Sample Candidate

Test ID: 391770004966439 | 0123456789 | sample@email.com

Test Date: June 9, 2022

Automata - SQL

86 /100

Automata - SQL

86 / 100

SELECT	UPDATE	ALTER
80 / 100	**100** / 100	**100** / 100

1 | Introduction

About the Report

This report provides a detailed analysis of the candidate's performance on different assessments. The tests for this job role were decided based on job analysis, O*Net taxonomy mapping and/or criterion validity studies. The candidate’s responses to these tests help construct a profile that reflects her/his likely performance level and achievement potential in the job role

This report has the following sections:

The **Summary** section provides an overall snapshot of the candidate’s performance. It includes a graphical representation of the test scores and the subsection scores.

The **Response** section captures the response provided by the candidate. This section includes only those tests that require a subjective input from the candidate and are scored based on artificial intelligence and machine learning.

The **Proctoring** section captures the output of the different proctoring features used during the test.

Score Interpretation

All the test scores are on a scale of 0-100. All the tests except personality and behavioural evaluation provide absolute scores. The personality and behavioural tests provide a norm-referenced score and hence, are percentile scores. Throughout the report, the colour codes used are as follows:

- Scores between 67 and 100
- Scores between 33 and 67
- Scores between 0 and 33

2 | Response

Automata - SQL

86 / 100

Problem Statement 1

Given a database with table members(**id**, name). Find the name of all members whose name start with ***'h'*** *or* ***'H'***. The final answer must have only column (name).

Response

```
Select name from members where name like '%h' or name like '%H';
Select name from members where name like 'h%' or name like 'H%';
```

Problem Statement 2

A company's database maintains table salesperson(**id**, name) and s_order(order_date, cust_id , sale_id) to keep track of marketing . Find the names of salesperson that have more than 1 order assuming each salesperson has unique *id*. Your final table should contain the number of orders and name of salesperson as columns.

Response

```
select salesperson.name, count(salesperson.id) from salesperson join s_order on salesperson.id = s_order.cust_id;
select salesperson.name, count(salesperson.id) from salesperson join s_order on salesperson.id = s_order.cust_id
group by salesperson.id;
select salesperson.name from salesperson join s_order on salesperson.id = s_order.cust_id;
select * from salesperson
select * from s_order
select salesperson.name from salesperson join s_order on salesperson.id = s_order.sale_id;
select salesperson.name, count(s_order.sale_id) from salesperson join s_order on salesperson.id = s_order.sale_id
group by s_order.sale_id having count(s_order.sale_id) > 1;
select * from s_order
select count(s_order.sale_id) as count, salesperson.name from salesperson join s_order on salesperson.id =
s_order.sale_id group by s_order.sale_id having count(s_order.sale_id) > 1;
```

Problem Statement 3

A college database maintains a table students(name, dept, score). Find the names of all the students who have scores greater than those of at least one student in ECE department.

Response

```
select name from( select min(score) as min_score from students where dept = 'ECE') as E where score > min_score
select name from students where score > (select min(score) as min_score from students where dept = 'ECE')
```

Problem Statement 4

A college database maintains a table students(name, dept, score). Find the department with highest average score. Display it's average score as well.

Response

select dept from (select dept, avg(score) as average from students group by dept) where average = max(average) ;
select dept from (select dept, avg(score) as average from students group by dept) as a where average = max(average) ;
with cte as (Select dept, avg(score) as answer group by dept order by avg(score) desc) Select dept , answer from cte limit 1 ;
with cte as (Select dept, avg(score) as answer group by dept order by avg(score) desc limit 1) Select dept , answer from cte ;
with cte as (Select dept, avg(score) as answer group by dept order by avg(score) desc limit 1); Select dept , answer from cte ;
Select dept, avg(score) as answer group by dept order by avg(score) desc limit 1
Select dept, avg(score) as answer group by dept
with cte as (Select dept, avg(score) as answer from students group by dept order by avg(score) desc limit 1); Select dept , answer from cte ;
with cte as (Select dept, avg(score) as answer from students group by dept order by avg(score) desc limit 1) Select dept , answer from cte ;
Select dept, avg(score) as answer from students group by dept order by avg(score) desc limit 1
Select dept, avg(score) as answer from students group by dept order by avg(score) desc limit 1

Problem Statement 5

The owner of a music store maintains a database of albums with table album (album, band, cost). Seeing the appreciation for *pink floyd* and *led zeppelin* from his customers, he decided to increase the album cost by 50 percent for all the albums of both the bands. Write a query to update the costs.

