

```

1  /*
2      Lesson Overview
3          In this lesson you will be:
4
5          1.Creating Joins
6          2.Using Primary - Foreign Keys
7          3.Integrating Aliases
8          4.Evaluating Various Join Types
9          5.Integrating Filters with Joins
10 */
11
12 -- Database Normalization :
13 -- When creating a database,
14 -- it is really important to think about how data will be stored.
15
16 /*
17     There are essentially three ideas that are aimed at database normalization:
18
19     1. Are the tables storing logical groupings of the data?
20     2. Can I make changes in a single location,
21        rather than in many tables for the same information?
22     3. Can I access and manipulate data quickly and efficiently?
23 */
24
25 SELECT orders.*
26     FROM orders ;
27
28 -- JOIN statements
29 --     is to allow us to pull data from more than one table at a time.
30
31 SELECT accounts.* ,orders.*
32     FROM accounts
33     JOIN orders
34         ON orders.account_id = accounts.id
35     ORDER BY accounts.id
36     LIMIT 15;
37
38 --ON
39 -- We use ON clause to specify a JOIN condition which is a logical statement to
40 -- combine the table in FROM and JOIN statements.
41
42 -- The ON statement holds the two columns that get linked across the two tables
43
44 -- This query only pulls two columns, not all the information in these two tables.
45
46 SELECT accounts.name, orders.occurred_at
47     FROM orders
48     JOIN accounts
49         ON orders.account_id = accounts.id;
50
51 -- the below query pulls all the columns from both the accounts and orders table.
52
53 SELECT *
54     FROM orders
55     JOIN accounts
56         ON orders.account_id = accounts.id;
57
58 -- query you ran pull all the information from only the orders table
59
60 SELECT orders.*
61     FROM orders
62     JOIN accounts
63         ON orders.account_id = accounts.id;
64
65 -- Quiz Questions :
66
67 /*
68     Try pulling all the data from the accounts table,
69     and all the data from the orders table.
70 */
71
72 SELECT accounts.* , orders.*
73     FROM orders
74     JOIN accounts
75         ON accounts.id = orders.account_id
76     LIMIT 5;

```

```

70
71 /*
72     Try pulling standard_qty, gloss_qty, and poster_qty from the orders table,
73     and the website and the primary_poc from the accounts table.
74 */
75 SELECT orders.poster_qty , orders.gloss_qty , orders.standard_qty , accounts.website , accounts.primary_poc
76     FROM accounts
77     JOIN orders
78         ON accounts.id = orders.account_id
79     LIMIT 5;
80
81 SELECT orders.* , accounts.*
82     FROM orders
83     JOIN accounts
84         ON orders.account_id = accounts.id
85     LIMIT 10;
86
87 SELECT orders.* , accounts.*
88     FROM accounts
89     JOIN orders
90         ON orders.account_id = accounts.id
91     LIMIT 3;
92
93 SELECT orders.* , accounts.*
94     FROM accounts
95     JOIN orders
96         ON accounts.id = orders.account_id
97     LIMIT 3;
98
99 SELECT orders.* , accounts.*
100     FROM orders
101     JOIN accounts
102         ON orders.account_id = accounts.id
103     LIMIT 10;
104
105 SELECT orders.standard_amt_usd , orders.gloss_amt_usd , orders.poster_amt_usd ,
106         accounts.website , accounts.primary_poc
107     FROM orders
108     JOIN accounts
109         ON accounts.id = orders.account_id ;
110
111 -- entity-relationship diagram (ERD)
112 -- is a common way to view data in a database.
113 -- It is also a key element to understanding how we can pull data from multiple tables.
114
115 -- the primary key
116 -- it is a column that has a unique value for every row.
117
118 -- foreign key
119 -- it is a column in one table that is a primary key in a different table.
