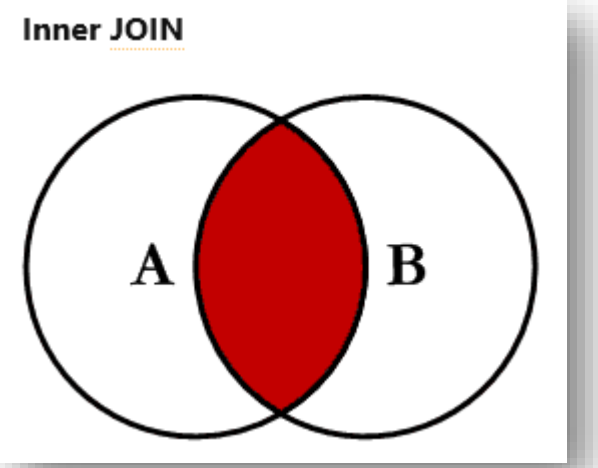
PetId	PetName	PetDateArrived
1	Fluffy	2017-10-13
2	Lucky	2017-10-07
3	Duke	2017-10-31

Remember that what makes JOINs possible is related *data*, not *schema*, per se. That said, look to PK/FK combinations when JOINing; supporting data will be there

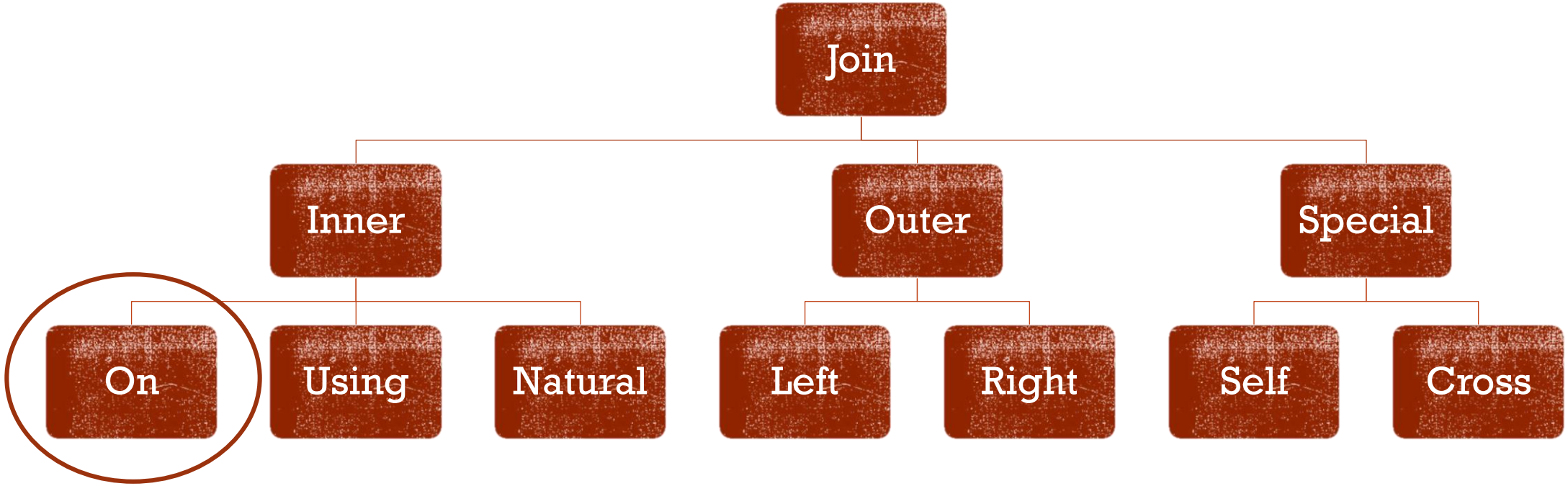


COMMON JOIN TYPES: INNER

- The "Inner Join" is by far the most common type of join
- As shown at right, you are interested in *data that matches up* between the two tables
- An AP DB example:
  - I want a list of Invoices and their associated Vendors
  - Invoices with no associated Vendor? They won't be there!
  - Vendors who have no associated Invoices? They won't be there!



JOIN HIERARCHY



INNER JOIN SYNTAX

- There are two major approaches:
  - Put the join condition in the FROM clause ("explicit") ← preferred
  - Put the join condition in the WHERE clause ("implicit")
- Let's start with the *explicit* syntax:
  - ```
    SELECT select_list
    FROM table_1
        [INNER] JOIN table_2
            ON join_condition;
    ```
- Join conditions are typically:
  - `table1column = table2column`
- Important: if there are ambiguous column names, qualify them: table.column


    ```

INNER JOIN SYNTAX ZOOM-IN

