```
-- Comments
-- We use comments to add notes to our code.This is a comment and it won’t get
executed.


--                      ----------------------   RETRIEVING DATA FROM SINGLE TABLE
      ----------------------



USE sql_store;



--                            ----------------------  The SELECT Statement
-----------------------
-- SQL is NOT case sensitive but it is a good practice to use upper case for
keywords and lower case for anything else

SELECT  customer _id, first_name FROM customers;


-- if you type the above code, you will see a red squeky line underlying the SELECT
statement and if you hover your mouse on it, you will see it is showing syntax
error.
-- you need to terminate the USE statement with a semi-colon and the error will
disappear



USE sql_store;


-- you can use keyboard shortcut  SHIFT + COMMAND/CONTROL + ENTER


-- you can select all columns

SELECT * FROM customers;


-- you can select specific columns

SELECT  customer_id, first_name FROM customers;

--                                ----------------------- The WHERE Cluase
      ------------------------

-- we can add the where clause to make our query even more specific

SELECT  customer_id, first_name FROM customers WHERE customer_id = 10;



--                                ----------------------- The ORDER BY Cluase
      ------------------------
```

```
SELECT  customer_id, first_name FROM customers WHERE customer_id = 10 ORDER BY
first_name;

-- as we keep expanding our code, the it becomes longer and longer. it is therefore
advisable to put breaks in your code to make it more readable

SELECT  customer_id, first_name
FROM customers
WHERE customer_id = 10
ORDER BY first_name;

-- executing the above will not give us much info since we have only selected one
customer.
-- let's delete the WHERE clause and run again.

SELECT  customer_id, first_name
FROM customers
ORDER BY first_name;

-- NOTE:
-- the three clauses used above must follow the exact order. you cannot interchange
their positions
-- else SQL will give syntax error.
-- the clauses are optional




--                                  ----------------------- The SELECT Cluase in
detail      ------------------------

SELECT  first_name, last_name
FROM customers;

-- we can also interchange the positions of the selected items:

SELECT  last_name , first_name
FROM customers;


-- we can also add some expressions to retrieve data that we want.
-- for example adding extra points to customers' points

SELECT  last_name , first_name , points + 20
FROM customers;

-- let's add the old points for clarity

SELECT  last_name , first_name , points, points + 20
FROM customers;

-- there are several other operators that you can use

-- SELECT  last_name , first_name , points, points (+,-,*,%) 20
-- FROM customers


-- you can also break the SELECT clause for more readability
```

```
SELECT
    last_name,
    first_name,
    points,
    points + 20
FROM
    customers;

-- NB: the order of operators in here follows the same logic as in math (i.e
BODMAS)
-- it is better to use brackets to tell SQL the order of operation you are looking
for

SELECT
    last_name,
    first_name,
    points,
    (points * 20) - 5
FROM
    customers;


    -- Aliasing the column names

    -- if you notice, the column names are given as the operation we performed
which does not look good or even make sense.
    -- we can aliase it or rename it to a more meaningful one

SELECT
    last_name,
    first_name,
    points,
    (points * 20)- 5, -- am leaving this here so you see the difference
    (points * 20) - 5 AS discount_factor -- if want to have a space in the colum
name, you will need to add quotes e.g. 'discount factor'
FROM
    customers;

-- we can use the asterisks to extract all columns instead of being specific

SELECT *,
    (points * 20)- 5, -- am leaving this here so you see the difference
    (points * 20) - 5 AS discount_factor -- if want to have a space in the colum
name, you will need to add quotes e.g. 'discount factor'
FROM
    customers;



                              -- Dealing with duplicates with the SELECT clause

SELECT state
FROM customers;

-- let's first intensionally create some duplicates in the state table.
-- go the customers table and double click to open it > double click on the state
of MA and manually change it to VA.
-- click on apply at the bottom right corner to make changes effective
-- now let's execute our query again
```

```
SELECT state
FROM customers;

-- let's say we want to only unique states where our customers are located.

