Congratulations! You passed!

Grade received 100% To pass 80% or higher

Go to next item

Final Exam

Latest Submission Grade 100%

1. 1 / 1 point

Question: Modify a table

Modify the columns in table `sales` to match the provided schema

Given the schema:

column	type
datetime	BIGINT
disc_code	INT
product_code	STRING
unit_cost	STRING
unit_price	STRING

Select the SQL query that achieves the following:

- Selects columns `datetime` , `unit_price` , and `disc_code`
- Converts `datetime` column to `TIMESTAMP` ; rename as `datetime`
- Converts `unit_price` to `FLOAT`
- Writes results to a table named `discounts`

(•)

```
--OPTION
DROP TABLE IF EXISTS discounts;
CREATE TABLE discounts AS
SELECT
  to_timestamp(datetime) as datetime,
  disc_code,
  CAST(unit_price AS FLOAT)
FROM
  sales;
SELECT
  *
FROM
  discounts
```

( )

```
--OPTION
CREATE
OR REPLACE TEMPORARY VIEW discounts AS
SELECT
  to_date(datetime) as datetime,
  disc_code,
  CAST(unit_price AS FLOAT)
FROM
  sales;
SELECT
  *
FROM
  discounts
```

( )

```
--OPTION
CREATE
OR REPLACE TEMPORARY VIEW discounts AS
SELECT
  to_timestamp(time) as datetime,
  disc_code,
  unit_price AS FLOAT
FROM
  sales;
SELECT
  *
FROM
```

```
13  discounts
```

○

```
1   --OPTION
2   DROP TABLE IF EXISTS discounts;
3   CREATE TABLE discounts AS
4   SELECT
5     to_date(datetime),
6     disc_code,
7     unit_price AS FLOAT
8   FROM
9     sales;
10  SELECT
11    *
12  FROM
13    discounts;
```

Correct

2. 1 / 1 point

Question: Basic Math

Create a new column for the table, `customers`, by applying basic math operations to an existing column.

Given the schema:

column	type
cust_first_name	STRING
cust_last_name	STRING
cust_city	STRING
cust_state	STRING
cust_country_id	STRING
cust_income	BIGINT
cust_credit_lim	BIGINT
overdue	STRING

Select the SQL query that acheives the following:

- Includes columns: `cust_first_name`, `cust_last_name`, `cust_city`, `cust_state`, `overdue`, and `overdue_charge`
- Converts the `overdue` column to `BOOLEAN`
- Creates a new column `overdue_charge` which should show 2% of the `cust_credit_lim` column, rounded to the nearest cent
- Stores results to a temporary view named `overdue_payments`

◉

```
1   --OPTION
2   CREATE OR REPLACE TEMPORARY VIEW overdue_payments AS
3   SELECT cust_first_name,
4     cust_last_name,
5     cust_city,
6     cust_state,
7     CAST(overdue AS BOOLEAN),
8     ROUND(cust_credit_lim * .02, 2) AS overdue_charge
9   FROM customers;
10
11  SELECT * FROM overdue_payments;
```

○

```
1   --OPTION
2   CREATE OR REPLACE TEMPORARY VIEW overdue_payments AS
3   SELECT cust_first_name,
4     cust_last_name,
5     cust_city,
6     cust_state,
7     overdue AS BOOLEAN,
8     ROUND(cust_credit_lim * .02) AS overdue_charge
9   FROM customers;
10
11  SELECT * FROM overdue_payments;
```

○

```
--OPTION
CREATE OR REPLACE TEMPORARY VIEW overdue_payments AS
SELECT cust_first_name,
  cust_last_name,
  cust_city,
  cust_state,
  CAST(overdue AS BOOLEAN),
  cust_credit_lim * .02
FROM customers;

SELECT * FROM overdue_payments;
```

○

```
--OPTION
CREATE OR REPLACE TEMPORARY VIEW overdue_payments AS
SELECT cust_first_name,
  cust_last_name,
  cust_city,
  cust_state,
  CAST(overdue) BOOLEAN,
  cust_credit_lim * .02
FROM customers;

SELECT * FROM overdue_payments;
```

✓ Correct

3. 1 / 1 point

Question: Define a table with schema

Write a table, `restaurant_reviews`, and define its schema in the creation statment.

