

Copyright © 2015 by The Learning House.

All rights reserved. No part of these materials may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of The Learning House. For permission requests, write to The Learning House, addressed “Attention: Permissions Coordinator,” at the address below.

The Learning House
427 S 4th Street #300
Louisville KY 40202

Relationships and Joins

.NET Cohort

Coding Bootcamp

Lesson Goals

- Learn how relational databases eliminate duplicate data
- Learn how to walk across table relations to get data from multiple tables

Why Relationships?

- There are several goals in mind when we create relationships in databases:
 1. Save memory by not replicating the same data repeatedly in a table
 2. Simplify updating relational data
 3. Simplify enforcing *constraints* on the data
- The process by which we eliminate duplicate data into separate tables is called *normalization*.
 - There are 5 normal forms, but most databases stop at the 3rd normal form.

1st Normal Form

- To be in the 1st Normal Form:
 - Each table must have a *primary key* that uniquely identifies a record.
 - The values in each column must be atomic (no multi-value attributes).
 - There are no repeating groups, e.g. two columns don't store similar information in the same table.

Example of Going to 1st Normal Form

- Take, for example, the table below.
- What if two people share the same name?
 - How would we uniquely identify them?
- There is no primary key, so this is called a *heap*.

| FirstName | LastName | Telephone |
|-----------|---------------|-----------|
| Eric | Wise | 555-1234 |
| Jenny | GotYourNumber | 867-5309 |

A Primary Key Must be Unique

- Most of the time, we use an auto-number or a GUID (globally unique identifier) to create a *surrogate key* (a key not based on the entity's data). The database can generate and manage these when we create rows.

| PersonID | FirstName | LastName | Telephone |
|----------|-----------|---------------|-----------|
| 1 | Eric | Wise | 555-1234 |
| 2 | Jenny | GotYourNumber | 867-5309 |

But what if we need to store more than one phone number?

- Often, requirements dictate that we store more than one phone number or keep a history of phone numbers. Inexperienced people want to do this:

| PersonID | FirstName | LastName | Home | Mobile |
|----------|-----------|---------------|----------|----------|
| 1 | Eric | Wise | 555-1234 | 555-2345 |
| 2 | Jenny | GotYourNumber | 867-5309 | 111-2222 |

But normalization suggests we should move phone to another table:

| PersonID | FirstName | LastName |
|----------|-----------|---------------|
| 1 | Eric | Wise |
| 2 | Jenny | GotYourNumber |

| PhoneID | PersonID | Number | Type |
|---------|----------|----------|--------|
| 1 | 1 | 555-1234 | Home |
| 2 | 1 | 555-2345 | Mobile |
| 3 | 2 | 867-5309 | Home |
| 4 | 2 | 111-2222 | Mobile |

2nd Normal Form

- We satisfied 1st Normal Form, but now we have duplicate information across rows.
- 2nd Normal Form Requires:
 - 1st Normal Form
 - Redundant data across multiple rows must be moved to a separate table and joined by a key.

Going 2nd Normal Form

| PhoneID | PersonID | Number | PhoneTypeID |
|---------|----------|----------|-------------|
| 1 | 1 | 555-1234 | 1 |
| 2 | 1 | 555-2345 | 2 |
| 3 | 2 | 867-5309 | 1 |
| 4 | 2 | 111-2222 | 2 |

| PhoneTypeID | TypeName |
|-------------|----------|
| 1 | Home |
| 2 | Mobile |

3rd Normal Form

- To reach 3rd Normal Form:
 - 1st and 2nd Normal Form must be met
 - Eliminate any field that does not depend on the primary key:
 - This means any field which is not dependent on the primary key but on another field must be moved
 - Most databases don't go this far.

3rd Normal Form Example

| Tournament | Year | Winner | WinnerDOB |
|----------------------|------|----------------|------------------------------|
| Indiana Invitational | 1998 | Al Fredrickson | July 21 st , 1975 |
| Cleveland Open | 1999 | Bob Albertson | Sept 28 th , 1968 |
| Des Moines Masters | 1999 | Al Fredrickson | July 21 st , 1975 |
| Indiana Invitational | 1999 | Chip Masterson | Mar 14 th , 1977 |

Here, tournament and year are candidates for a key because they are unique, but the winner's date of birth is dependent on the winner name, not the tournament. 3rd normal form would have us move the winner and DOB to a new table.

Foreign Keys

- In the previous example, the CategoryID on Categories is the primary key, but it also exists on the Products table as a *Foreign Key*.
 - MySQL did not support foreign keys for a long time due to concerns that they would impact speed and performance. This means that *referential integrity* wasn't there and invalid data could easily be stored in MySQL.
 - Foreign keys in MySQL work on tables as long as they are of the *InnoDB* type.
- We don't have to join on foreign keys — we can join on any condition — but, for a good data model, you typically will (and the DBA will optimize the lookups for these matches).

Now that we've split all the data up...

We need some joins!

The first join is the INNER JOIN. It allows us to join the data of two tables on a matching key.

INNER JOIN returns rows where the key in both tables is an exact match.

For the INNER JOIN to work, we have to specify which tables and what field in each table to join on.