SELECT DISTINCT state
FROM customers;

-- EXERCISE

-- Write an SQL query to return:
--                      1. all the products in the database
--                      2. name
--                      3. unit price
--                      2. discounted price of 5%


-- SAMPLE SOLUTION:

SELECT
            name,
            unit_price,
            unit_price - (unit_price * 0.05)  AS 'discounted price'
FROM products;

-- you can add a round() function to round it to less significant figures.

SELECT
            name,
            unit_price,
            round (unit_price - (unit_price * 0.05), 2)  AS 'discounted price'
FROM products;




--                                  ---------------------- The WHERE Cluase in
detail      ----------------------
-- we use the where clause to filter data

-- e.g. we want customers with points greater than 1000

SELECT *
FROM customers
WHERE points > 1000;

-- the > sign is one of SQL's comparison operator.
-- below is the list of most popular SQL comparison operators

-- >
-- >=
-- <
-- <=
-- =
-- != or <>
```

```sql
SELECT *
FROM customers
WHERE points = 947;

-- ---

SELECT *
FROM customers
WHERE state = 'VA';

-- ---
SELECT *
FROM customers
WHERE birth_date > '1980-01-02'; -- in SQL we need to add quotes to dates


-- EXERCISE

-- Write an SQL query to return the orders that were made before 2019

-- SAMPLE SOLUTION:
SELECT *
FROM orders
WHERE order_date < '2019-01-01';




--                                  ----------------------- The AND, OR and NOT
operators   ----------------------

-- -- AND


SELECT *
FROM customers
WHERE birth_date > '1980-01-02' AND points >= 1500;

-- -- OR

SELECT *
FROM customers
WHERE birth_date > '1980-01-02' OR points >= 1500;


-- -- combine AND and OR with operator sign


SELECT *
FROM customers
WHERE birth_date > '1990-01-02' OR points >= 1500 AND state = 'VA';

-- NB: SQL gives preference to the AND operator before the OR operator
-- so in our example, the AND part first gets executed before the OR part


-- NOT
-- we can use the NOT operator to perform exclusive selections.
```

```
-- that is, to select anything else apart from the condition

SELECT *
FROM customers
WHERE NOT (birth_date > '1990-01-02' OR points > 1500 );
-- now we can see that if we appy the NOT to negate the items in the bracket we get
the same results as shown below:
SELECT *
FROM customers
WHERE birth_date <= '1990-01-02' AND points <= 1500 ;


-- EXERCISE

-- From the order_items table , get the items:
-- for oder number 6
-- where the total price is greater than $20
-- and display the unit price

-- SAMPLE SOLUTION:
SELECT
order_id, product_id, quantity, unit_price,
unit_price* quantity AS total_price
FROM order_items
WHERE order_id = 6 AND unit_price * quantity > 20;

-- IN
-- let's say we waant to locate all the customers that are located in say Florida
or Virginia, etc
SELECT *
FROM customers
WHERE state = 'VA' OR state = 'FL' OR state = 'GA';

-- the above code works fine but it becomes very tedious to write especially if
your dataset is huge
-- you want to be smart and write the following:

SELECT *
FROM customers
WHERE state = 'FL' OR 'VA' OR  'GA'; -- but this will not give you what you want.