Select the table creation statement that achieves the following:

- Uses the path: `/mnt/training/restaurants/reviews.csv`
- Allows the first line to be read as a header
- Specifies that that table is created using `csv`
- Creates a table with the following schema:

column	type
restaurant_id	STRING
user_id	INT
rating	FLOAT
text_review	STRING
time_recorded	INT

○

```
--OPTION
DROP TABLE IF EXISTS restaurant_reviews;
CREATE TABLE restaurant_reviews (
  restaurant_id,
  user_id,
  rating,
  text_review,
  time_recorded
) USING csv OPTIONS (
  path "mnt/training/restaurants/reviews.csv",
);
```

◉

```
--OPTION
```

```
--OPTION
DROP TABLE IF EXISTS restaurant_reviews;
CREATE TABLE restaurant_reviews (
  restaurant_id STRING,
  user_id INT,
  rating FLOAT,
  text_review STRING,
  time_recorded INT
) USING csv OPTIONS (
  path "/mnt/training/restaurants/reviews.csv",
  header "true"
);
```

○

```
--OPTION
CREATE OR REPLACE TEMPORARY VIEW restaurant_reviews (
  restaurant_id STRING,
  user_id INT,
  rating FLOAT,
  text_review STRING,
  time_recorded INT
) USING csv OPTIONS (
  path "/mnt/training/restaurants/reviews.csv",
  header "true"
);
```

○

```
--OPTION
CREATE OR REPLACE TEMPORARY VIEW restaurant_reviews (
  restaurant_id,
  user_id,
  rating,
  text_review,
  time_recorded
) USING parquet OPTIONS (
  path "mnt/training/restaurants/reviews.csv",
  header "true"
);

```

✓ Correct

4. 1 / 1 point

Question: Remove duplicates and sort

Deduplicate and sort restaurant review information from the table, `restaurant_evals`

Given the following schema:

column	type
restaurant_id	STRING
user_id	INT
rating	DOUBLE
text_review	STRING
time_recorded	BIGINT

Select the query that achieves the following:

- Removes duplicate rows
- Casts `time_recorded` as a timestamp. This value can be converted from unixtime
- Sorts rows by `rating` in descending order, and then by `time_recorded` in ascending order
- Diplays **only** columns `restaurant_id` , `user_id` , `rating` , `text_review` , `time`

◉

```
--OPTION
SELECT
  DISTINCT *
FROM
  (
    SELECT
      restaurant_id,
      user_id,
      rating
```

```
    text_review,
    from_unixtime(time_recorded) as time
  FROM
    restaurant_evals
)
ORDER BY
  rating DESC,
  time ASC
```

○

```
--OPTION
SELECT
  DISTINCT *
FROM
  (
    SELECT
      restaurant_id,
      user_id,
      rating,
      text_review,
      from_unixtime(time_recorded) as time
    FROM
      restaurant_evals
  )
ORDER BY
  rating
  time
```

○

```
--OPTION
SELECT
  DISTINCT *,
  from_unixtime(time_recorded) as time
FROM
  restaurant_evals
ORDER BY
  rating DESC,
  time ASC
```

○

```
--OPTION
SELECT
  DISTINCT *,
  from_unixtime(time_recorded) as time
FROM
  restaurant_evals
ORDER BY
  rating,
  time
```

✓ Correct

5. 1 / 1 point

Question: Limit Results

Return the top 10 results for data that matches the given criteria.

Given the schema:

column	type
employee_id	STRING
time_recorded	TIMESTAMP
net_gain	DOUBLE

Write a SQL query on the table **`q1_sales`** that achieves the following:

- Includes the columns: `employee_id`, `closing_date`, and `net_gain`
- Casts **`time_recorded`** column as date and renames to **`closing_date`**
- Sorts rows by the **`net_gain`** column in descending order
- Limits the results to the top 10 **`net_gain`**

○

```
--OPTION
SELECT
  employee_id,
  to_date(time_recorded) closing_date,
  net_gain
FROM
  q1_sales
ORDER BY
  net_gain DESC
```