```
SELECT select_list
FROM table_1
    [INNER] JOIN table_2
        ON join_condition
[WHERE condition]
[ORDER BY column1 [, column2]..
```

Can draw on data from either of the tables involved

Should focus on PK/FK columns that relate the two tables

Can draw on data from either of the tables involved

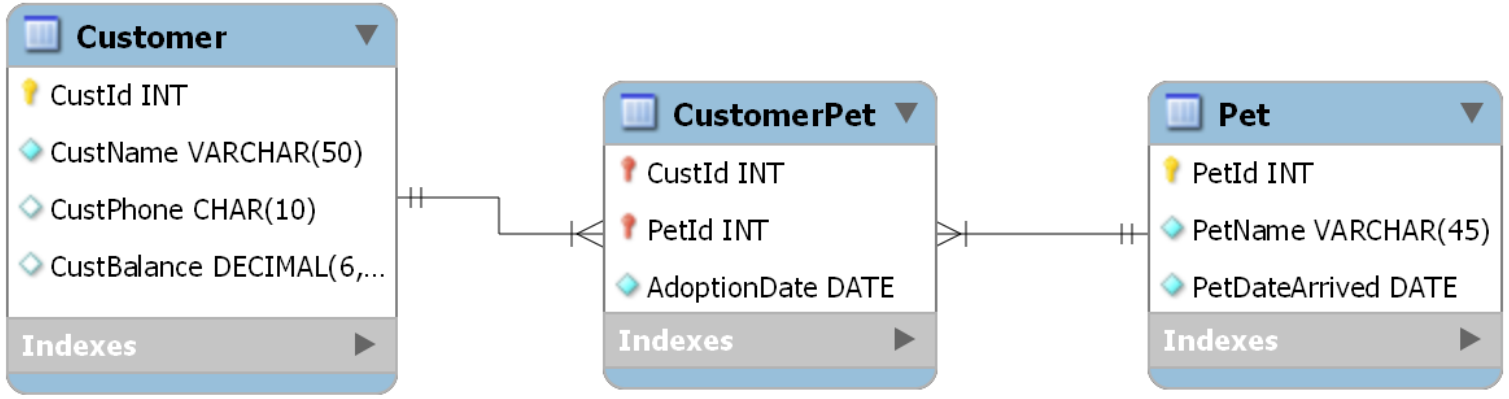
Can draw on data from either of the tables involved

INNER JOIN "ON" EXAMPLE

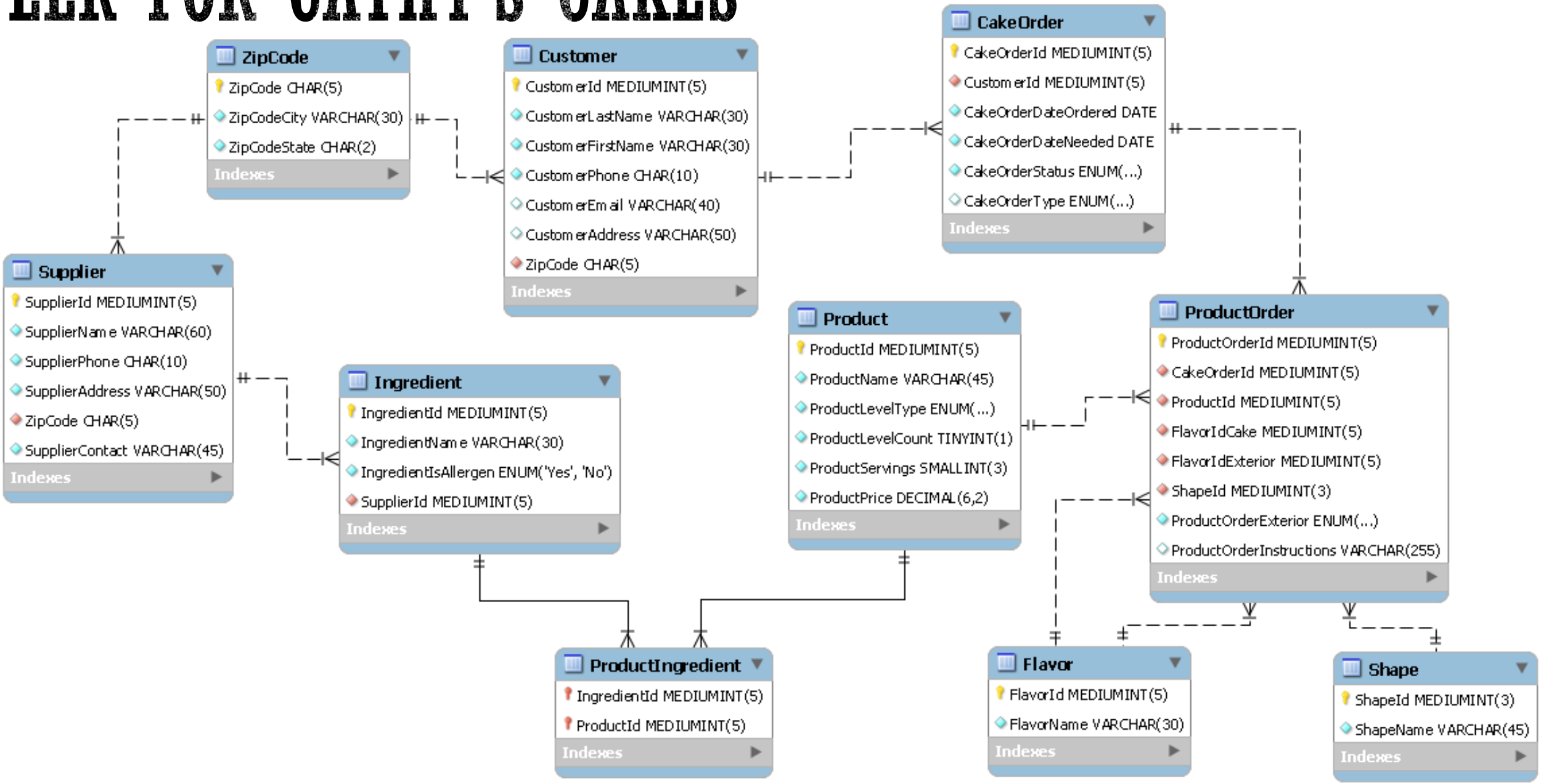
- Get a list of Pets (by name) and when they were adopted
- SELECT PetName, AdoptionDate
  FROM Pet JOIN CustomerPet ON Pet.PetId = CustomerPet.PetId;

PetName	AdoptionDate
Fluffy	2017-11-06
Lucky	2017-11-03

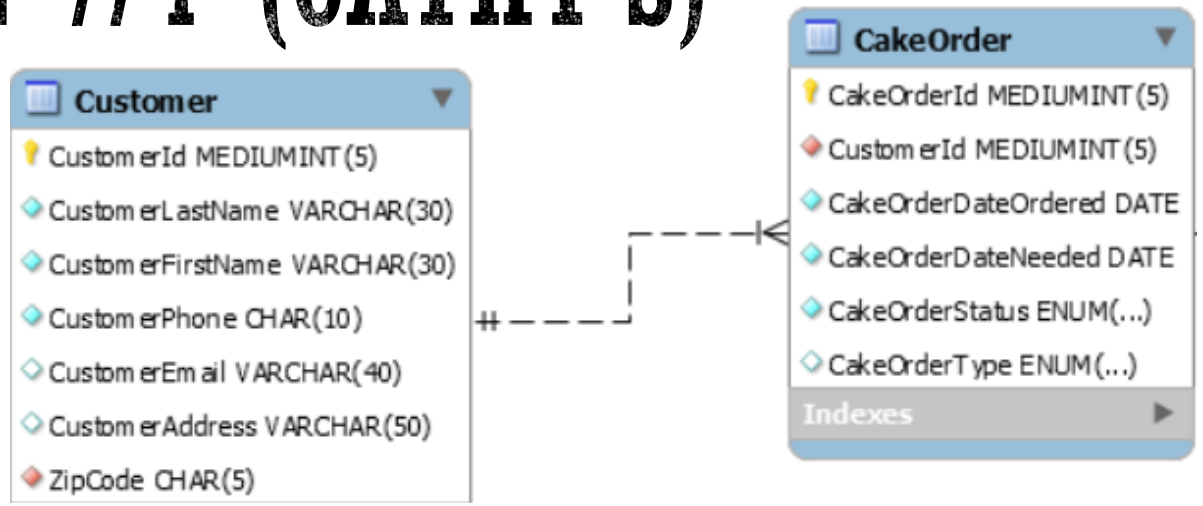
INNER JOIN means that only *adopted* pets will be shown



EER FOR CATHY'S CAKES

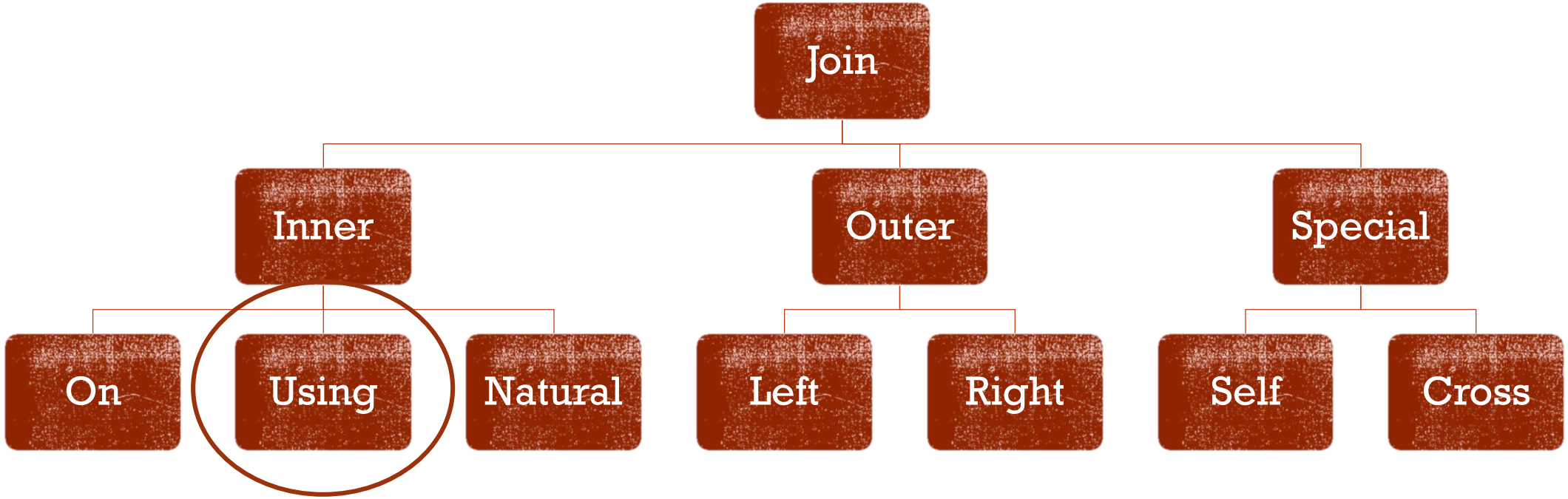


INNER JOIN PRACTICE #1 (CATHY'S)



- Inner Join syntax:
  - ```
    SELECT select_list
    FROM table_1
      [INNER] JOIN table_2
        ON table1col = table2col
    ```
- Get a list of customers
  (full name in "last, first" order) and orders they've placed (id, order date, type)
- Give friendly names to each column in the results list
- Sort results by customer name, then by order date (most recent first)
    ```

JOIN HIERARCHY



INNER JOINS: "USING" SYNTAX

- If the column name matches between the two tables, there's a shorter way:
  - ```
    SELECT select_list
    FROM table_1
        [INNER] JOIN table_2
            USING (common_field_name);
    ```

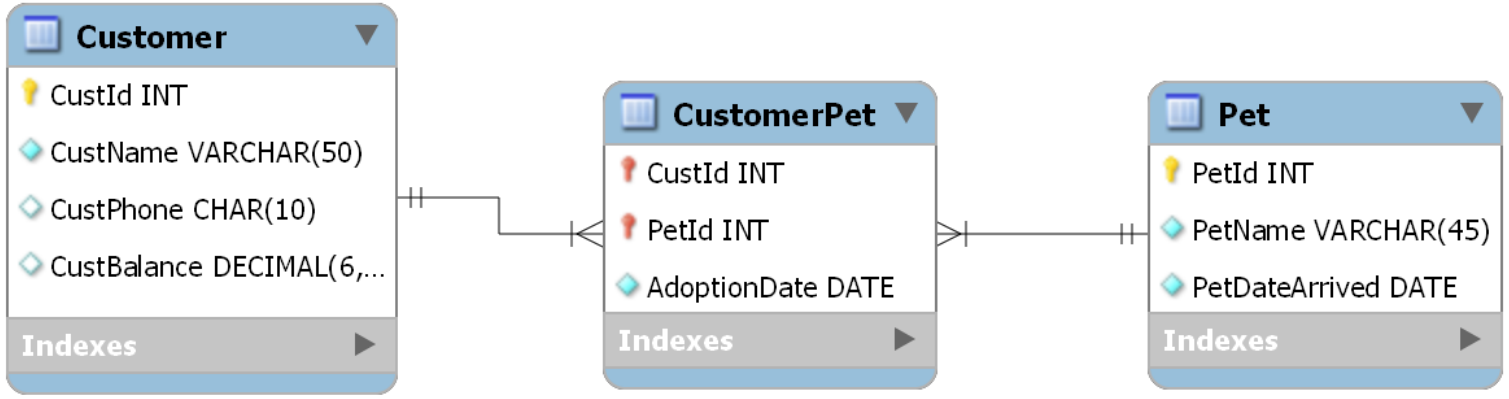
- Hint: the parentheses aren't optional; you'll often forget these!
    ```

INNER JOIN "USING" EXAMPLE

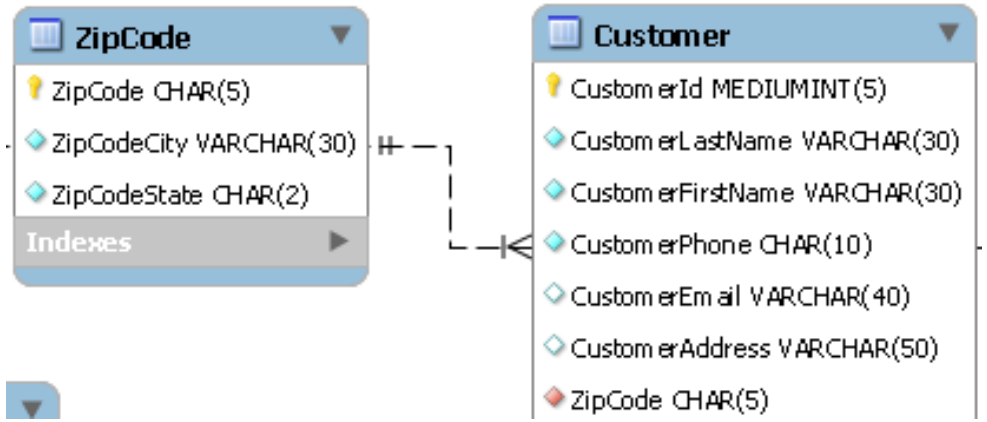
- Get a list of Pets (by name) and when they were adopted
- SELECT PetName, AdoptionDate
  FROM Pet JOIN CustomerPet USING (PetId);

PetName	AdoptionDate
Fluffy	2017-11-06
Lucky	2017-11-03

USING is possible here only because of the common column name



INNER JOIN "USING" PRACTICE #2



- Inner Join "Using" syntax:
  - ```
    SELECT select_list
    FROM table_1
        [INNER] JOIN table_2
            USING (common_field_name);
    ```
- Get a list of customers who live in Bellevue, Washington
- Include the customer's name and phone number, along with the city, state, and zip
- Sort by zip code, then by customer last name, then first name
- Hints:
  - Don't look up IDs to do this; use the information provided. Same in projects!
  - If columns are ambiguous (exist in more than one table), *qualify* the column name by prepending the name of the table (e.g., *table.column*)
    ```