120
121 -- JOIN More than Two Tables
122 SELECT *
123     FROM web_events
124     JOIN accounts
125         ON web_events.account_id = accounts.id
126     JOIN orders
127         ON accounts.id = orders.account_id;
128
129 -- notes
130 -- we can create a SELECT statement that could pull specific columns from any of the three tables.
131 -- JOIN holds a table
132 -- ON is a link for our PK to equal the FK.
133 SELECT web_events.channel, accounts.name, orders.total
134     FROM web_events
135     JOIN accounts
136         ON web_events.account_id = accounts.id
137     JOIN orders
138         ON accounts.id = orders.account_id;

```

```

139 -- Questions
140
141
142 /*
143     Provide a table for all web_events associated with the account name of Walmart.
144     There should be three columns. Be sure to include the primary_poc,
145     time of the event, and the channel for each event. Additionally,
146     you might choose to add a fourth column to
147     assure only Walmart events were chosen.
148 */
149 SELECT accounts.name , accounts.primary_poc , accounts.website ,
150        web_events.occurred_at , web_events.channel
151 FROM web_events
152 JOIN accounts
153     ON accounts.id = web_events.account_id
154 WHERE accounts.name = 'Walmart';
155
156 /*
157     Provide a table that provides the region for each sales_rep
158     along with their associated accounts.
159     Your final table should include three columns: the region name,
160     the sales rep name, and the account name.
161     Sort the accounts alphabetically (A-Z) according to the account name.
162 */
163 SELECT region.name , sales_reps.name , accounts.name
164 FROM sales_reps -- from , it is a table has PK @ 1st step
165 JOIN region
166     ON region.id = sales_reps.region_id
167 JOIN accounts
168     ON accounts.sales_rep_id = sales_reps.id
169 ORDER BY accounts.name;
170
171 /*
172     Provide the name for each region for every order,
173     as well as
174         1.the account name and
175         2.the unit price they paid (total_amt_usd/total) for the order.
176     Your final table should have 3 columns:
177         region name, account name, and unit price.
178
179     >>A few accounts have 0 for total, so I divided by (total + 0.01) to
180     assure not dividing by zero.
181 */
182 SELECT accounts.name , (orders.total_amt_usd/(orders.total+0.01)) as unit_price ,
183        sales_reps.name
184 FROM region
185 JOIN sales_reps
186     ON sales_reps.region_id = region.id
187 JOIN accounts
188     ON accounts.sales_rep_id = sales_reps.id
189 JOIN orders
190     ON accounts.id = orders.account_id;
191
192 -- LEFT and RIGHT JOINS
193
194 -- Notice
195 -- 1.The first shows JOINS the way you have currently been working with data.
196 -- 2.The second shows LEFT and RIGHT JOIN statements.
197
198 -- TYPES OF JOINS
199
200 -- 1-equy join
201 /*
202     SELECT <columns_name>
203     FROM <table1_name> , <table2_name>
204     WHERE <table1_name>.<column_name>(PK) = <table2_name>.<column_name>(FK);
205 */
206 SELECT accounts.id , orders.id , accounts.name , orders.total
207 FROM orders , accounts

```

```

208 WHERE accounts.id = orders.account_id
209 ORDER BY accounts.id
210 LIMIT 10;
211
212 -- 2-INNER join JOIN
213 /*
214 SELECT <columns_name>
215 FROM <table1_name>
216 INNER JOIN <table2_name>
217 ON <table1_name>.<column_name>(fK) = <table2_name>.<column_name>(pK);
218 */
219 SELECT accounts.id , orders.id , accounts.name , orders.total
220 FROM orders
221 JOIN accounts
222 ON accounts.id = orders.account_id
223 ORDER BY accounts.id
224 LIMIT 10;
225
226 SELECT accounts.id , orders.id , accounts.name , orders.total
227 FROM orders INNER JOIN accounts
228 ON accounts.id = orders.account_id
229 ORDER BY accounts.id
230 LIMIT 10;
231
232 -- 3-OUTER join
233
234 -- 3.1-LEFT OUTER join
235 /*
236 SELECT <columns_name>