○

```
--OPTION
SELECT
  employee_id,
  to_date(time_recorded),
  net_gain
FROM
  q1_sales
ORDER BY
  net_gain DESC
LIMIT
  10
```

○

```
--OPTION
SELECT
  *
FROM
  q1_sales
ORDER BY
  net_gain DESC
LIMIT
  10
```

◉

```
--OPTION
SELECT
  employee_id,
  to_date(time_recorded) closing_date,
  net_gain
FROM
  q1_sales
ORDER BY
  net_gain DESC
LIMIT
  10
```

Correct

6. 1 / 1 point

Question: Join Tables

Perform a join on two tables `games` and `goals`.

Given the schemas:

orders

column	type
order_id	INT
order_date	DATE
cust_id	STRING

customers

column	type
cust_id	STRING
cust_name	STRING
contact_name	STRING
country	STRING

Select the query that achieves the following:

- Perfoms an inner join on `orders` and `customers` on the column `cust_id`
- Includes **only** the following columns: `cust_id`, `order_date`, and `cust_name`

◉

```
--OPTION
SELECT
  orders.cust_id,
  order_date,
  cust_name
FROM
  orders
  JOIN customers ON orders.cust_id = customers.cust_id
```

○

```
--OPTION
SELECT
  orders.cust_id,
  order_date,
  cust_name
FROM
  orders
  JOIN customers ON cust_id = customers.cust_id
```

○

```
--OPTION
SELECT
  cust_id,
  order_date,
  cust_name
FROM
  orders
  JOIN customers ON cust_id = cust_id
```

○

```
--OPTION
SELECT
  cust_id,
  order_date,
  cust_name
FROM
  orders
  JOIN customers ON cust_id = customers.cust_id
```

✓ Correct

7. 1 / 1 point

Question: Union Tables

Perform a union of two tables `q1_earnings` and `q2_earnings`

Assume that these two tables have the same schema.

Select the query that achieves the following:

- Includes all records from both tables, including any duplicates

○

```sql
--OPTION
SELECT *
FROM q1_earnings
UNION JOIN
SELECT *
FROM q2_earnings
```

○

```sql
--OPTION
SELECT DISTINCT *
FROM q1_earnings
UNION ALL
SELECT *
FROM q2_earnings
```

◉

```sql
--OPTION
SELECT *
FROM q1_earnings
UNION ALL
SELECT *
FROM q2_earnings
```

○

```sql
--OPTION
SELECT *
FROM q1_earnings
UNION
SELECT *
FROM q2_earnings
```

✓ Correct

8. 1 / 1 point

Question: Extract year and month

Extract the year and month from the `timestamp` field in the table `timetable1`

Given the schema:

column	type
time	timestamp
rating	DOUBLE

Select the SQL query that achieves the following:

- Extracts the year and month from the timestamp
- Includes columns `year`, `month`, and `rating` in the results

◉

```sql
--OPTION
SELECT
  month(time) month,
  day(time) day,
  rating
FROM
  timetable1
```

○

```
--OPTION
SELECT
  CAST(time AS month) month,
  CAST(time AS day) day,
  rating
FROM
  timetable1
```

○

```
--OPTION

SELECT time "MM",
time "dd",
rating
FROM timetable1
```

○

```
--OPTION
SELECT month(time),
day(time),
rating
FROM timetable1
```

Correct

9. 1 / 1 point

Question: Extract day of week

Create a new `day` column from the given table, `sales`.

Given the schema:

column	type
time	TIMESTAMP
purchase_amt	DOUBLE
sale_id	INT

Select the query that achieves the following:

- Creates a new column, `day`, that contains the day of the week associated with the timestamp. This column should contain the name of the day, e.g. Sunday, Monday, Tuesday.
- Returns three columns: `day`, `purchase_amt`, and `sale_id`

☐

```
--OPTION
SELECT
  date_format(time, "E"),
  purchase_amt,
  sale_id
FROM
  sales
```

☐

```
--OPTION
SELECT
  day(to_date(time), "E") day,
  purchase_amt,
  sale_id
FROM
  sales
```

- [x]

```
--OPTION
SELECT
  date_format(to_date(time), "E") day,
  purchase_amt,
  sale_id
FROM
  sales
```

Correct

- [x]

```
--OPTION
SELECT
  date_format(time, "E") day,
  purchase_amt,
  sale_id
FROM
  sales
```

Correct

10. 1 / 1 point

Question: Minimum Function

Display the minimum `total_rev` for each `category` in the table `feb_sales`.