TABLE ALIASES

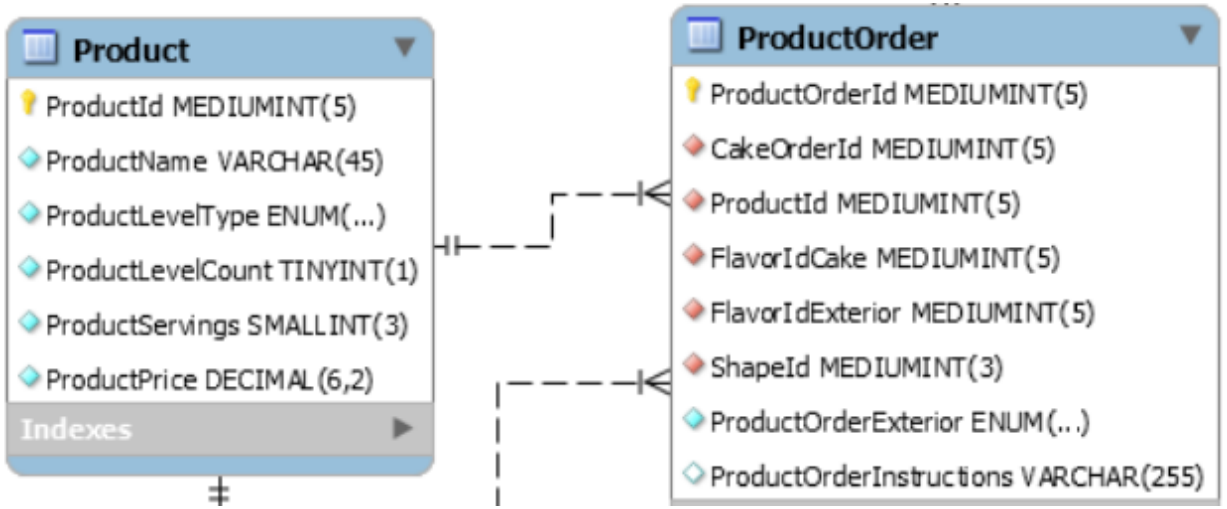
- Sometimes it gets crazy prepending table names everywhere. SQL provides a way for you to give an alias to a table. Usually we like short ones, often one letter
- To do this, just place the alias after the table name
  - ```
    SELECT select_list
    FROM table_1 [alias1]
        [INNER] JOIN table_2 [alias2]
            ON table1col = table2col;
    ```

Often, you'll see ALL column qualified when table aliases are used; might as well be explicit and not make the reader guess

- Then use the alias everywhere you'd prepend the table name and a dot, *even in the select list on the first line* of the query
- Example:

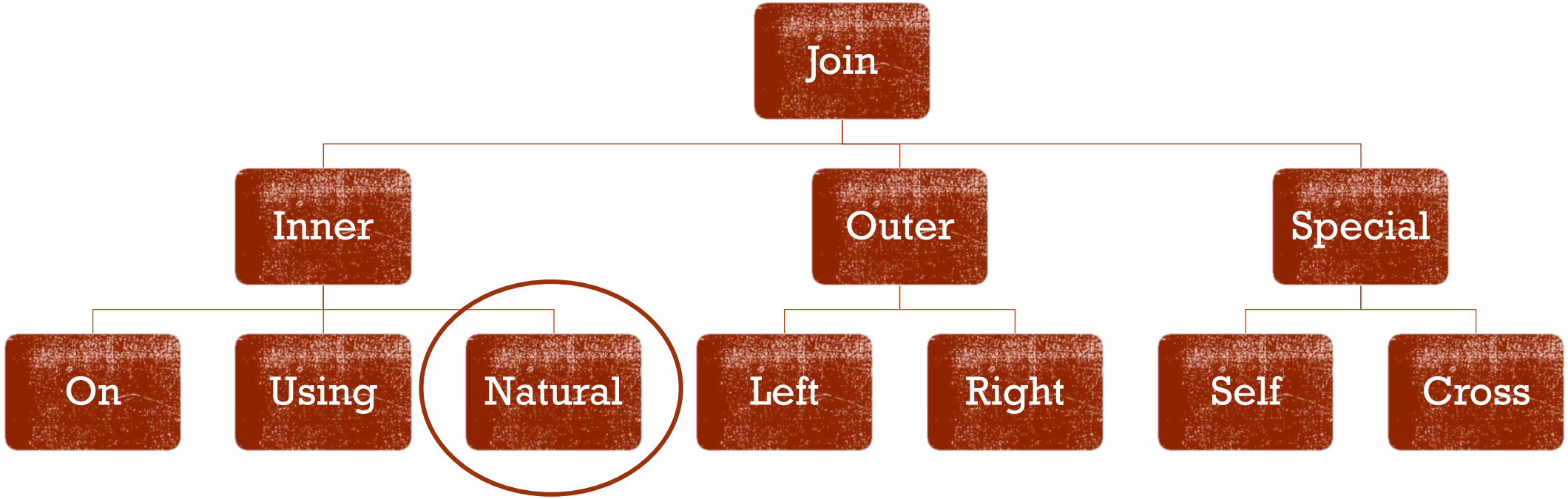
```
SELECT c.cust_id, c.cust_name            # c.cust_name optional but clear
FROM customers c
    INNER JOIN orders o
        ON c.cust_id = o.cust_id;
```
```

INNER JOIN TABLE ALIASES PRACTICE #3



- Inner Join syntax:
  - ```
    SELECT select_list
    FROM table_1 [alias1]
      [INNER] JOIN table_2 [alias2]
        ON table1col = table2col
    ```
- Get a list of Order Id's along with the product ordered (by name) and prices
- Assign and use table aliases for each table
- Qualify all table-specific column references
- Sort by Order ID, then by price (highest first)
    ```

JOIN HIERARCHY



INNER JOINS: "NATURAL" SYNTAX

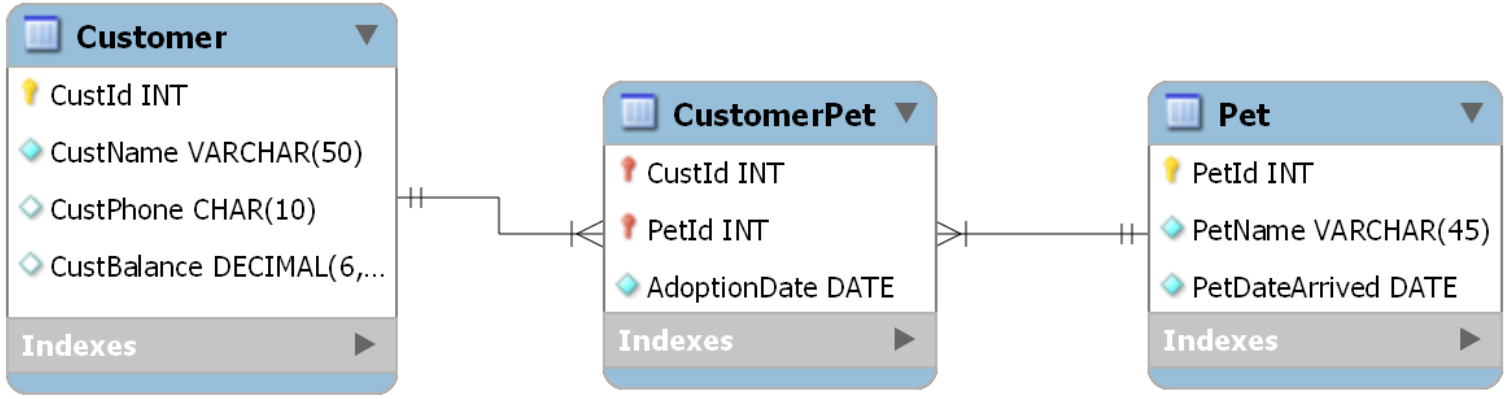
- If you trust SQL to "just do the right thing" and it can figure it out:
  - ```
    SELECT select_list
    FROM table_1
        NATURAL JOIN table_2;
    ```
- You get what you get and hope it's the right thing!
- In practice, this isn't generally used; we don't like the fact the results may change as the database evolves over time
- Important Caveat: "The NATURAL keyword tells the server to match up **any column names** between the two tables, and automatically use ***all* those columns** to resolve the join." That means if there are two matching column names, NATURAL JOIN may fail
  - Imagine a case where each table has a TimeStamp column, for instance
    ```