-- Alternativerly we can use the IN clause to simplify this

SELECT *
FROM customers
WHERE state IN ( 'FL' , 'VA' ,'GA'); -- the order does not matter

-- we can also use the NOT operator to select data that are not in the specified
states

SELECT *
FROM customers
WHERE state NOT IN ( 'FL' , 'VA' ,'GA'); -- the order does not matter

-- EXERCISE

-- Return products with
      -- quantity in stock equal to 49, 38, 72
```

```
-- SAMPLE SOLUTION:


SELECT *
FROM products
WHERE quantity_in_stock IN (49, 38, 72 ); -- the order does not matter


-- BETWEEN
-- let's say we want to find out all the customers with points greater than or
equal 1000 but less than or equal to 3000
-- we can write something like the following:

SELECT *
FROM customers
WHERE points >= 1000 AND points <= 3000;

-- we can rather use the BETWEEN operator to make our code shorter and cleaner:

SELECT *
FROM customers
WHERE points BETWEEN 1000 AND 3000;


-- EXERCISE

-- Return CUSTOMERS born
      -- between 1/1/1990 and 1/1/2000

-- SAMPLE SOLUTION:


SELECT *
FROM customers
WHERE birth_date BETWEEN '01/01/1990' AND '01/01/2000'; -- this give me nothing
because that is not how our date is formatted

--

SELECT *
FROM customers
WHERE birth_date BETWEEN '1990/01/01' AND '2000/01/01';


-- --LIKE
-- let's say we want only customers whose last name starts with b

SELECT *
FROM customers
WHERE last_name LIKE 'b%'; -- we use the % sign to indicate any number of
characters can come after b, but first it should start with b, it doesn't matter if
uppercase or lower case b

-- we can also get customers whose last name starts with say 'brush'

SELECT *
FROM customers
WHERE last_name LIKE 'brush%';
```

```
-- -- let's say we want only customers whose last name contains 'b'

SELECT *
FROM customers
WHERE last_name LIKE '%b%';

-- let's say we want only customers whose first name ends with 't'

SELECT *
FROM customers
WHERE first_name LIKE '%y';

-- let's check for last name
SELECT *
FROM customers
WHERE last_name LIKE '%y';

-- we can also use the underscore _ to get exactly the number of characters long we
want

SELECT *
FROM customers
WHERE last_name LIKE '_y'; -- meaning we want customers whose last name is exactly
2 characters and ends with 'y'

--
SELECT *
FROM customers
WHERE last_name LIKE '_____y' ;-- meaning we want customers whose last name is
exactly 5 characters and ends with 'y'

--

SELECT *
FROM customers
WHERE last_name LIKE 'b____y' ;-- meaning we want customers whose last name starts
with 'b' and has exactly 4 characters in between and also ends with 'y'
--

-- EXERCISE

-- Get the customers whose
      -- 1. addresses contain TRAIL or AVENUE
      -- 2. phone numbers end with 9

-- SAMPLE SOLUTION:
-- 1. addresses contain TRAIL or AVENUE

SELECT *
FROM customers
WHERE address LIKE '%trail%' OR
                   address LIKE '%avenue%';


      -- 2. phone numbers end with 9

SELECT *
FROM customers
WHERE phone LIKE '%9';
```

```
-- we can also add the NOT operator to selector all customers whose phone number
does not end with 9
SELECT *
FROM customers
WHERE phone NOT LIKE '%9';


-- REGEXP
-- let's say we want customers who have the word 'field' in their last name

SELECT *
FROM customers
WHERE last_name LIKE '%field%' ;

-- we also have an alternative way of doing this using regular expression (REGEXP)
-- REGEXP helps us to do even more complex queries

SELECT *
FROM customers
WHERE last_name REGEXP 'field' ; -- same results as above

-- we can use ^ to indicate a start of an expression

SELECT *
FROM customers
WHERE last_name REGEXP '^field' ; -- we do not have any name that start with
'field' so we won't get any results for this.