Given the schema:

column	type
item_id	INT
category	STRING
sale_price	DOUBLE
quantity	INT

Select the query that achieves the following:

- Creates a new column `total_revenue` by multiplying `sale_price` by the `quantity`, rounded to the nearest cent.
- Displays the minimum `total_revenue` in each `category`. Label that column `min_revenue`
- Displays the sum of all `total_revenues` in each `category`. Label that column `category_sum`
- Includes columns: `category`, `category_sum`, `min_revenue`

- [x]

```
--OPTION
WITH total_rev AS (
  SELECT
    *,
    ROUND(sale_price * quantity, 2) total_revenue
  FROM
    feb_sales
)
SELECT
  category,
  ROUND(SUM(total_revenue), 2) category_sum,
  MIN(total_revenue) min_revenue
FROM
  total_rev
GROUP BY
  category
```

Correct

- [ ]

```
--OPTION
SELECT
```

```sql
  category,
  ROUND(SUM(total_revenue), 2) category_sum,
  MIN(total_revenue) min_revenue
FROM
  (
    SELECT
      *,
      ROUND(sale_price * quantity, 2) total_revenue
    FROM
      feb_sales
  )
GROUP BY
  category,
  total_revenue
```

☑

```sql
--OPTION
SELECT
  category,
  ROUND(SUM(total_revenue), 2) category_sum,
  MIN(total_revenue) min_revenue
  FROM (
    SELECT *,
    ROUND(sale_price * quantity, 2) total_revenue
    FROM feb_sales
  )
  GROUP BY category
```

Correct

☐

```sql
--OPTION
SELECT
  category,
  ROUND(SUM(total_revenue), 2) category_sum,
  MIN(total_revenue) min_revenue
FROM
  (
    SELECT
      *,
      ROUND(sale_price * quantity, 2) total_revenue
    FROM
      feb_sales
  )
```

11. **Question: Maximum Function** 1 / 1 point

Display the maximum `total_revenue` for each `category` and `item_id` in the table `feb_sales`.

Given the schema:

column	type
item_id	INT
category	STRING
sale_price	DOUBLE
quantity	INT

Select the query that achieves the following:

- Creates a new column `total_revenue` by multiplying `sale_price` by the `quantity`, rounded to the nearest cent.
- Display the maximum `total_revenue` for each `category` and `item_id`. Label that column `max_revenue`.

◉

```sql
--OPTION
SELECT
  category,
  item_id,
  ROUND(SUM(total_revenue), 2) category_sum,
  MAX(total_revenue) max_revenue
  FROM (
    SELECT *,
    ROUND(sale_price * quantity, 2) total_revenue
    FROM feb_sales
  )
  GROUP BY category, item_id
```

○

```sql
--OPTION
SELECT
  category,
  item_id,
  ROUND(SUM(total_revenue), 2) category_sum,
  MAX(total_revenue) max_revenue
  FROM (
    SELECT *,
    ROUND(sale_price * quantity, 2) total_revenue
    FROM feb_sales
  )
  GROUP BY category
```

○

```sql
--OPTION
SELECT
  category,
  item_id,
  ROUND(SUM(total_revenue), 2) category_sum,
  MAX(total_revenue) max_revenue
  FROM (
    SELECT *,
    ROUND(sale_price * quantity, 2) total_revenue
    FROM feb_sales
   )

```

○

```sql
--OPTION
WITH total_rev AS (
  SELECT
    *,
    ROUND(sale_price * quantity, 2) total_revenue
  FROM
    feb_sales
)
SELECT
  category,
  ROUND(SUM(total_revenue), 2) category_sum,
  MAX(total_revenue) max_revenue
FROM
  total_rev
GROUP BY
  category item_id
```

Correct

12. 1 / 1 point

Question: Average Function

Display the average `total_revenue` for each `category` in the table `feb_sales`.