INNER JOIN "NATURAL" EXAMPLE

- Get a list of Pets (by name) and when they were adopted
- SELECT PetName, AdoptionDate
  FROM Pet NATURAL JOIN CustomerPet;

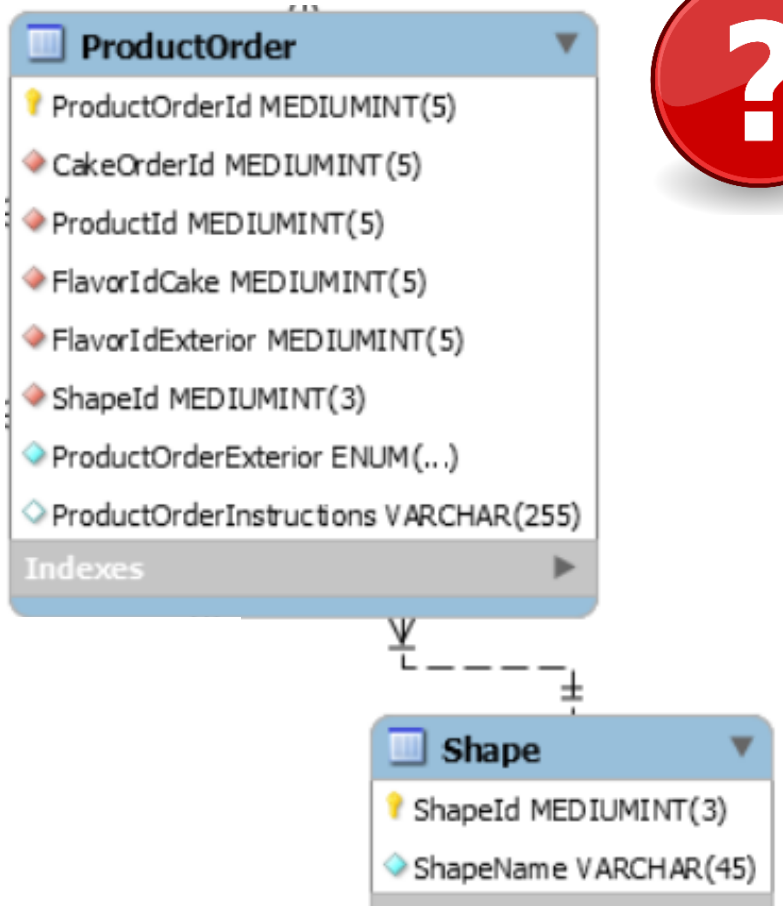
PetName	AdoptionDate
Fluffy	2017-11-06
Lucky	2017-11-03

NATURAL JOIN works here only because the only common column name happens to be right JOIN condition



INNER JOIN "NATURAL" PRACTICE #4

- Inner Join "Natural" syntax:
  - ```
    SELECT select_list
    FROM table_1
        NATURAL JOIN table_2;
    ```
- Find out what combinations of shapes and product exteriors have ever been ordered. Look carefully at the data to see if some additional syntax would be helpful
- Use a NATURAL JOIN to accomplish this
- Sort by exterior, then by shape
- Don't show rows where the shape name is "None" (which means there wasn't a customer choice available for this type of product, e.g., cupcakes or sheet cakes)


    ```

INNER JOINS: IMPLICIT SYNTAX

- Another form of joins you might see is the *implicit* syntax
- This puts the join condition in the WHERE clause
  - This can make it less clear what portion of the clause is *filtering* and what portion *joining*
  - For that reason it's less preferable to the explicit syntax we've learned so far
- Syntax
  - ```
    SELECT select_list
    FROM table1, table2
    WHERE table1col = table2col;
    ```

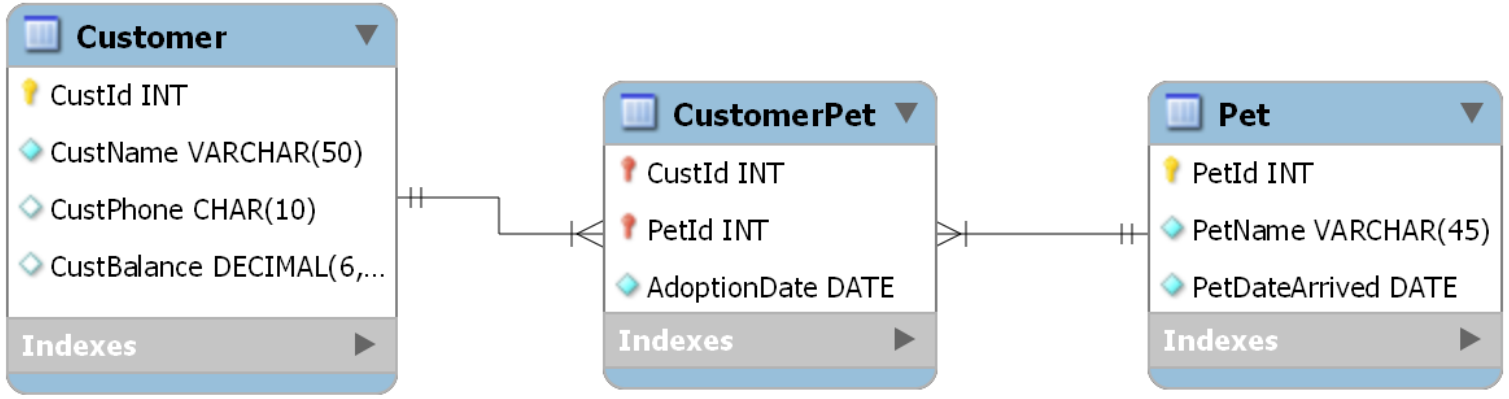
Note: remember to qualify (disambiguate) column names where they are the same in two tables; you may need *table1.columnName* instead of just *columnName*
    ```

INNER JOIN "IMPLICIT" EXAMPLE

- Get a list of Pets (by name) and when they were adopted
- SELECT PetName, AdoptionDate
  FROM PetJOIN CustomerPet
  WHERE Pet.PetId = CustomerPet.PetId;

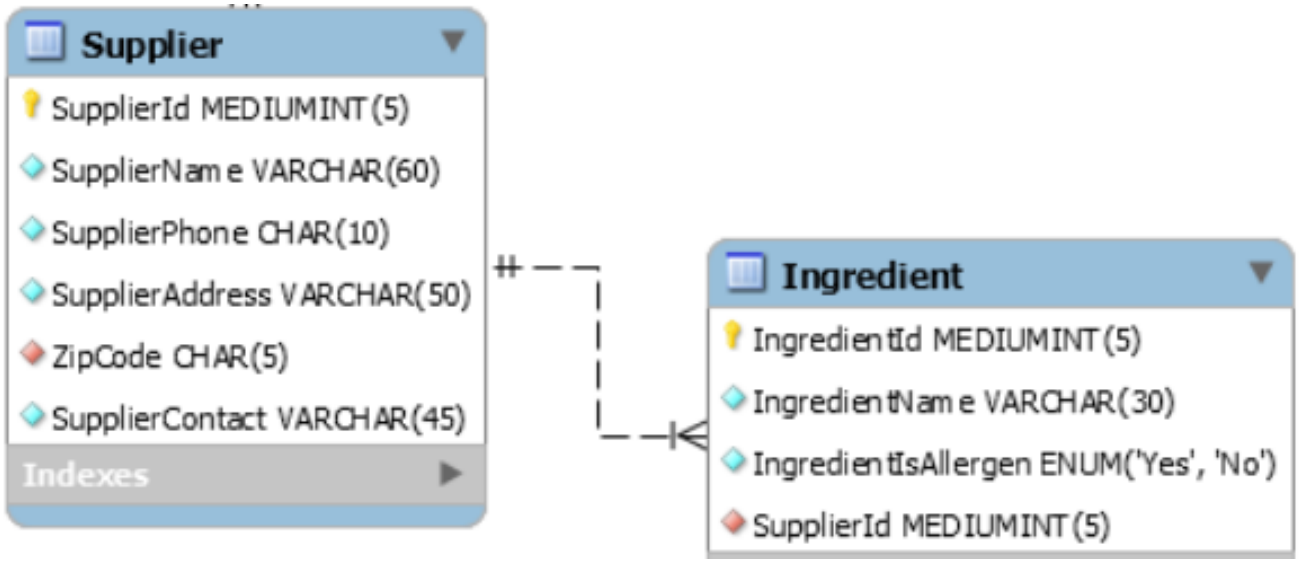
PetName	AdoptionDate
Fluffy	2017-11-06
Lucky	2017-11-03

