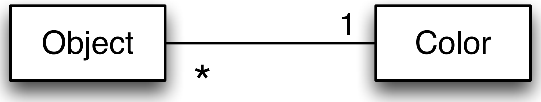
Using ERD and Relational Vocab to build joins

Our database design is very useful for building our join queries. Joining is about linking a foreign key to a primary key (or the reverse). The cardinalities tell us the relational vocab, and the relational vocab tell us which table the foreign key is in.



Color **:has_many** Object
Object **:belongs_to** Color

Remember, the foreign key is always in the table to the left of :belongs_to

Therefore, the foreign key is: **objects.color_id**
And the primary key is: **colors.id**

Therefore the join condition is: **objects.color_id = colors.id**
or (if going the other way) **colors.id = objects.color_id**

In this way we can "move" across our ERD to answer any question asked of the database. What we need to do is look closely at the question to know where we start and where we end. We end with the table that has the column that holds the answer to the question. We start with the table that has the information given in the question.

For example: "What color is the mug?" The information we are given is the name of an object ("the mug") and the information we are asked for is the name of a color ("What color"). (Note that your answer is not going to be an id, unless specifically asked for).

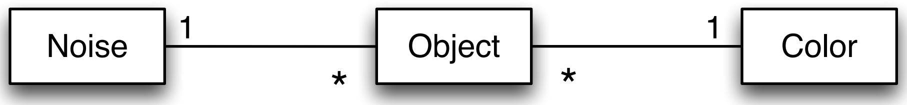
Therefore we start in the objects table (since that is where "mug" is.

```
-- check we are spelling table name correctly
SELECT *
FROM objects

-- check that we can find the starting information
SELECT *
FROM objects
WHERE objects.name = "mug"

-- Ok, now we have the right row, time to join in the color information for that row.
-- First we add the table name to the FROM clause
SELECT *
FROM objects, color
WHERE objects.name = "mug"

-- Whenever we add to the FROM clause, we must immediately add the join condition
-- (without first running the query). We're "moving" from objects to color, so we choose
-- the join condition that has the objects table first. We add that with an AND.
SELECT *
FROM objects, color
WHERE objects.name = "mug" AND
      objects.color_id = colors.id
```

--Finally we can alter our SELECT clause to get just the answer column: SELECT **colors.name**

Joining three or more tables

The procedure for moving from 2 to 3 tables applies the same procedure that we applied in moving from 1 to 2 tables. First we inspect the ERD and relational vocal to figure out the location of the foreign key and thus the join condition. Then we add the table to the FROM clause and the join condition to the WHERE clause.



Noise :**has_many Object**
Object :**belongs_to** Noise
therefore, join condition is
noises.id = objects.noise_id
or (other direction)
objects.noise_id = noises.id

Color :**has_many** Object
Object :**belongs_to** Color
therefore, join condition is
objects.color_id = color.id
or (other direction
color.id = objects.color_id

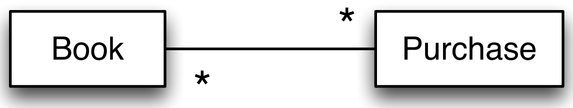
For the question, "What are the colors of the clicky objects?" We start in noises (since "clicky" is a noise and "move" over to colors (which the end, because it's the answer). We can't get directly from noises to colors, so we move "through" objects.

```
SELECT *
FROM noises

SELECT *
FROM noises
WHERE noises.name = "clicky"

-- Add in objects table and it's join condition
SELECT *
FROM noises, objects
WHERE noises.name = "clicky"  AND
    noises.id = objects.noise_id
```

```
--Now move from objects to colors
SELECT *
FROM noises, objects, colors
WHERE noises.name = "clicky"  AND
    noises.id = objects.noise_id AND
    objects.color_id = color.id

-- Finally we can narrow down to just
-- the name of the color.
SELECT colors.name
FROM noises, objects, colors
WHERE noises.name = "clicky"  AND
    noises.id = objects.noise_id AND
    objects.color_id = color.id
```

Joining :habtm

:has_and_belongs_to_many always involves joining three tables, because those are actually three tables (the two entity tables and the simple association table.)