237 FROM <table1_name>
238 LEFT OUTER <table2_name>
239 ON <table1_name>.<column_name>(fK) = <table2_name>.<column_name>(pK)
240 WHERE <table2_name>.pk is NULL;
241 */
242 SELECT accounts.id , orders.id , accounts.name , orders.total
243 FROM orders
244 left OUTER JOIN accounts
245 ON accounts.id = orders.account_id
246 ORDER BY accounts.id
247 LIMIT 10;
248
249 SELECT accounts.id , orders.id , accounts.name , orders.total
250 FROM orders
251 left JOIN accounts
252 ON accounts.id = orders.account_id
253 ORDER BY accounts.id
254 LIMIT 10;
255
256 -- 3.1.1-LEFT OUTER join with EXCLUSION
257 /*
258 SELECT <columns_name>
259 FROM <table1_name>
260 LEFT OUTER <table2_name>
261 ON <table1_name>.<column_name>(fK) = <table2_name>.<column_name>(pK)
262 WHERE <table2_name>.pk is NULL;
263 */
264 SELECT accounts.id , orders.id , accounts.name , orders.total
265 FROM orders
266 left OUTER JOIN accounts
267 ON accounts.id = orders.account_id
268 WHERE accounts.id is NULL
269 ORDER BY accounts.id
270 LIMIT 10;
271
272 -- RIGHT and FULL OUTER JOINS are not currently supported
273
274 -- 3.2-RIGHT OUTER join
275 /*
276 SELECT <columns_name>

```

```

277         FROM <table1_name>
278         RIGHT OUTER <table2_name>
279         ON <table1_name>.<column_name>(fK) = <table2_name>.<column_name>(pK)
280         WHERE <table2_name>.pk is NULL;
281     */
282     SELECT *
283     FROM orders
284     RIGHT OUTER JOIN accounts
285     ON accounts.id = orders.account_id
286     ORDER BY accounts.id
287     LIMIT 10;
288
289     SELECT accounts.id , orders.id , accounts.name , orders.total
290     FROM orders
291     RIGHT JOIN accounts
292     ON accounts.id = orders.account_id
293     ORDER BY orders.id
294     LIMIT 10;
295
296     -- 3.2.1-RIGHT OUTER join with EXCLUSION
297     /*
298         SELECT <columns_name>
299         FROM <table1_name>
300         RIGHT OUTER <table2_name>
301         ON <table1_name>.<column_name>(fK) = <table2_name>.<column_name>(pK)
302         WHERE <table2_name>.pk is NULL;
303     */
304     SELECT accounts.id , orders.id , accounts.name , orders.total
305     FROM orders
306     RIGHT OUTER JOIN accounts
307     ON accounts.id = orders.account_id
308     WHERE accounts.id is NULL
309     ORDER BY accounts.id
310     LIMIT 10;
311
312     -- RIGHT and FULL OUTER JOINS are not currently supported
313
314     -- 3.3-FULL OUTER join
315     /*
316         SELECT <columns_name>
317         FROM <table1_name>
318         RIGHT OUTER <table2_name>
319         ON <table1_name>.<column_name>(fK) = <table2_name>.<column_name>(pK)
320         WHERE <table2_name>.pk is NULL;
321     */
322     SELECT *
323     FROM orders
324     LEFT OUTER JOIN accounts
325     ON accounts.id = orders.account_id
326     WHERE accounts.id is NULL
327     UNION
328     SELECT accounts.id , orders.id , accounts.name , orders.total
329     FROM orders
330     RIGHT OUTER JOIN accounts
331     ON accounts.id = orders.account_id
332     WHERE accounts.id is NULL ;
333
334     -- 3.3-FULL OUTER join with exclusion
335     /*
336         SELECT <columns_name>
337         FROM <table1_name>
338         RIGHT OUTER <table2_name>
339         ON <table1_name>.<column_name>(PK) = <table2_name>.<column_name>(FK);
340     */
341     SELECT *
342     FROM orders
343     LEFT OUTER JOIN accounts
344     ON accounts.id = orders.account_id
345     UNION

```

```

346 SELECT accounts.id , orders.id , accounts.name , orders.total
347 FROM orders
348 RIGHT OUTER JOIN accounts
349 ON accounts.id = orders.account_id ;
350
351
352 -- TESTING
353 SELECT a.id, a.name, o.total FROM orders o RIGHT JOIN accounts a ON o.account_id = a.id;
354
355
356
357 -- JOINS and Filtering
358 /*
359 A simple rule to remember is that, when the database executes this query,
360 it executes the join and everything in the ON clause first.