Note that the JOIN condition has moved from the FROM clause to the WHERE clause

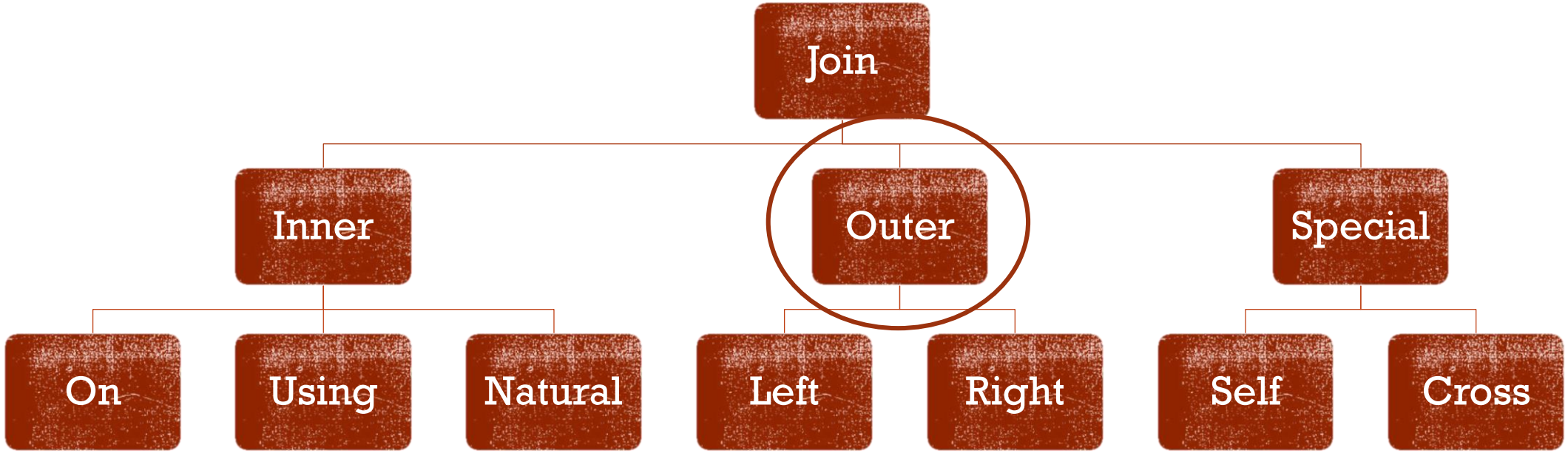


INNER JOIN IMPLICIT SYNTAX PRACTICE #5

- Inner Join implicit syntax:
  - ```
    SELECT select_list
    FROM table1, table2
    WHERE table1col = table2col;
    ```
- Get a list of ingredients (by name) and what supplier from which Cathy buys that ingredient
- Don't include any sugar products
- Sort by ingredient name


    ```

JOIN HIERARCHY



WHAT IS AN OUTER JOIN? MEMBER/DONATION EXAMPLE

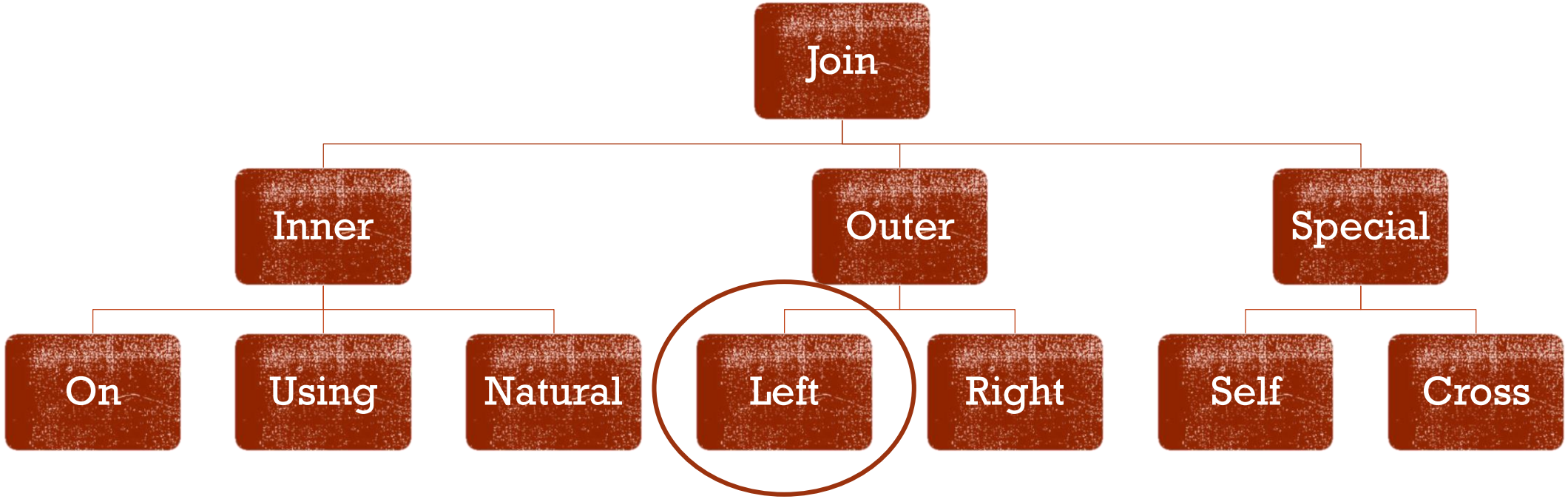
MemberId*	MemberName	MemberPhone
1	Pat Johnson	206-555-1234
2	Lupe Valdez	206-555-4321
3	Chris Sanada	425-555-2345
4	Dana Smith	206-555-7654

MemberId*	DonationDate*	DonationAmt
1	5/15/16	$25
2	6/1/16	$15
1	7/3/16	$10
3	7/3/16	$30
52	6/15/16	$50

MemberId	MemberName	DonationDate	DonationAmt
1	Pat Johnson	5/15/16	$25
1	Pat Johnson	7/3/16	$10
2	Lupe Valdez	6/1/16	$15
3	Chris Sanada	7/3/16	$30
4	Dana Smith	(null)	(null)

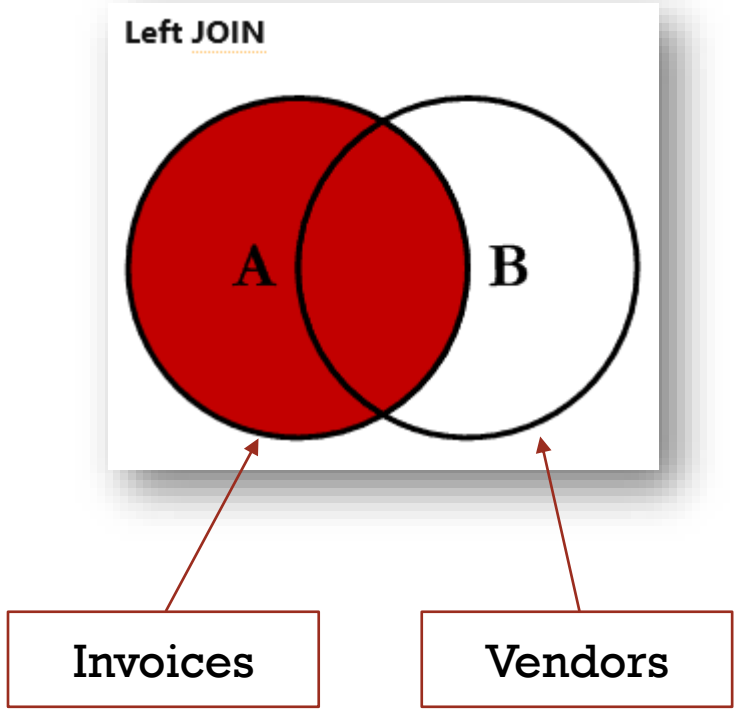
This is a LEFT outer join—why?

JOIN HIERARCHY



COMMON JOIN TYPES: LEFT OUTER JOIN

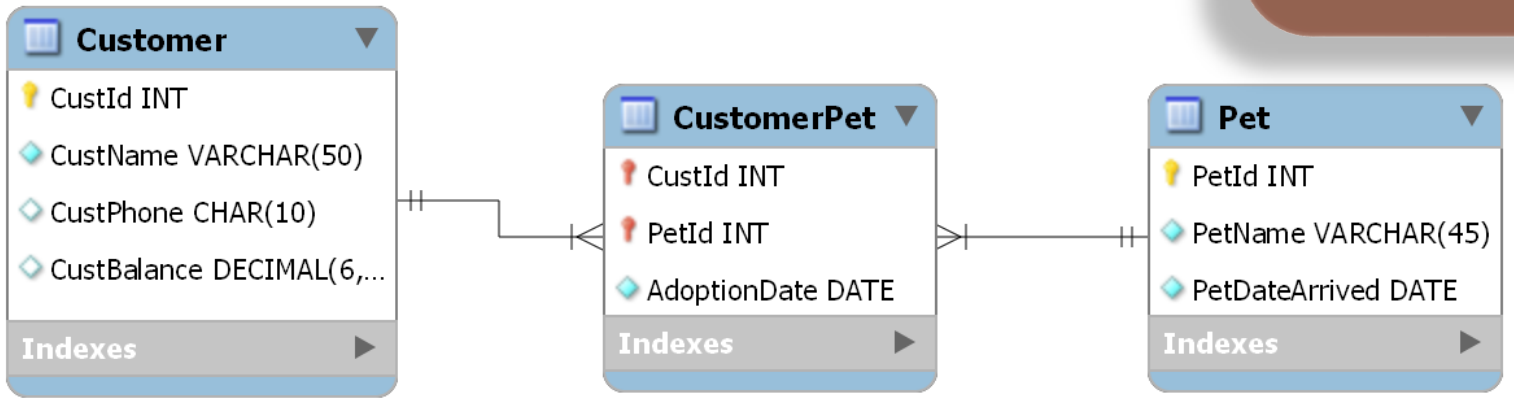
- Another common type is a Left Outer Join
- As shown at right, you are interested in all the data from the first ("left") table, and matching data from the second table
- You shouldn't be surprised to see nulls (unmatched data)
- An AP DB example:
