```sql
/*
    Lesson Overview
        In this lesson you will be:

            1.Creating Joins
            2.Using Primary - Foreign Keys
            3.Integrating Aliases
            4.Evaluating Various Join Types
            5.Integrating Filters with Joins
*/

-- Database Normalization :
--  When creating a database,
--  it is really important to think about how data will be stored.

/*
    There are essentially three ideas that are aimed at database normalization:

    1. Are the tables storing logical groupings of the data?
    2. Can I make changes in a single location,
        rather than in many tables for the same information?
    3. Can I access and manipulate data quickly and efficiently?
*/

SELECT orders.*
    FROM orders  ;

-- JOIN statements
--      is to allow us to pull data from more than one table at a time.

SELECT  accounts.* ,orders.*
    FROM accounts
    JOIN orders
        ON orders.account_id = accounts.id
        ORDER BY accounts.id
        LIMIT 15;
--ON
-- We use ON clause to specify a JOIN condition which is a logical statement to
--      combine the table in FROM and JOIN statements.
-- The ON statement holds the two columns that get linked across the two tables

-- This query only pulls two columns, not all the information in these two tables.
SELECT accounts.name, orders.occurred_at
    FROM orders
    JOIN accounts
        ON orders.account_id = accounts.id;

-- the below query pulls all the columns from both the accounts and orders table.
SELECT *
    FROM orders
    JOIN accounts
        ON orders.account_id = accounts.id;

-- query you ran pull all the information from only the orders table
SELECT orders.*
    FROM orders
    JOIN accounts
        ON orders.account_id = accounts.id;

-- Quiz Questions :
/*
    Try pulling all the data from the accounts table,
    and all the data from the orders table.
*/
SELECT accounts.* , orders.*
    FROM orders
    JOIN accounts
        ON accounts.id = orders.account_id
    LIMIT 5;
```

```sql

/*
    Try pulling standard_qty, gloss_qty, and poster_qty from the orders table,
    and the website and the primary_poc from the accounts table.
*/
SELECT orders.poster_qty , orders.gloss_qty , orders.standard_qty , accounts.website , accounts.primary_poc
    FROM accounts
    JOIN orders
        ON accounts.id = orders.account_id
    LIMIT 5;

SELECT orders.* , accounts.*
    FROM orders
    JOIN accounts
        ON orders.account_id = accounts.id
        LIMIT 10;

SELECT orders.* , accounts.*
    FROM accounts
    JOIN orders
        ON orders.account_id = accounts.id
    LIMIT 3;

SELECT orders.* , accounts.*
    FROM accounts
    JOIN orders
        ON accounts.id = orders.account_id
    LIMIT 3;

SELECT orders.* , accounts.*
    FROM orders
    JOIN accounts
        ON orders.account_id = accounts.id
        LIMIT 10;

SELECT orders.standard_amt_usd , orders.gloss_amt_usd , orders.poster_amt_usd ,
        accounts.website , accounts.primary_poc
    FROM orders
    JOIN accounts
        ON  accounts.id = orders.account_id ;

-- entity-relationship diagram (ERD)
    -- is a common way to view data in a database.
    -- It is also a key element to understanding how we can pull data from multiple tables.

-- the primary key
--  it is a column that has a unique value for every row.

-- foreign key
--  it is a column in one table that is a primary key in a different table.

-- JOIN More than Two Tables
SELECT *
    FROM web_events
    JOIN accounts
        ON web_events.account_id = accounts.id
    JOIN orders
        ON accounts.id = orders.account_id;

-- notes
--  we can create a SELECT statement that could pull specific columns from any of the three tables.
--  JOIN holds a table
-- ON is a link for our PK to equal the FK.
SELECT web_events.channel, accounts.name, orders.total
    FROM web_events
        JOIN accounts
            ON web_events.account_id = accounts.id
        JOIN orders
            ON accounts.id = orders.account_id;
```

```sql

-- Questions

/*
    Provide a table for all web_events associated with the account name of Walmart.
    There should be three columns. Be sure to include the primary_poc,
    time of the event, and the channel for each event. Additionally,
    you might choose to add a fourth column to
    assure only Walmart events were chosen.
*/
SELECT accounts.name , accounts.primary_poc , accounts.website ,
        web_events.occurred_at , web_events.channel
    FROM web_events
    JOIN accounts
        ON accounts.id = web_events.account_id
    WHERE accounts.name = 'Walmart';

/*
    Provide a table that provides the region for each sales_rep
    along with their associated accounts.