-- we can also use a $ sign to indicate the end of a string

SELECT *
FROM customers
WHERE last_name REGEXP 'field$' ; -- showing customers with 'field at the end

-- let's we want to search for multiple words, we can use the pipe sign '|'

SELECT *
FROM customers
WHERE last_name REGEXP 'field|mac' ;  -- make sure there no spaces included

--

SELECT *
FROM customers
WHERE last_name REGEXP 'field|mac|rose' ;  -- make sure there no spaces included

-- let's say want all custmers whose have 'e' in their last name, and before the
letter 'e', there should be

SELECT *
FROM customers
WHERE last_name REGEXP '[gim]e' ; -- it matches with any customer have 'ge', 'ie',
or 'me' in their last name

-- --

SELECT *
FROM customers
```

```
WHERE last_name REGEXP 'e[gim]' ; -- it matches with any customer have 'eg', 'ei',
or 'em' in their last name

-- we can also provide range of characters

SELECT *
FROM customers
WHERE last_name REGEXP '[a-h]e' ;

-- summary of REGEXP:
-- ^ for beginning of a string
-- $ to indicate end of a string
-- | logical or
-- [a,b,c,d] to indicate combination of characters before or after a letter
-- [a-m]      to indicate combination of characters before or after a letter


-- EXERCISE

-- Get the customer whose
      -- first names are ELKA or AMBUR
    -- last names end with EY or ON
    -- last names starts with MY or contains SE
    -- last names contain B followed by R or U

-- SOLUTION
-- first names are ELKA or AMBUR

SELECT *
FROM customers
WHERE first_name REGEXP 'elka|ambur';

 -- last names end with EY or ON

SELECT *
FROM customers
WHERE last_name REGEXP 'ey$|on$';

-- last names starts with MY or contains SE

SELECT *
FROM customers
WHERE last_name REGEXP '^my|se';

-- last names contain B followed by R or U

SELECT *
FROM customers
WHERE last_name REGEXP 'b[r|u]';

-- alternatively

SELECT *
FROM customers
WHERE last_name REGEXP 'br|bu';
```

```
-- IS NULL operator
-- sometimes you want to know if there are records that are not available and the
IS NULL operator comes in handy.

SELECT *
FROM customers; -- we can observe that customer no. 5 does not have phone number
-- let's say we want to find all the customers who do not have phone

SELECT *
FROM customers
WHERE phone IS NULL;
-- we can also use the IS NOT NULL to do the opposite

SELECT *
FROM customers
WHERE phone IS NOT NULL;

-- EXERCISES
      -- get all the orders that are not shipped


-- SOLUTION
-- if you look into the orders table you will realise that there are lots of
shipped_date and shipper_id

SELECT *
FROM orders
WHERE shipper_id IS NULL;

-- alternatively

SELECT *
FROM orders
WHERE shipped_date IS NULL;


-- The ORDER BY clause
-- normally, our dataset is sorted by the ID since it is the primary key, we can
also explicitly order the table by any other column

SELECT *
FROM customers;
--
SELECT *
FROM customers
ORDER BY first_name;

-- we can also add the DESC clause
SELECT *
FROM customers
ORDER BY first_name DESC;
--
SELECT *
FROM customers
ORDER BY state, first_name;
--
SELECT first_name, last_name
FROM customers
ORDER BY 1,2; -- this becomes a problem especially when u add a new column
```

```
-- LIMIT Clause
-- you can limit the number of reports with the LIMIT clause. e.g. get the first 5
customers

SELECT *
FROM customers
LIMIT 5;

-- we can also set an offset to for conditional selection. e.g skipping the first 5
items and selecting the last 3 items

SELECT *
FROM customers
LIMIT 5,3;

-- EXERCISE
      -- get the top 3 loyal customers

SELECT *
FROM customers
ORDER BY points DESC
LIMIT 3;



--          ----------------------- Retrieving Data From Multiple Tables
-----------------------

--                                        ----------------------- INNER JOIN
-----------------------

-- let's say we want to select the orders from the orders table and instead of
showing the orderID, we want to show the particular customer that ordered the item

SELECT *
FROM orders
JOIN customers -- we can add INNER but that is optional
ON orders.customer_id  = customers.customer_id;

-- we can see from the output that the orders table comes first and then followed
by the customers table
-- we can simplify this further:

SELECT  order_id, first_name, last_name
FROM orders
JOIN customers -- we can add INNER but that is optional
ON orders.customer_id  = customers.customer_id;

-- if we try to add the customer ID we will get an error, that is because SQL is
not sure which table to select customer ID from. We can get around that by
specifying that as follows:

SELECT  order_id, customer_id, first_name, last_name
FROM orders