  - I want a list of all Invoices and associated Vendors where they exist
  - Invoices with no associated Vendor? They'll be included
  - Vendors who have no associated Invoices? Don't include them
- Syntax
  -
```
SELECT select_list
FROM table_1
    LEFT [OUTER] JOIN table_2
        ON table1col = table2col;
```



OUTER JOIN "LEFT" EXAMPLE

- Get a list of ALL Pets (by name, regardless of their adoption status) and their associated adoption dates
- `SELECT PetName, AdoptionDate`
  `FROM Pet LEFT JOIN CustomerPet USING (PetId);`

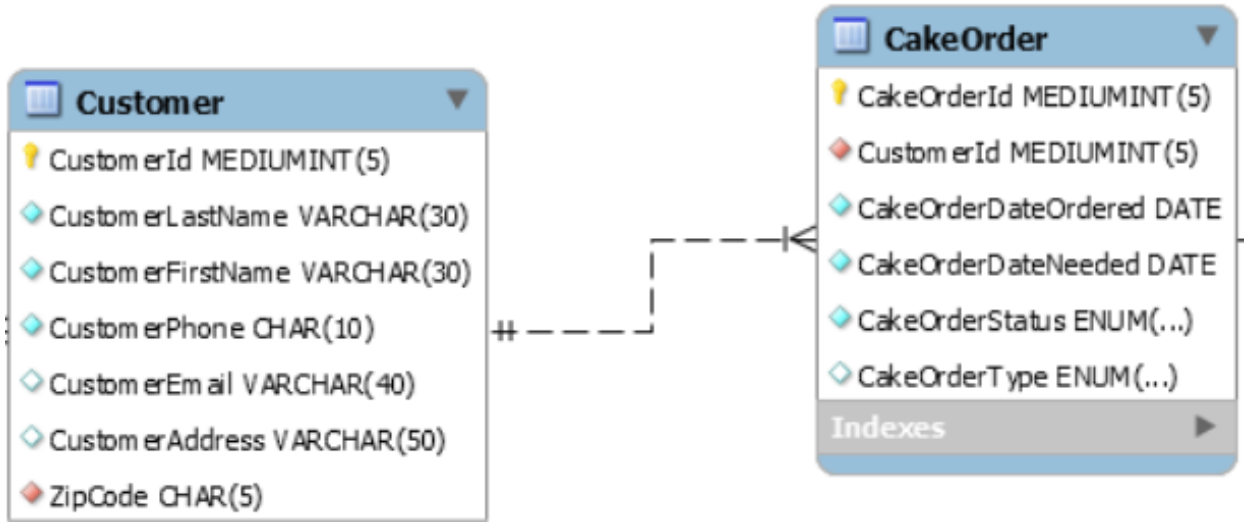
PetName	AdoptionDate
Lucky	2017-11-03
Fluffy	2017-11-06
Duke	NULL



OUTER JOINs *may* yield null data, where there aren't perfect matches; that's perfect, here

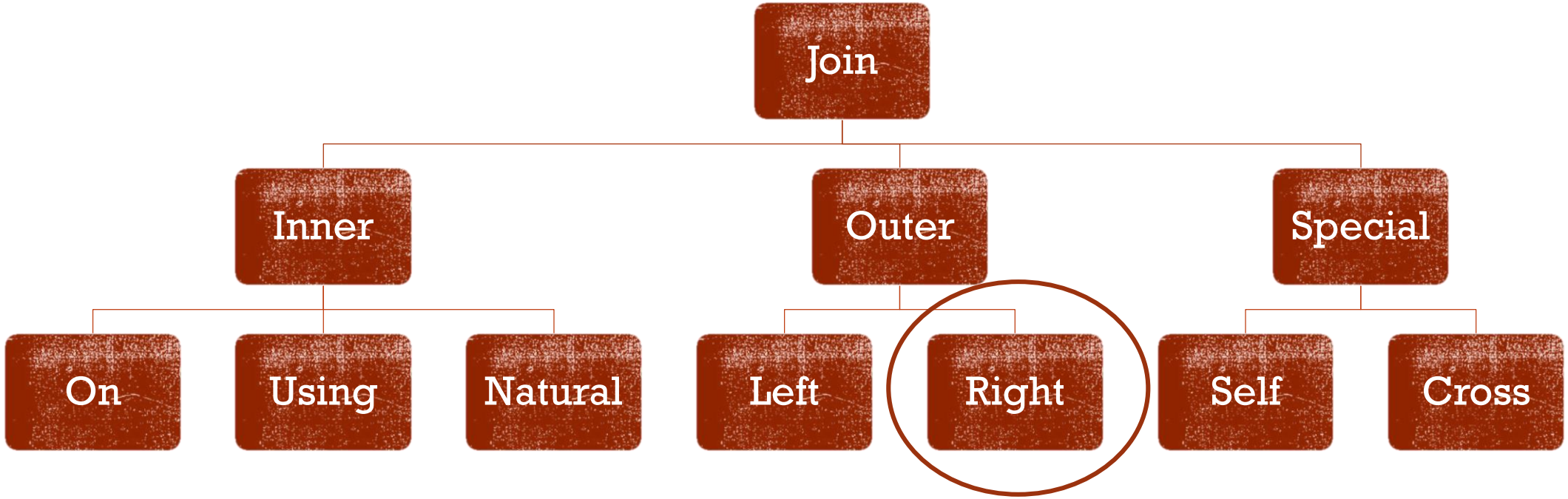
OUTER JOINs still require a JOIN condition; don't forget that!

LEFT OUTER JOIN PRACTICE #6



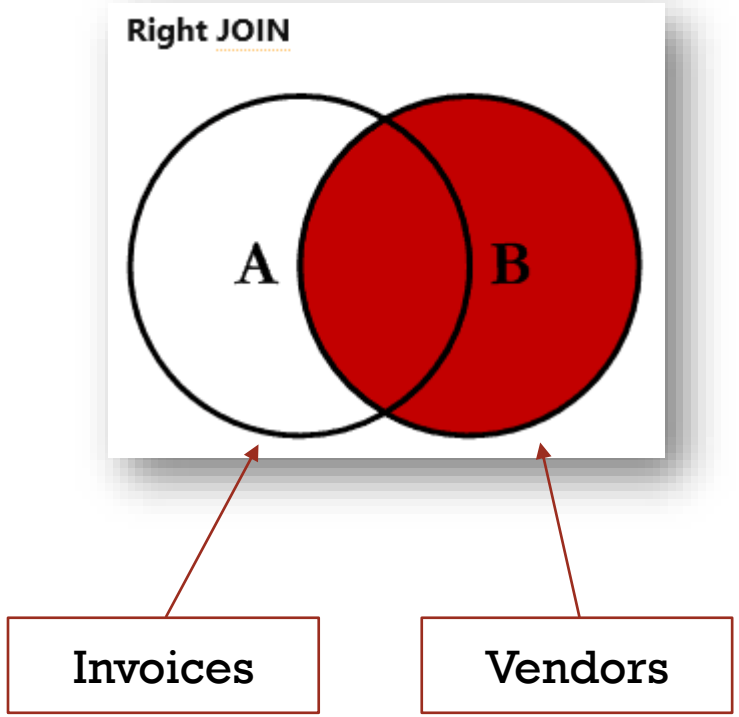
- Left Outer Join syntax:
  - ```
    SELECT select_list
    FROM table_1
        LEFT [OUTER] JOIN table_2
            ON table1col = table2col;
    ```
- Get a list of all customers who never placed an order
- Provide a call list that includes customer names and phone numbers
- Sort by last name, then first name
- Hints:
  - Get the join right before worrying about the "never placed an order" and the rest; develop your query incrementally
  - Don't forget that outer joins still need join conditions
    ```