    Your final table should include three columns: the region name,
    the sales rep name, and the account name.
    Sort the accounts alphabetically (A-Z) according to the account name.
*/
SELECT region.name , sales_reps.name , accounts.name
    FROM sales_reps -- from , it is a table has PK @ 1st step
    JOIN region
        ON region.id = sales_reps.region_id
    JOIN accounts
        ON accounts.sales_rep_id = sales_reps.id
    ORDER BY accounts.name;

/*
    Provide the name for each region for every order,
    as well as
        1.the account name and
        2.the unit price they paid (total_amt_usd/total) for the order.
    Your final table should have 3 columns:
        region name, account name, and unit price.

    >>A few accounts have 0 for total, so I divided by (total + 0.01) to
        assure not dividing by zero.
*/
SELECT accounts.name , (orders.total_amt_usd/(orders.total+0.01)) as unit_price ,
            sales_reps.name
    FROM region
    JOIN sales_reps
        ON sales_reps.region_id = region.id
    JOIN accounts
        ON accounts.sales_rep_id = sales_reps.id
    JOIN orders
        ON accounts.id = orders.account_id;

-- LEFT and RIGHT JOINs

-- Notice
--  1.The first shows JOINs the way you have currently been working with data.
--  2.The second shows LEFT and RIGHT JOIN statements.

-- TYPES OF JOINS

-- 1-equy join
/*
    SELECT <columns_name>
        FROM <table1_name> , <table2_name>
        WHERE <table1_name>.<column_name>(PK) = <table2_name>.<column_name>(FK);
*/
SELECT accounts.id ,orders.id ,accounts.name , orders.total
    FROM orders , accounts
```

```
    WHERE accounts.id = orders.account_id
    ORDER BY accounts.id
    LIMIT 10;

-- 2-INNER join   JOIN
/*
    SELECT <columns_name>
        FROM <table1_name>
            INNER JOIN <table2_name>
            ON <table1_name>.<column_name>(fK) = <table2_name>.<column_name>(pK);
*/
SELECT accounts.id ,orders.id ,accounts.name , orders.total
    FROM orders
    JOIN accounts
        ON accounts.id = orders.account_id
    ORDER BY accounts.id
    LIMIT 10;

SELECT accounts.id ,orders.id ,accounts.name , orders.total
    FROM orders INNER JOIN accounts
        ON accounts.id = orders.account_id
    ORDER BY accounts.id
    LIMIT 10;

-- 3-OUTER join

    -- 3.1-LEFT OUTER join
    /*
        SELECT <columns_name>
            FROM <table1_name>
            LEFT OUTER <table2_name>
                ON <table1_name>.<column_name>(fK) = <table2_name>.<column_name>(pK)
            WHERE <table2_name>.pk is NULL;
    */
    SELECT accounts.id ,orders.id ,accounts.name , orders.total
        FROM orders
        left OUTER JOIN accounts
            ON accounts.id = orders.account_id
        ORDER BY accounts.id
        LIMIT 10;

    SELECT accounts.id ,orders.id ,accounts.name , orders.total
        FROM orders
        left JOIN accounts
            ON accounts.id = orders.account_id
        ORDER BY accounts.id
        LIMIT 10;

    -- 3.1.1-LEFT OUTER join with EXCLUSION
    /*
        SELECT <columns_name>
            FROM <table1_name>
            LEFT OUTER <table2_name>
                ON <table1_name>.<column_name>(fK) = <table2_name>.<column_name>(pK)
                WHERE <table2_name>.pk is NULL;
    */
    SELECT accounts.id ,orders.id ,accounts.name , orders.total
        FROM orders
        left OUTER JOIN accounts
            ON accounts.id = orders.account_id
        WHERE accounts.id is NULL
        ORDER BY accounts.id
        LIMIT 10;

-- RIGHT and FULL OUTER JOINs are not currently supported

-- 3.2-RIGHT OUTER join
    /*
        SELECT <columns_name>
```

```
            FROM <table1_name>
            RIGHT OUTER <table2_name>
                ON <table1_name>.<column_name>(fK) = <table2_name>.<column_name>(pK)
            WHERE <table2_name>.pk is NULL;
    */
    SELECT *
        FROM orders
        RIGHT OUTER JOIN accounts
            ON accounts.id = orders.account_id
        ORDER BY accounts.id
        LIMIT 10;

    SELECT accounts.id ,orders.id ,accounts.name , orders.total
        FROM orders
        RIGHT JOIN accounts
            ON accounts.id = orders.account_id
        ORDER BY orders.id
        LIMIT 10;

    -- 3.2.1-RIGHT OUTER join with EXCLUSION