JOIN customers -- we can add INNER but that is optional
ON orders.customer_id  = customers.customer_id;
```

```
--

SELECT  order_id, orders.customer_id, first_name, last_name
FROM orders
JOIN customers -- we can add INNER but that is optional
ON orders.customer_id  = customers.customer_id;

-- we can also give an alias to the names: orders and customers to avoid them being
repeated often.alter

SELECT  order_id, o.customer_id, first_name, last_name
FROM orders o
JOIN customers c
ON o.customer_id  = c.customer_id;

-- NOTE: once you create an alias, you need to use that alias throughout , you
can't alias orders to 'o' and still want to use the name orders instead of the
alias'o'

-- EXERCISE
-- write a query to join the order_items table with the products table and return
            -- product ID
        -- product name
        -- quantity
        -- unit price
-- NB: use alias to simplify your code.alter

-- SOLUTION

 SELECT order_id, oi.product_id, quantity, oi.unit_price -- we select unit price
from the order_items since it is the current price at which the product is being
purchased
 FROM order_items oi
 JOIN products p
 ON oi.product_id = p.product_id;



 --         ----------------  JOINING Across Databases ----------------

 -- more often you may have to work acorss multiple tables.-
 -- let's see how we can combine columns from tables in multiple places.
 -- if u go into the sql_inventory databse, we have products table there which is
same as the
 -- products table under sql_store database. Ideally, this is not a good database
design to have repeated tables like that.
 -- but for the learning let's imagine it has happen.
 -- what we are going to do now is to join the order items table in the sql_store
database to the products table in the sql_inventory database.

 SELECT *
 FROM order_items oi
 JOIN sql_inventory.products p
 ON oi.product_id = p.product_id


 -- Important note:
 -- you need to prefix the table that is not in the current database, e.g. in the
above code, we run it in the sql_store database so we had to prefix the products
```

with sql_inventory.products since we selected product table that is not in the active database(sql_store)

```
 --         ----------------  SEFL JOIN ----------------

-- we can also join a table with itself
-- if we look at the sql_hr database, we can find an employee table there.
-- each employee has their manager or a one they report to. we can see that even one employee(37270) has not manager, meaning he/she is the CEO of the company.
-- now we want to see the name of each employee and their manager.

USE sql_hr;

SELECT *
FROM employees e
JOIN employees m -- we are using a different letter here since it is the same table, and we prefer m to represent managers
ON e.reports_to = m.employee_id;
-- we can observe that we have only one manager that everyone reports to and that manager does not report to anyone so his report to column is having null.
-- we can simplify the table:

SELECT
           e.employee_id,
        e.first_name,
        m.first_name
FROM employees e
JOIN employees m
ON e.reports_to = m.employee_id


 --         ----------------  JOINING ACROSS MULTIPLE TABLES ----------------

 -- if we look at the orders table in the sql_store databse, we notice that we have status column there,
 -- however, the meaning of those status can be found in the order_status table.
 -- now we want to write a query to join the orders table with two other tables. i.e. the customers table and order_status table
 -- we want the order_id, order_date, first_name, last_name, status

 USE sql_store;

 SELECT *
 FROM orders o
 JOIN customers c
                  ON o.customer_id = c.customer_id
JOIN order_statuses os
                  ON o.status = os.order_status_id

-- nb: in the real world you can even join 10 tables so this is not uncommon, it happens.

-- now we get information from the orders table followed by info from the customers table and lastly from the order status table

-- let's simplify the results by selecting few columns
```

```
 SELECT
                  o.order_id,
            o.order_date,
            c.first_name,
            c.last_name,
            os.name AS status
 FROM orders o
 JOIN customers c
                  ON o.customer_id = c.customer_id
JOIN order_statuses os
                  ON o.status = os.order_status_id


-- EXERCISE
-- write a query and join the payments table with the payment_methods table as well
as the clients
   -- producce a report that shows the payment with more details such as the name
of the client, and the payment method.