Typically, in a good data model, the join columns will be named the same, so we have to qualify the match with the table names.

```
SELECT ProductName, CategoryName
FROM Products
  INNER JOIN Categories
  ON Products.CategoryID = Categories.CategoryID
```

| # | ProductName | CategoryName |
|----|---------------------------------|----------------|
| 1 | Chai | Beverages |
| 2 | Chang | Beverages |
| 3 | Aniseed Syrup | Condiments |
| 4 | Chef Anton's Cajun Seasoning | Condiments |
| 5 | Chef Anton's Gumbo Mix | Condiments |
| 6 | Grandma's Boysenberry Spread | Condiments |
| 7 | Uncle Bob's Organic Dried Pears | Produce |
| 8 | Northwoods Cranberry Sauce | Condiments |
| 9 | Mishi Kobe Niku | Meat/Poultry |
| 10 | Ikura | Seafood |
| 11 | Queso Cabrales | Dairy Products |

Note About Inner Joins

Keep in mind that if the join column is null in the adjoining table, the record will not be displayed in the result set! They must be an exact match!

We can join almost (256) as many tables as we like

```
SELECT ProductName, CategoryName, CompanyName
FROM Products
    INNER JOIN Categories
    ON Products.CategoryID = Categories.CategoryID
    INNER JOIN Suppliers
    ON Products.SupplierID = Suppliers.SupplierID
```

| # | ProductName | CategoryName | CompanyName |
|---|---------------------------------|--------------|----------------------------|
| 1 | Chai | Beverages | Exotic Liquids |
| 2 | Chang | Beverages | Exotic Liquids |
| 3 | Aniseed Syrup | Condiments | Exotic Liquids |
| 4 | Chef Anton's Cajun Seasoning | Condiments | New Orleans Cajun Delights |
| 5 | Chef Anton's Gumbo Mix | Condiments | New Orleans Cajun Delights |
| 6 | Grandma's Boysenberry Spread | Condiments | Grandma Kelly's Homestead |
| 7 | Uncle Bob's Organic Dried Pears | Produce | Grandma Kelly's Homestead |

Lab Exercises

- Get a list of each employee and their territories.
- Get the Customer Name, Order Date, and each order detail's Product name for USA customers only.
- Get all the order information where Chai was sold.

LEFT JOINS

Whereas an inner join matches on exact matches, a left join contains all records from the left table and only matching records in the right table.

Say, for example, an internet order has no EmployeeID assigned to it

If we inner join orders to employee, the internet records will be excluded because there is no match.

If we left join from orders, we would get all of the orders regardless of the employee attached.

Right joins are the same as left joins, except reversed.

```
SELECT OrderID, CustomerID, Orders.EmployeeID,
       LastName, FirstName
FROM Orders
     INNER JOIN Employees
       ON Orders.EmployeeID = Employees.EmployeeID
```

| # | OrderID | CustomerID | EmployeeID | LastName | FirstName |
|---|---------|------------|------------|----------|-----------|
| 1 | 10258 | ERNSH | 1 | Davolio | Nancy |
| 2 | 10270 | WARTH | 1 | Davolio | Nancy |
| 3 | 10275 | MAGAA | 1 | Davolio | Nancy |
| 4 | 10285 | QUICK | 1 | Davolio | Nancy |

```
SELECT OrderID, CustomerID, Orders.EmployeeID,
       LastName, FirstName
FROM Orders
     LEFT JOIN Employees
       ON Orders.EmployeeID = Employees.EmployeeID
```

| | | | | |
|-------|-------|--------|-----------|--------|
| 10251 | VICTE | <NULL> | <NULL> | <NULL> |
| 10252 | SUPRD | <NULL> | <NULL> | <NULL> |
| 10253 | HANAR | 3 | Leverling | Janet |
| 10254 | CHOPS | 5 | Buchanan | Steven |
| 10255 | RICSU | 9 | Dodsworth | Anne |
| 10256 | WELLI | 3 | Leverling | Janet |

Full Outer Joins

- Full outer joins are the combination of left and right joins. It is the equivalent of saying “give me all rows from both tables regardless of whether they match.”
- Non-matching records in either table will have nulls for the other table’s columns

Filtering on Nulls

To filter on nulls, use the IS NULL or IS NOT NULL predicate:

```
SELECT OrderID, CustomerID, Orders.EmployeeID,  
       LastName, FirstName  
FROM Orders  
     LEFT JOIN Employees  
         ON Orders.EmployeeID = Employees.EmployeeID  
WHERE Orders.EmployeeID IS NULL OR LastName LIKE 'S%'
```

| # | OrderID | CustomerID | EmployeeID | LastName | FirstName |
|---|---------|------------|------------|----------|-----------|
| 1 | 10248 | VINET | <NULL> | <NULL> | <NULL> |
| 2 | 10249 | TOMSP | <NULL> | <NULL> | <NULL> |
| 3 | 10250 | HANAR | <NULL> | <NULL> | <NULL> |
| 4 | 10251 | VICTE | <NULL> | <NULL> | <NULL> |
| 5 | 10252 | SUPRD | <NULL> | <NULL> | <NULL> |
| 6 | 10264 | FOLKO | 6 | Suyama | Michael |
| 7 | 10271 | SPLIR | 6 | Suyama | Michael |
| 8 | 10272 | RATTC | 6 | Suyama | Michael |
| 9 | 10274 | VINET | 6 | Suyama | Michael |

Fin

- Next up: Query Writing Strategies!