    /*
        SELECT <columns_name>
            FROM <table1_name>
            RIGHT OUTER <table2_name>
                ON <table1_name>.<column_name>(fK) = <table2_name>.<column_name>(pK)
                WHERE <table2_name>.pk is NULL;
    */
    SELECT accounts.id ,orders.id ,accounts.name , orders.total
        FROM orders
        RIGHT OUTER JOIN accounts
            ON accounts.id = orders.account_id
        WHERE accounts.id is NULL
        ORDER BY accounts.id
        LIMIT 10;

-- RIGHT and FULL OUTER JOINs are not currently supported

-- 3.3-FULL OUTER join
    /*
        SELECT <columns_name>
            FROM <table1_name>
            RIGHT OUTER <table2_name>
                ON <table1_name>.<column_name>(fK) = <table2_name>.<column_name>(pK)
                WHERE <table2_name>.pk is NULL;
    */
    SELECT *
        FROM orders
        LEFT OUTER JOIN accounts
            ON accounts.id = orders.account_id
        WHERE accounts.id is NULL
    UNION
    SELECT accounts.id ,orders.id ,accounts.name , orders.total
        FROM orders
        RIGHT OUTER JOIN accounts
            ON accounts.id = orders.account_id
        WHERE accounts.id is NULL   ;

-- 3.3-FULL OUTER join with exclusion
    /*
        SELECT <columns_name>
            FROM <table1_name>
            RIGHT OUTER <table2_name>
                ON <table1_name>.<column_name>(PK) = <table2_name>.<column_name>(FK);
    */
    SELECT *
        FROM orders
        LEFT OUTER JOIN accounts
            ON accounts.id = orders.account_id
    UNION
```

```sql
    SELECT accounts.id ,orders.id ,accounts.name , orders.total
        FROM orders
        RIGHT OUTER JOIN accounts
            ON accounts.id = orders.account_id ;


-- TESTING
SELECT a.id, a.name, o.total FROM orders o RIGHT JOIN accounts a ON o.account_id = a.id;



-- JOINs and Filtering
/*
    A simple rule to remember is that, when the database executes this query,
    it executes the join and everything in the ON clause first.
    Think of this as building the new result set.
    That result set is then filtered using the WHERE clause.

    The fact that this example is a left join is important.
    Because inner joins only return the rows for which the two tables match,
    moving this filter to the ON clause of an inner join will produce
    the same result as keeping it in the WHERE clause.
*/

SELECT orders.*, accounts.*
    FROM orders
    LEFT JOIN accounts
        ON orders.account_id = accounts.id
    WHERE accounts.sales_rep_id = 321500;

-- Questions
/*
    Q1
    Provide a table that provides the region for each sales_rep along with
    their associated accounts. This time only for the Midwest region.
    Your final table should include three columns:
    the region name, the sales rep name, and the account name.
    Sort the accounts alphabetically (A-Z) according to the account name.
*/
--code
SELECT region.name AS REGION_NAME , sales_reps.name AS SALES_REPS_NAME , accounts.name AS ACCOUNTS_NAME
    FROM sales_reps
    JOIN region
        ON sales_reps.region_id = region.id
    JOIN accounts
        ON accounts.sales_rep_id = sales_reps.id
    WHERE region.name = 'Midwest'
    ORDER BY accounts.name;


/*
    Q2
    Provide a table that provides the region for each sales_rep along with
    their associated accounts. This time only for accounts where
    the sales rep has a first name starting with S and in the Midwest region.
    Your final table should include three columns:
    the region name, the sales rep name, and the account name.
    Sort the accounts alphabetically (A-Z) according to the account name.
*/
--code
SELECT region.name AS REGION_NAME , sales_reps.name AS SALES_REPS_NAME , accounts.name AS ACCOUNTS_NAME
    FROM sales_reps
    JOIN region
        ON sales_reps.region_id = region.id
    JOIN accounts
        ON accounts.sales_rep_id = sales_reps.id
    WHERE region.name = 'Midwest' and sales_reps.name like 'S%'
    ORDER BY accounts.name;

```

```sql

/*
    Q3
    Provide a table that provides the region for each sales_rep along with
    their associated accounts. This time only for accounts where
    the sales rep has a last name starting with K and in the Midwest region.