   -- nb: the tables are found in the sql_invoicing database


   -- SOLUTION

   USE sql_invoicing;

   SELECT *
   FROM payments p
   JOIN clients c
                  ON p.client_id = c.client_id
   JOIN payment_methods pm
                  ON p.payment_method = pm.payment_method_id;

-- you can modify the code by selecting only the necessary columns

   SELECT
                  p.date,
            p.invoice_id,
            p.amount,
            c.name,
            pm.name AS payment_method
   FROM payments p
   JOIN clients c
                  ON p.client_id = c.client_id
   JOIN payment_methods pm
                  ON p.payment_method = pm.payment_method_id;



 --         ----------------  OUTER JOIN  ----------------

 -- we will first create inner join and convert to an outer join
 USE sql_store;

 SELECT
            c.customer_id,
        c.first_name,
        o.order_id
FROM customers c
JOIN orders o
```

```
            ON c.customer_id = o.customer_id
ORDER BY c.customer_id

-- we can see from the output that we have results for only customers that have
orders, meanwhile if you take a look at the customer table, you will realise there
other customers as well.->
-- we want to see all the customers, whether they have an order or not, that's when
we need an outer join

-- LEFT OUTER JOIN

 SELECT
            c.customer_id,
        c.first_name,
        o.order_id
FROM customers c
LEFT JOIN orders o -- left join returns all the customers whether they have orders
or not(i.e whether the ON condition is true or not)
            ON c.customer_id = o.customer_id
ORDER BY c.customer_id

-- we can see from the output that all the customers are returned including those
with no orders

-- RIGHT JOIN

 SELECT
            c.customer_id,
        c.first_name,
        o.order_id
FROM customers c
RIGHT JOIN orders o -- RIGHT join returns all the orders (i.e whether the ON
condition is true or not)
            ON c.customer_id = o.customer_id
ORDER BY c.customer_id

-- if you want to see all the cusomers, you can swap the position of the tables

 SELECT
            c.customer_id,
        c.first_name,
        o.order_id
FROM orders o
RIGHT JOIN customers c -- you can add the keyword OUTER but that is optional
ORDER BY c.customer_id

-- EXERCISE
-- write a query that produces a table with the following results:
            -- product_id, name(i.e. name of product), quantity(you can get that
from the order items table)
            -- return the product even if it has never been ordered

-- SOLUTION

USE sql_store;

SELECT *
FROM products p
LEFT JOIN order_items oi
```

```
      ON    p.product_id = oi.product_id

-- you can select the required columns

SELECT
      p.product_id,
    p.name,
    oi.quantity
FROM products p
LEFT JOIN order_items oi
      ON    p.product_id = oi.product_id;


 --         ----------------  The USING clause  ----------------
 -- using CLAUSE helps us to simplify the ON condition


 SELECT
           o.order_id,
        c.first_name
FROM orders o
JOIN customers c
--           ON o.customer_id = c.customer_id
           USING (customer_id); -- we get the same results as above ON condition

-- we can even add more queries

 SELECT
           o.order_id,
        c.first_name,
        sh.name AS Shipper
FROM orders o
JOIN customers c
           USING (customer_id)
LEFT  JOIN shippers sh
           USING (shipper_id);

-- we have all the customers alongside with their shippers and order id
-- NB: the USING keyword only works if the column name that you want to join are
same. so you cannot join say order_id with order_status_id

-- EXERCISE
      -- using the sql_invoicing database, write a query that returns:
        -- date
        -- amount
        -- client
        -- name (i.e name of payment menthod)

-- SOLUTION

USE sql_invoicing;

SELECT
       p.date,
       p.amount,
       c.name AS client,
       pm.name AS payment_method

FROM payments p
```

```
JOIN clients c USING (client_id)
JOIN payment_methods pm
            ON p.payment_method = payment_method_id



 --         -----------------  Inserting, Updating, and Deleting Data
-----------------
```