Book :**habtm** Purchase
Purchase :**habtm** Book

therefore there is a table book_purchase., which has both of the foreign keys, making the join conditions
books.id = book_purchase.book_id (or reversed)
book_purchase.purchase_id = purchases.id (or reversed)

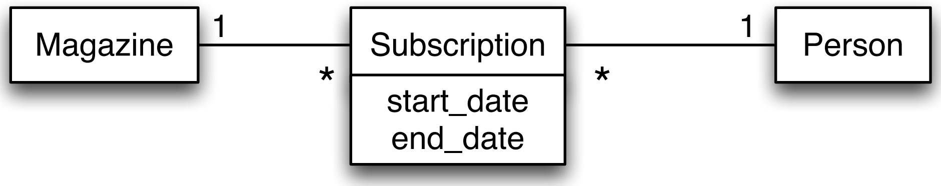
When was "Wuthering Heights" purchased? We start in books (using the title "Wuthering Heights" and the answer is a date ("when"), which is in the purchases table.

```
SELECT *
FROM books

SELECT *
FROM books
WHERE books.title = "Wuthering Heights"

-- Add in the simple association table and the
-- join condition
SELECT *
FROM books, book_purchase
WHERE books.title = "Wuthering Heights"  AND
     books.id = book_purchase.book_id
```

```
-- Now move to the purchases table
SELECT *
FROM books, book_purchase, purchases
WHERE books.title = "Wuthering Heights"  AND
     books.id = book_purchase.book_id AND
     book_purchase.purchase_id = purchases.id

--Ok we're in the answer table.  Now choose the
-- column with the answer
SELECT purchases.purchase_date
FROM books, book_purchase, purchases
WHERE books.title = "Wuthering Heights"  AND
     books.id = book_purchase.book_id AND
     book_purchase.purchase_id = purchases.id
```

Adding conditions "en route"

Sometimes the question will give information that has to be used not in the starting table, but in a table "en route" to the answer. This is common when working with :through relationships.



Magazine **:has_many** People **:through** Subscription
People **:has_many** Magazine **:through** Subscription
Subscription **:belongs_to** Magazine
Subscription **:belongs_to** People

therefore the foreign keys are in Subscription
making the join conditions
magazines.id = subscriptions.magazine_id (or reversed)
subscriptions.person_id = people.id (or reversed)

Which people have subscriptions to "The Week" ending this year (2015)?

The start is a magazine "The week" and the end is people's names. But we also have to restrict this to relationships that end this year. And that is in the subscriptions table. btw, "ending this year" can be written as end_date < "01-01-2016 00:00:00" (i.e. midnight on new year's eve).

```
SELECT *
FROM magazines

SELECT *
FROM magazines
WHERE magazine.title = "The Week"

-- Add in the subscriptions table
SELECT *
FROM magazines, subscriptions
WHERE magazines.title = "The Week"  AND
     magazine.id = subscriptions.magazine_id

--Now, if we look at these results, we can see that we
have all of the subscriptions for "The Week".  We need
to restrict those to the ones that end this year, so we
add this filter now (as we "pass through" subscriptions)
SELECT *
FROM magazines, subscriptions
WHERE magazines.title = "Wuthering Heights"  AND
     magazines.id = subscriptions.magazine_id AND
    subscriptions.end_date < "01-01-2016 00:00:00"

-- Now we have only subscriptions ending this
year, so we can move on to the people table
to get names.
SELECT *
FROM magazines, subscriptions, people
WHERE books.title = "The Week"  AND
     magazines.id = subscriptions.magazine_id AND
    subscriptions.end_date < "01-01-2016 00:00:00" AND
     subscriptions.person_id = people.id

-- All looks good, so finally we get the name column.
SELECT people.name
FROM magazines, subscriptions, people
WHERE magazines.title = "The Week"  AND
     magazines.id = subscriptions.magazine_id AND
    subscriptions.end_date < "01-01-2016 00:00:00" AND
     subscriptions.person_id = people.id
```