JOIN HIERARCHY

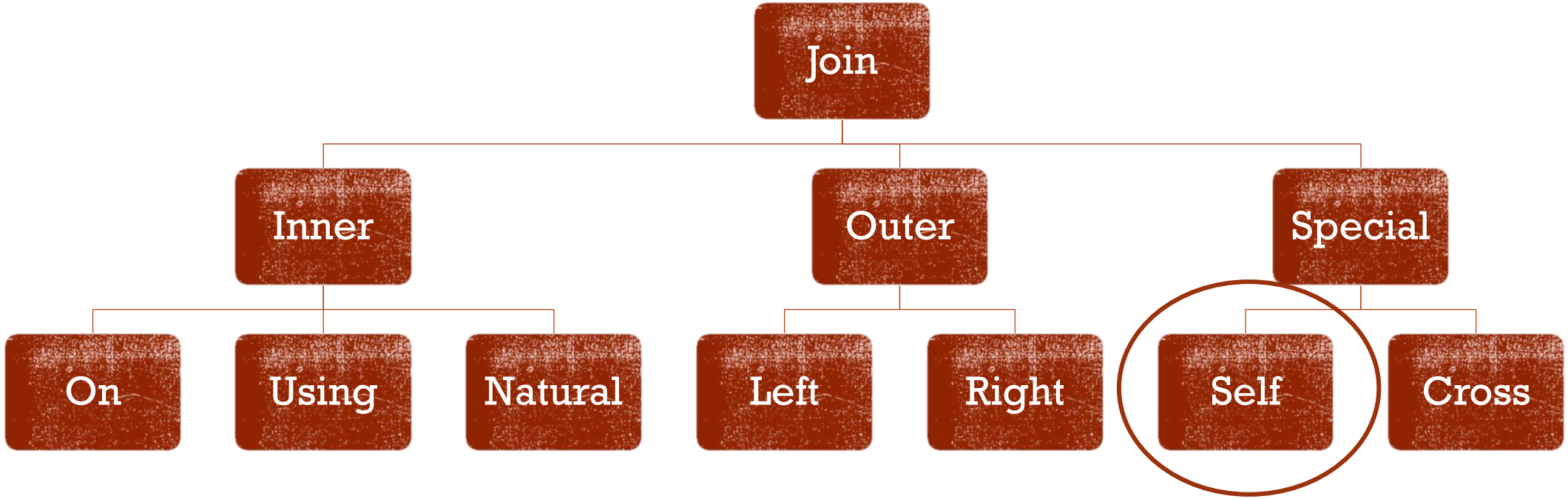


COMMON JOIN TYPES: RIGHT OUTER JOIN

- Less common is the Right Outer Join
- As shown at right, you are interested in all the data from the second ("right") table, and matching data from the first table
- You shouldn't be surprised to see nulls (unmatched data)
- An AP DB example:
  - I want a list of matching Invoices, where they exist, but all Vendors
  - Invoices with no associated Vendor? They won't be included
  - Vendors who have no associated Invoices? They'll be included
- In practice it's recommended you do only Left Outer Joins; it's common practice to list the "all of them" table first

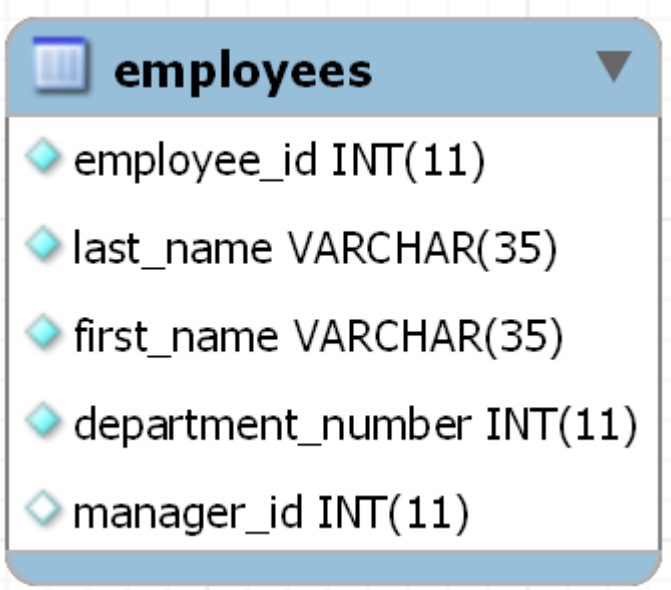


JOIN HIERARCHY



LESS COMMON JOIN TYPES: SELF JOIN

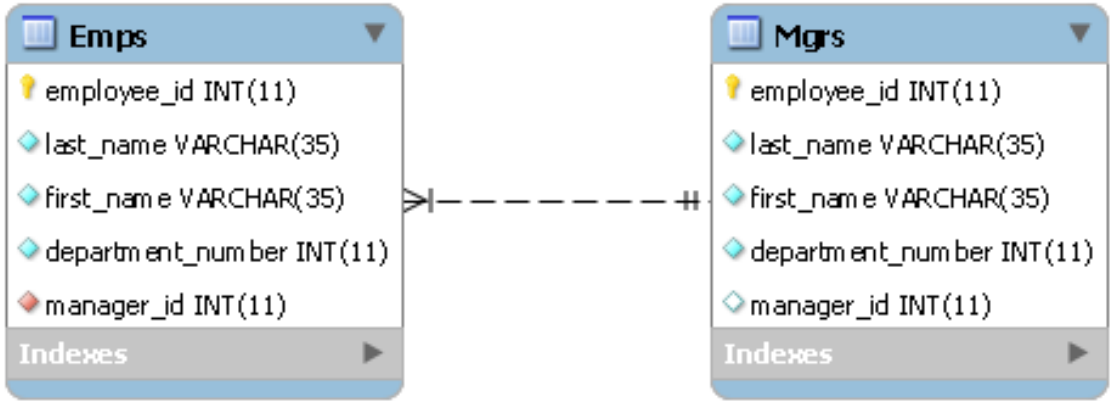
- In this join, you use aliases to inner join a single table *to itself*
- Example: from Ex's employees table, get a list of employees and their managers
  - Other interesting variations: find employees who *have no manager*, or find employees who are *not* managers



employee_id	last_name	first_name	department_number	manager_id
1	Smith	Cindy	2	NULL
2	Jones	Elmer	4	1
3	Simonian	Ralph	2	2
4	Hernandez	Olivia	1	9
5	Aaronsen	Robert	2	4
6	Watson	Denise	6	8
7	Hardy	Thomas	5	2
8	O'Leary	Rhea	4	9
9	Locario	Paulo	6	1

HOW TO CONCEPTUALIZE A SELF-JOIN

- Our brains might explode if we try to do this directly; there's a simpler way
- Let's pretend we have **two** tables instead of **one**. Draw it this way:



- In this case the query isn't difficult; to find a list of employees and managers:

```
SELECT Emps.last_name AS EmpLast, Emps.first_name AS EmpFirst,
       Mgrs.last_name AS MgrLast, Mgrs.first_name AS MgrFirst
FROM Emps JOIN Mgrs ON Emps.manager_id = Mgrs.employee_id;
```

- Now, what we we do to pretend ONE table is really TWO?
  - Anyone remember how to alias a table?

MORPHING "PRETEND" TWO TABLE SOLUTION INTO A SELF JOIN

EmpLast	EmpFirst	MgrLast	MgrFirst
Jones	Elmer	Smith	Cindy
Locario	Paulo	Smith	Cindy
Simonian	Ralph	Jones	Elmer
Hardy	Thomas	Jones	Elmer
Aaronsen	Robert	Hernandez	Olivia
Watson	Denise	O'Leary	Rhea
Hernandez	Olivia	Locario	Paulo
O'Leary	Rhea	Locario	Paulo

"Pretend" two-table SQL:

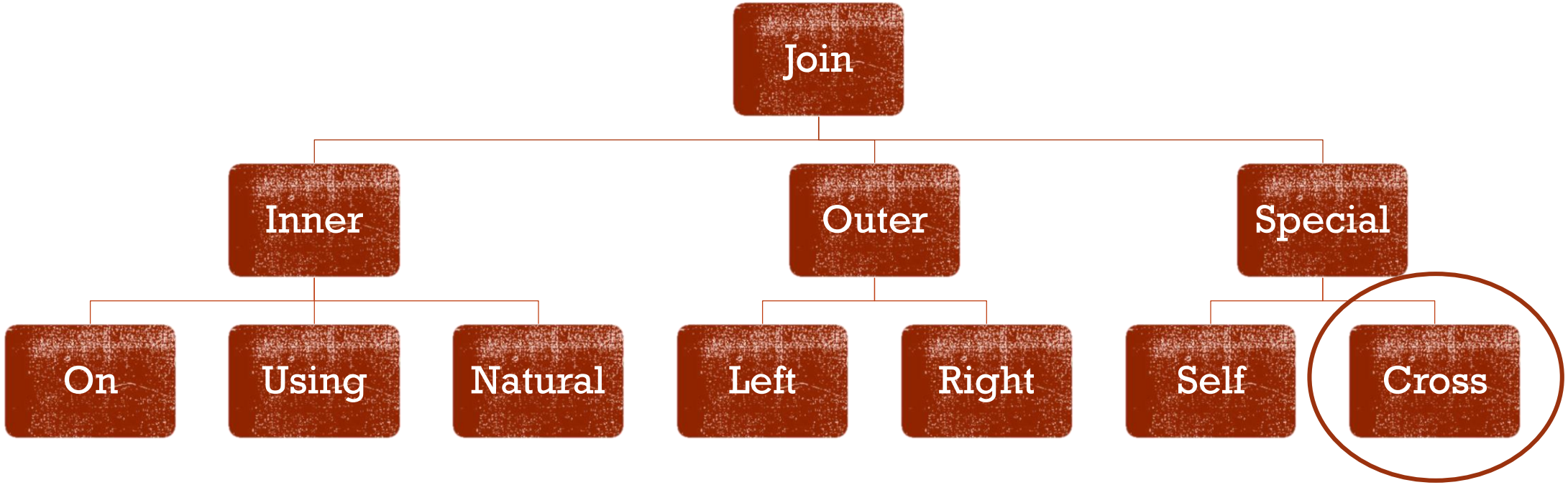
- ```
  SELECT Emps.last_name AS EmpLast, Emps.first_name AS EmpFirst,
         Mgrs.last_name AS MgrLast, Mgrs.first_name AS MgrFirst
  FROM Emps JOIN Mgrs ON Emps.manager_id = Mgrs.employee_id;
  ```

Real Self Join SQL:

- ```
  SELECT Emps.last_name AS EmpLast, Emps.first_name AS EmpFirst,
         Mgrs.last_name AS MgrLast, Mgrs.first_name AS MgrFirst
  FROM Employees Emps JOIN Employees Mgrs
      ON Emps.manager_id = Mgrs.employee_id;
  ```

- Note that everything stays the same except the little aliasing trick!