Given the schema:

column	type
item_id	INT
category	STRING
sale_price	DOUBLE
quantity	INT

Select the query that achieves the following:

- Creates a new column `total_revenue` by multiplying `sale_price` by the `quantity`, rounded to the nearest cent.
- Displays the average `total_revenue` in each `category`. Label that column `avg_revenue`.
- Displays the standard deviation for `avg_revenue`. Label that column `std_deviation`

○

```sql
--OPTION
SELECT
  category,
  ROUND(STD(total_revenue), 3) std_deviation,
  ROUND(AVG(total_revenue),2) avg_revenue
FROM
  (
    SELECT
      *,
      ROUND(sale_price * quantity, 2) total_revenue
    FROM
      feb_sales
  )
GROUP BY
  category,
  item_id
```

○

```sql
--OPTION
SELECT
  category,
  ROUND(STD(total_revenue), 3) std_deviation,
  ROUND(AVG(total_revenue),2) avg_revenue
FROM
  (
    SELECT
      *,
      ROUND(sale_price * quantity, 2) total_revenue
    FROM
      feb_sales
  )
GROUP BY
  category, quantity
```

○

```sql
--OPTION

SELECT
  category,
  ROUND(STD(total_revenue), 3) std_deviation,
  ROUND(AVG(total_revenue),2) avg_revenue
FROM
  (
    SELECT
      *,
      ROUND(sale_price * quantity, 2) total_revenue
    FROM
      feb_sales
  )

```

◉

```sql
--OPTION
SELECT
  category,
  ROUND(STD(total_revenue), 3) std_deviation,
  ROUND(AVG(total_revenue),2) avg_revenue
FROM
  (
    SELECT
      *,
      ROUND(sale_price * quantity, 2) total_revenue
    FROM
      feb_sales
  )
GROUP BY
  category
```

Correct

13. 1 / 1 point

Question: Pivot table

Display `categories` as columns. Show `total_rev` in each category.

Given the schema:

column	type
item_id	INT
category	STRING
sale_price	DOUBLE
quantity	INT

Select the query that achieves the following:

- Creates a new column `total_revenue` by multiplying `sale_price` by the `quantity`, rounded to the nearest cent.
- Display the sum of `total_revenue` in each `category`, where each `category` is a column.

○

```sql
--OPTION
SELECT
  *
FROM
  (
    SELECT
      category,
      ROUND(SUM(total_revenue), 2)
    FROM
      feb_sales
  ) PIVOT (
    ROUND(sale_price * quantity, 2) total_revenue FOR category IN ('books', 'magazines', 'movies')
  );
```

○

```sql
--OPTION
SELECT
  *
FROM
  (
    SELECT
      ROUND(sale_price * quantity, 2) total_revenue
    FROM
      feb_sales
  ) PIVOT (
    ROUND(SUM(total_revenue), 2) FOR category IN ('books', 'magazines', 'movies')
  );
```

◉

```sql
--OPTION
SELECT
  *
FROM
  (
    SELECT
      category,
      ROUND(sale_price * quantity, 2) total_revenue
    FROM
      feb_sales
  ) PIVOT (
    ROUND(SUM(total_revenue), 2) FOR category IN ('books', 'magazines', 'movies')
  );
```

○

```sql
--OPTION
SELECT
  *
FROM
  (
    SELECT
      *,
      ROUND(sale_price * quantity, 2) total_revenue
    FROM
      feb_sales
  ) PIVOT (
    ROUND(SUM(total_revenue), 2) FOR category IN ('books', 'magazines', 'movies')
  );
```

Correct

14. 1 / 1 point

Question: Null Values and Aggregates

Compute sums of `sales_rev` grouped by `dept` after dropping null values from `inventory` table.