361 Think of this as building the new result set.
362 That result set is then filtered using the WHERE clause.
363
364 The fact that this example is a left join is important.
365 Because inner joins only return the rows for which the two tables match,
366 moving this filter to the ON clause of an inner join will produce
367 the same result as keeping it in the WHERE clause.
368 */
369
370 SELECT orders.*, accounts.*
371 FROM orders
372 LEFT JOIN accounts
373 ON orders.account_id = accounts.id
374 WHERE accounts.sales_rep_id = 321500;
375
376 -- Questions
377 /*
378 Q1
379 Provide a table that provides the region for each sales_rep along with
380 their associated accounts. This time only for the Midwest region.
381 Your final table should include three columns:
382 the region name, the sales rep name, and the account name.
383 Sort the accounts alphabetically (A-Z) according to the account name.
384 */
385 --code
386 SELECT region.name AS REGION_NAME , sales_reps.name AS SALES_REPS_NAME , accounts.name AS ACCOUNTS_NAME
387 FROM sales_reps
388 JOIN region
389 ON sales_reps.region_id = region.id
390 JOIN accounts
391 ON accounts.sales_rep_id = sales_reps.id
392 WHERE region.name = 'Midwest'
393 ORDER BY accounts.name;
394
395
396 /*
397 Q2
398 Provide a table that provides the region for each sales_rep along with
399 their associated accounts. This time only for accounts where
400 the sales rep has a first name starting with S and in the Midwest region.
401 Your final table should include three columns:
402 the region name, the sales rep name, and the account name.
403 Sort the accounts alphabetically (A-Z) according to the account name.
404 */
405 --code
406 SELECT region.name AS REGION_NAME , sales_reps.name AS SALES_REPS_NAME , accounts.name AS ACCOUNTS_NAME
407 FROM sales_reps
408 JOIN region
409 ON sales_reps.region_id = region.id
410 JOIN accounts
411 ON accounts.sales_rep_id = sales_reps.id
412 WHERE region.name = 'Midwest' and sales_reps.name like 'S%'
413 ORDER BY accounts.name;
414

```

```

415  /*
416  /*
417      Q3
418      Provide a table that provides the region for each sales_rep along with
419      their associated accounts. This time only for accounts where
420      the sales rep has a last name starting with K and in the Midwest region.
421      Your final table should include three columns:
422      the region name, the sales rep name, and the account name.
423      Sort the accounts alphabetically (A-Z) according to the account name.
424  */
425  --code
426  SELECT region.name AS REGION_NAME , sales_reps.name AS SALES_REPS_NAME , accounts.name AS ACCOUNTS_NAME
427  FROM sales_reps
428  JOIN region
429      ON sales_reps.region_id = region.id
430  JOIN accounts
431      ON accounts.sales_rep_id = sales_reps.id
432  WHERE region.name = 'Midwest' and sales_reps.name like '%K%'
433  ORDER BY accounts.name;
434
435
436  /*
437      Q4
438      Provide the name for each region for every order, as well as
439      the account name and the unit price they paid (total_amt_usd/total)
440      for the order. However, you should only provide the results
441      if the standard order quantity exceeds 100.