Response

Update table album Set cost = 1.5*cost where album in ('pink floyd', 'led zeppelin')
Update album Set cost = 1.5*cost where album in ('pink floyd', 'led zeppelin')
select * from album
Update album Set cost = 1.5*cost where band in ('pink floyd', 'led zeppelin')

Problem Statement 6

An online gaming website required registration of participants. A database was designed to maintain records in table participant(**id** , name, age). Initially *id* was of integer type. Later it was decided to make *id* of type text for easy usability. Modify the table schema to do the needful.

Response

ALTER TABLE particpant modify COLUMN id varchar(255);
ALTER TABLE participant modify COLUMN id varchar(255);
ALTER TABLE participant modify COLUMN id text;

Problem Statement 7

A government agency maintains a database that stores the addresses of all the people living in the country. The data is stored in the table – **tbl_address_data**(**sNo**, personId, Address).
The address contains a zip code – a six-digit numeric code unique to every location in the country. In the address, the zip code can be obtained by searching for the keyword – 'zip' (case insensitive) followed by space(s), dash (-) or both and then the numeric code. Write a query to parse the given addresses and extract the list of **personId's** whose address does not have any zip code.

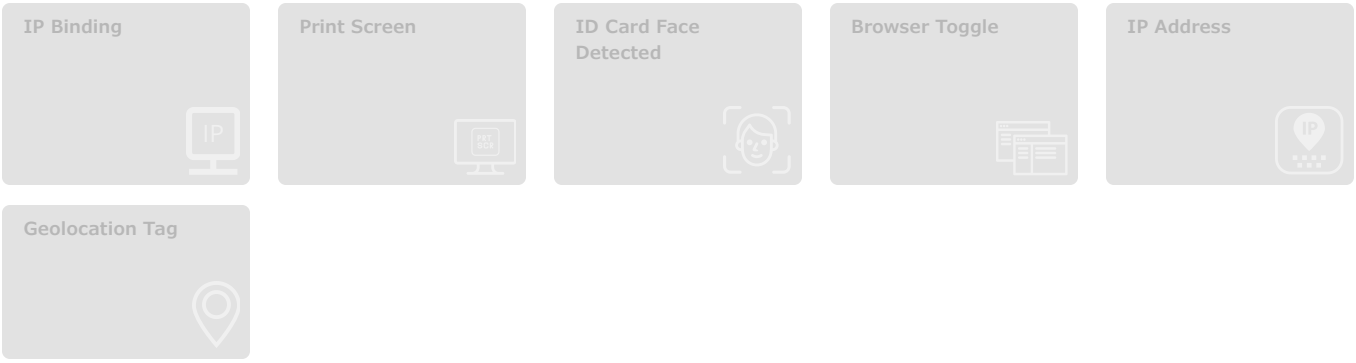
Response

The candidate did not attempt this question.

3 | Proctoring

Proctoring Index | Low

The Proctoring Index is a measure of the likelihood a participant was engaging in potentially suspicious behavior. The Index score constitutes a set of pre-determined parameters that are outlined below as a breakdown of the measure. Please hover over the tiles for a brief overview of the respective parameter.



AI Proctoring Information

Print Screen: The number of times the candidate attempted to take a screenshot of the assessment screen using the “print screen” function on their device. Note: This impacts proctoring index.

ID Card Face Detected: Looks at the candidate images captured during the assessment and flags anywhere different people appear to be present. Snapshots are included in the report.

Browser Toggle: Either the proportion of time the candidate spent focused on a tab/window other than that of assessment screen (%), or the number of times the candidate toggled to another tab/window (count). Note: This impacts proctoring index.

IP Address: Confirms that the candidate took the assessment from the specified IP address(s).

Geolocation Tag: Detects whether the candidate attempted the assessment from a location beyond the distance set by the administrator.