    Your final table should include three columns:
    the region name, the sales rep name, and the account name.
    Sort the accounts alphabetically (A-Z) according to the account name.
*/
--code
SELECT region.name AS REGION_NAME , sales_reps.name AS SALES_REPS_NAME , accounts.name AS ACCOUNTS_NAME
    FROM sales_reps
    JOIN region
        ON sales_reps.region_id = region.id
    JOIN accounts
        ON accounts.sales_rep_id = sales_reps.id
    WHERE region.name = 'Midwest' and sales_reps.name like '%K%'
    ORDER BY accounts.name;


/*
    Q4
    Provide the name for each region for every order, as well as
    the account name and the unit price they paid (total_amt_usd/total)
    for the order. However, you should only provide the results
    if the standard order quantity exceeds 100.
    Your final table should have 3 columns:
    region name, account name, and unit price.
    In order to avoid a division by zero error,
    adding .01 to the denominator here is helpful total_amt_usd/(total+0.01).
*/
--code
SELECT region.name AS REGION_NAME , accounts.name AS ACCOUNTS_NAME,
        orders.total_amt_usd /(orders.total+0.01)AS UNIT_PRICE
    FROM sales_reps
    JOIN region
        ON sales_reps.region_id = region.id
    JOIN accounts
        ON accounts.sales_rep_id = sales_reps.id
    JOIN orders
        ON orders.account_id = accounts.id
    WHERE orders.standard_qty > 100
    ORDER BY accounts.name;


/*
    Q5
    Provide the name for each region for every order, as well as
    the account name and the unit price they paid (total_amt_usd/total)
    for the order. However, you should only provide the results
    if the standard order quantity exceeds 100 and
    the poster order quantity exceeds 50.
    Your final table should have 3 columns:
    region name, account name, and unit price.
    Sort for the smallest unit price first.
    In order to avoid a division by zero error,
    adding .01 to the denominator here is helpful (total_amt_usd/(total+0.01).
*/
--code
SELECT region.name AS REGION_NAME , accounts.name AS ACCOUNTS_NAME,
         orders.total_amt_usd /(orders.total+0.01)AS UNIT_PRICE
    FROM sales_reps
    JOIN region
        ON sales_reps.region_id = region.id
    JOIN accounts
        ON accounts.sales_rep_id = sales_reps.id
    JOIN orders
```

```sql
        ON orders.account_id = accounts.id
    WHERE orders.standard_qty > 100 AND orders.poster_qty > 50
    ORDER BY UNIT_PRICE;


/*
    Q6
    Provide the name for each region for every order,
    as well as the account name and the unit price they paid (total_amt_usd/total)
    for the order. However, you should only provide the results
    if the standard order quantity exceeds 100 and
    the poster order quantity exceeds 50.
    Your final table should have 3 columns:
    region name, account name, and unit price.
    Sort for the largest unit price first.
    In order to avoid a division by zero error,
    adding .01 to the denominator here is helpful (total_amt_usd/(total+0.01).
*/
--code
SELECT region.name AS REGION_NAME , accounts.name AS ACCOUNTS_NAME,
         orders.total_amt_usd /(orders.total+0.01)AS UNIT_PRICE
    FROM region
    JOIN sales_reps
        ON sales_reps.region_id = region.id
    JOIN accounts
        ON accounts.sales_rep_id = sales_reps.id
    JOIN orders
        ON orders.account_id = accounts.id
    WHERE orders.standard_qty > 100 AND orders.poster_qty > 50
    ORDER BY UNIT_PRICE DESC;


/*
    Q7
    What are the different channels used by account id 1001?
    Your final table should have only 2 columns:
    account name and the different channels.
    You can try SELECT DISTINCT to narrow down the results
    to only the unique values.
*/
--code
SELECT DISTINCT accounts.name , web_events.channel
    FROM accounts
    JOIN web_events
        ON web_events.account_id = accounts.id
    WHERE accounts.id = '1001';


/*
    Q8
    Find all the orders that occurred in 2015.
    Your final table should have 4 columns:
    occurred_at, account name, order total, and order total_amt_usd.
*/
--code
SELECT orders.occurred_at , accounts.name , orders.total , orders.total_amt_usd
    FROM accounts
    JOIN orders
        ON orders.id= accounts.id
    WHERE orders.occurred_at = '2015';

```