442      Your final table should have 3 columns:
443      region name, account name, and unit price.
444      In order to avoid a division by zero error,
445      adding .01 to the denominator here is helpful total_amt_usd/(total+0.01).
446  */
447  --code
448  SELECT region.name AS REGION_NAME , accounts.name AS ACCOUNTS_NAME,
449      orders.total_amt_usd /(orders.total+0.01) AS UNIT_PRICE
450  FROM sales_reps
451  JOIN region
452      ON sales_reps.region_id = region.id
453  JOIN accounts
454      ON accounts.sales_rep_id = sales_reps.id
455  JOIN orders
456      ON orders.account_id = accounts.id
457  WHERE orders.standard_qty > 100
458  ORDER BY accounts.name;
459
460
461
462  /*
463      Q5
464      Provide the name for each region for every order, as well as
465      the account name and the unit price they paid (total_amt_usd/total)
466      for the order. However, you should only provide the results
467      if the standard order quantity exceeds 100 and
468      the poster order quantity exceeds 50.
469      Your final table should have 3 columns:
470      region name, account name, and unit price.
471      Sort for the smallest unit price first.
472      In order to avoid a division by zero error,
473      adding .01 to the denominator here is helpful (total_amt_usd/(total+0.01).
474  */
475  --code
476  SELECT region.name AS REGION_NAME , accounts.name AS ACCOUNTS_NAME,
477      orders.total_amt_usd /(orders.total+0.01) AS UNIT_PRICE
478  FROM sales_reps
479  JOIN region
480      ON sales_reps.region_id = region.id
481  JOIN accounts
482      ON accounts.sales_rep_id = sales_reps.id
483  JOIN orders

```

```

484     ON orders.account_id = accounts.id
485 WHERE orders.standard_qty > 100 AND orders.poster_qty > 50
486 ORDER BY UNIT_PRICE;
487
488
489 /*
490 Q6
491 Provide the name for each region for every order,
492 as well as the account name and the unit price they paid (total_amt_usd/total)
493 for the order. However, you should only provide the results
494 if the standard order quantity exceeds 100 and
495 the poster order quantity exceeds 50.
496 Your final table should have 3 columns:
497 region name, account name, and unit price.
498 Sort for the largest unit price first.
499 In order to avoid a division by zero error,
500 adding .01 to the denominator here is helpful (total_amt_usd/(total+0.01)).
501 */
502 --code
503 SELECT region.name AS REGION_NAME , accounts.name AS ACCOUNTS_NAME,
504        orders.total_amt_usd /(orders.total+0.01) AS UNIT_PRICE
505 FROM region
506 JOIN sales_reps
507     ON sales_reps.region_id = region.id
508 JOIN accounts
509     ON accounts.sales_rep_id = sales_reps.id
510 JOIN orders
511     ON orders.account_id = accounts.id
512 WHERE orders.standard_qty > 100 AND orders.poster_qty > 50
513 ORDER BY UNIT_PRICE DESC;
514
515
516 /*
517 Q7
518 What are the different channels used by account id 1001?
519 Your final table should have only 2 columns:
520 account name and the different channels.
521 You can try SELECT DISTINCT to narrow down the results
522 to only the unique values.
523 */
524 --code
525 SELECT DISTINCT accounts.name , web_events.channel
526 FROM accounts
527 JOIN web_events
528     ON web_events.account_id = accounts.id
529 WHERE accounts.id = '1001';
530
531
532 /*
533 Q8
534 Find all the orders that occurred in 2015.
535 Your final table should have 4 columns:
536 occurred_at, account name, order total, and order total_amt_usd.
537 */
538 --code
539 SELECT orders.occurred_at , accounts.name , orders.total , orders.total_amt_usd
540 FROM accounts
541 JOIN orders
542     ON orders.id= accounts.id
543 WHERE orders.occurred_at = '2015';
544

```