JOIN HIERARCHY



LESS COMMON JOIN TYPES: CROSS JOIN

- Joins *every* row in one table with *every* row in another table, a rare requirement!
- No condition is specified for this type of join
- If table1 has 20 rows and table2 has 30 rows, 600 rows will result!
- Example (ex DB): you want every department to vote on every color; get a list of all the voting that needs to be done; eliminate null color names from participation
-
```
-- Cross join example
SELECT department_name AS Dept, color_name AS Color, ' ' AS Vote
FROM ex.departments
    CROSS JOIN ex.color_sample
WHERE ex.color_sample.color_name IS NOT NULL;
```
- Note: you'll get an unintentional CROSS JOIN if you forget a JOIN condition on any other type of query; that will contaminate all the data in a multi-table JOIN, too!

JOINING MORE THAN TWO TABLES

JOINS: NOT JUST FOR TWO!

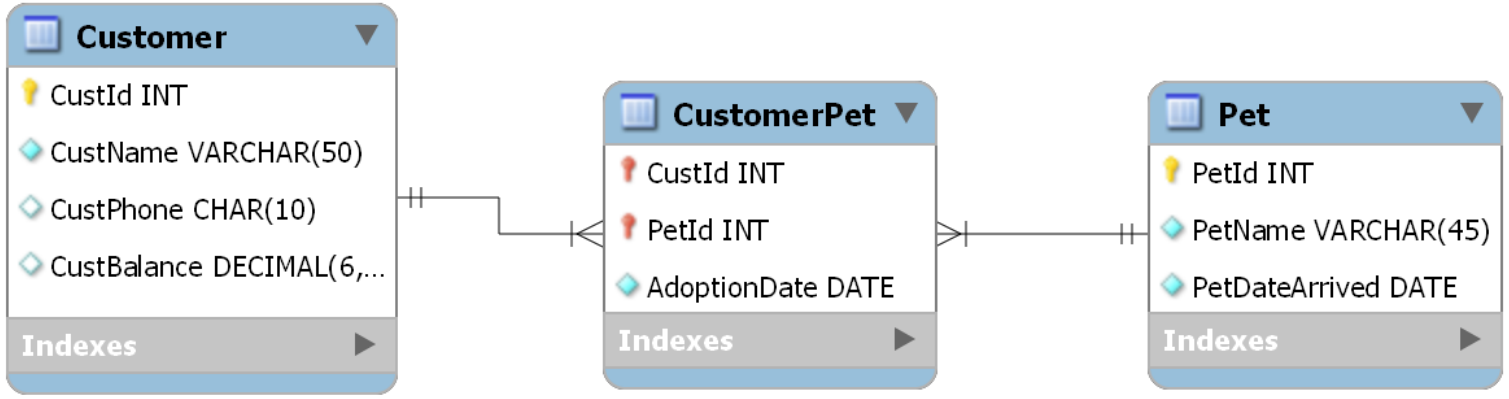
- More tables = more interesting
- Just keep joining on more tables; start at one end, JOIN until you reach the other
- Follow the relationships (PKs and FKs), in general; again an EER is super helpful
- Typical syntax:
  -
```
SELECT select_list
FROM table1
    INNER JOIN table2
        ON table1col = table2col
    INNER JOIN table3
        ON table2col = table3col
    …
```
- You can use a combination of inner and outer joins if necessary; but once you use an OUTER you'll likely need OUTERs from then on to preserve the non-matching data

MULTI-TABLE EXAMPLE

- Get a list of Pets (by name) and their adoptive parent's name and phone
-
```
SELECT PetName, CustName, CustPhone
FROM Pet JOIN CustomerPet USING (PetId)
         JOIN Customer    USING (CustId);
```

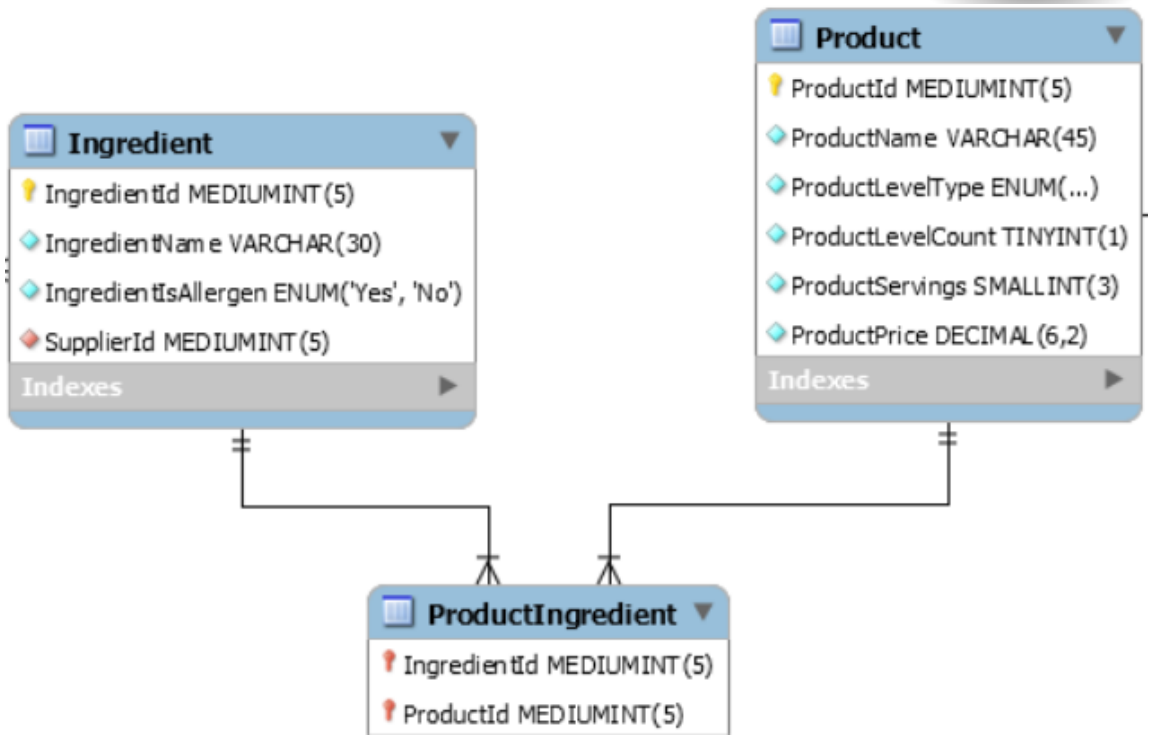
PetName	CustName	CustPhone
Lucky	Harriett Hanson	4258882626
Fluffy	Betty Beaumont	2068884747

Don't forget that *each* JOIN needs a condition



MULTI-TABLE PRACTICE #7

- Get a list of products along with the ingredients they contain (and whether the ingredient is a known allergen)
- Sort by ingredient, then by product name



MULTI-TABLE PRACTICE #8

- Get a list of customers who have ever ordered any type of cupcake
- Sort by name (last, then first)
- Don't show the same customer name more than once
- Don't look up and use IDs to accomplish this (here, or on projects!)

MULTI-TABLE PRACTICE #9

- Get a list of product orders of frosting-covered products (not fondant-covered ones) along with the cake flavor and frosting (exterior) flavor the customer chose
- Show the product name, the cake flavor name, and the exterior flavor name
- Show only cases where the two flavors aren't the same (we're looking for odd combinations). Don't repeat combinations
- Sort by product name, cake flavor name, then frosting flavor name
- Hint: write the query incrementally; this is important in complicated ones

This is a tough one; if you can do this one, you've graduated on this topic!

TIPS FOR MULTI-TABLE QUERIES

- Figure out what tables are involved
  - Look at what data is requested; it may come from multiple tables
  - Data may not be in adjacent tables; if so, all the tables in between need to be included
- Find a path between the tables
  - Usually you can find a direct path between the tables
  - In rare cases there may be two paths; pick that one that makes the most sense given the JOIN context
- Find the join conditions
  - Look for the PK/FK combinations that relate each pair of tables; use those to construct your join condition using either ON or USING
- Figure out what kind of join is required
  - Most questions require (normal) inner joins
  - Look for wording that gives you additional clues, e.g., "find ALL customers, plus…"
  - Look out for the rarer types; you will probably see one self join and one cross join among the project questions

WHAT SHOULD I DO NEXT?

- Take the quiz on Chapter 4
- Start Proj03
  - Querying multiple tables
  - Table design
- Before we meet again…
  - Submit Proj03
  - Read next week's material:
    - Chapter 5: Inserting, Updating, and Deleting Data
    - Chapter 8: Data Types

QUESTIONS?