Given the schema:

column	type
item_id	STRING
dept	STRING
sales_rev	DOUBLE
price	DOUBLE

Select the query that achieves the following:

- Drops any rows that contain null values in either the `item_id` or the `dept` column
- Aggregates sums of the `sales_rev` column grouped by `dept`

○

```
--OPTION
SELECT
  dept,
  SUM(sales_rev)
FROM
  inventory
GROUP BY
  dept
```

◉

```
--OPTION
SELECT
  dept,
  SUM(sales_rev)
FROM
  inventory
WHERE
  (
    item_id IS NOT NULL
    AND dept IS NOT NULL
  )
GROUP BY
  dept
```

○

```
--OPTION
SELECT
  dept,
  SUM(sales_rev)
FROM
  inventory
WHERE
    item_id AND dept IS NOT NULL
```

○

```
--OPTION
SELECT
  dept,
  SUM(sales_rev)
FROM
  inventory
WHERE
    item_id AND dept IS NOT NULL
GROUP BY
  dept
```

Correct

15. 1 / 1 point

Question: Generate subtotals by rollup

Compute averages of `revenue` grouped by `title` and `month` such that the results include averages across all months as well as a subtotal for an individual months from the `book_sales` table.

Given the schema:

column	type
title	STRING
month	INTEGER
revenue	DOUBLE

Select the query that achieves the following:

- Coalesces null values generated by the `ROLLUP` clause in the `month` column
- Creates a new column, `avg_revenue`, rounded to the nearest dollar
- Groups records by `title` and then `month`

○

```
--OPTION
SELECT
  title,
  month
  ROUND(AVG(revenue), 2) as avg_revenue
FROM book_sales
GROUP BY ROLLUP (title, month)
```

◉

```
--OPTION
SELECT
  COALESCE(title, "All items") AS title,
  COALESCE(month, "All months") AS month,
  ROUND(AVG(revenue), 2) as avg_revenue
FROM book_sales
GROUP BY ROLLUP (title, month)
ORDER BY title, month;

```

○

```
--OPTION
SELECT
  COALESCE(title AS title),
  COALESCE(month AS month),
  ROUND(AVG(revenue), 2) as avg_revenue
FROM book_sales
GROUP BY ROLLUP (title, month)
ORDER BY title, month;


```

○

```
--OPTION
SELECT
  title,
  month
  ROUND(AVG(revenue), 2) as avg_revenue
FROM book_sales
GROUP BY ROLLUP (title, month)
ORDER BY title, month;

```

Correct

16. 1 / 1 point

Question: MapTypes

Flatten the `customers` table described below so that there is no `MAP` and the query results include 4 columns: `first_name`, `last_name`, `titles`, and `payment_methods`

Given the schema:

column	type
first_name	`STRING`
last_name	`STRING`
subscriptions	`MAP<STRING, MAP <titles:ARRAY<STRING>, payment_methods:ARRAY<STRING> > >`

Select the SQL query that achieves the following:

- Returns results with the following schema

column	type
first_name	STRING
last_name	STRING
titles	ARRAY
payment_methods	ARRAY

Note: Remember, when you `EXPLODE` a `MapType` column, it breaks the field into `key`, `value` pairs.

○

```
--OPTION
  SELECT
    first_name,
    last_name,
    EXPLODE (subscriptions),
    key.titles,
    key.payment_methods
  FROM
    customers
```

○

```
--OPTION
SELECT
  first_name,
  last_name,
  subscriptions.titles,
  subscriptions.payments
FROM
  customers
```

○

```
--OPTION
WITH explode_subs AS (
  SELECT
    first_name,
    last_name,
    EXPLODE (subscriptions)
  FROM
    customers
)
SELECT
  first_name,
  last_name,
  subscriptions.titles,
  subscriptions.payment_methods
FROM
  explode_subs
```

◉

```
--OPTION
WITH explode_subs AS (
  SELECT
    first_name,
    last name,
```

```
    EXPLODE (subscriptions)
  FROM
    customers
)
SELECT
  first_name,
  last_name,
  value.titles,
  value.payment_methods
FROM
  explode_subs
```

Correct

17. 1 / 1 point

Question: Arrays and Indexes

Access the first element of the `titles` array from the table `customers`.

Given the schema:

column	type
first_name	`STRING`
last_name	`STRING`
subscriptions	`MAP<STRING, MAP <titles:ARRAY<STRING>, payment_methods:ARRAY<STRING> > >`

Select the SQL query that achieves the following:

- Returns results that include only columns: `first_name`, `last_name`, and `title`
- Lists the first element of the `titles` array as the string in the `title` column

○

```
--OPTION
WITH explode_subs AS (
  SELECT
    first_name,
    last_name,
    EXPLODE (subscriptions)
  FROM
    customers
)
SELECT
  first_name,
  last_name,
  subscriptions.titles[0]
FROM
  explode_subs
```

○

```
--OPTION
WITH explode_subs AS (
  SELECT
    first_name,
    last_name,
    EXPLODE (subscriptions)
  FROM
    customers
)
SELECT
  first_name,
  last_name,
  subscriptions.titles,
  subscriptions.payment_methods
FROM
  explode_subs
```

◉

```
--OPTION
SELECT
  first_name,
  last_name,
  value.titles[0] AS title
FROM (
    SELECT
```

```
    first_name,
    last_name,
    EXPLODE (subscriptions)
  FROM
    customers
  )
```

○

```
--OPTION
SELECT
  first_name,
  last_name,
  value.titles[0] AS title
FROM
    SELECT
    first_name,
    last_name,
    EXPLODE (subscriptions)
  FROM
    customers

```

Correct

18. 1 / 1 point

Question: Transform

Transform mixed-case `titles`, in the table `books`, to upper case.

Given the schema:

column	type
titles	ARRAY

Each array contains mixed-case book titles, e.g. `[the red barn, Living, HAPPINESS]`.

Select the query that achieves the following:

- Returns the `titles` column with all values written in uppercase.

○

```
--OPTION
SELECT
  TRANSFORM(t, titles -> UPPER(t)) titles
FROM
  books
```

◉

```
--OPTION
SELECT
  TRANSFORM(titles, t -> UPPER(t)) titles
FROM
  books

```

○

```
--OPTION
SELECT
  TRANSFORM(titles, t -> UPPER(t))
FROM
  books
```

○

```
--OPTION
SELECT
  TRANSFORM(t, titles -> UPPER(t)) titles
```

Correct

19. 1 / 1 point

Question: Flag Records

Create a new column to flag low battery levels with no available backups from the table `devices`

Given the schema:

column	type
battery_levels	ARRAY (INT)
backups	INT
location_id	INT

Select the query that achieves the following:

- Creates a new column named `needs_service` that holds Boolean values
- Uses a higher-order function to flag locations reporting battery levels under 4 **and** no available backups
- Returns columns `location_id` and `needs_service`

```
--OPTION
SELECT
  location_id,
  backups,
  EXISTS(battery_levels, b -> b < 4 AND backups = 0) needs_service
FROM devices
```

```
--OPTION
SELECT
  location_id,
  backups,
  FILTER(battery_levels, b -> b < 4 AND backups = 0)
FROM devices
```

```
--OPTION
SELECT
  location_id,
  FILTER(battery_levels, b -> b < 4 AND backups = 0) needs_service
FROM devices
```

```
--OPTION
SELECT
  location_id,
  EXISTS(battery_levels, b -> b < 4 AND backups = 0) needs_service
FROM devices
```

Correct

20.

1 / 1 point

Question: Reduce

Find the sum of the `battery_levels` in each array in the table `devices`

Given the schema:

column	type
battery_levels	ARRAY(INT)
backups	INT
location_id	INT

Select the query that achieves the following:

- Creates an new column, `total_batt_levels`, that holds the sum of all battery levels at a location
- Returns two columns: `location_id` and `total_batt_levels`

◉

```
--OPTION
SELECT
  location_id,
  REDUCE(battery_levels, 0, (level, acc) -> level + acc) total_batt_levels
FROM
  devices
```

○

```
--OPTION
SELECT
  location_id,
  REDUCE(level, 0, battery_levels -> level + acc) total_batt_levels
FROM
  devices
```

○

```
--OPTION
SELECT
  location_id,
  REDUCE(battery_levels, 0, level, acc -> level + acc) total_batt_levels
FROM
  devices
```

○

```
--OPTION
SELECT
  location_id,
  REDUCE(level, 0, (battery_levels, acc) -> level + acc) total_batt_levels
FROM
  devices
```

✓ Correct