Alexandros Veremis

Assignment for the Junior Software Engineer position

Answers

Task 1

1) {'workabledemo.com/api/accounts/3': 4378, 'www.workabledemo.com/user_password_resets': 1, 'www.workabledemo.com/petitions': 1, 'www.workabledemo.com/accounts': 1, 'sampleco.workabledemo.com/backend/subscription/update_billing': 1, 'www.workabledemo.com/uas/request-password-reset?trk=uas-resetpass': 1}

2) Average response time is: 323.5726403641879 ms

3) [('delayed_jobs', 16742), ('accounts', 5417), ('jobs', 1426), ('members', 1216), ('collaborations', 918), ('candidates', 851), ('postings', 607), ('job_boards', 391), ('stages', 380), ('users', 301), ('subscriptions', 238), ('job_stats', 108), ('plans', 91), ('keywords', 89), ('slot_plans', 75), ('activities', 74), ('slots', 72), ('experiences', 63), ('jobs_keywords', 54), ('auth_identities', 45), ('educations', 37), ('questions', 37), ('tag_lists', 37), ('answers', 36), ('events', 36), ('keyword_matches', 35), ('offices', 29), ('followings', 27), ('skills', 26), ('candidates_skills', 26), ('applicants', 23), ('boards_integrations', 19), ('administrators', 17), ('choices', 12), ('answers_choices', 9), ('attachments', 9), ('invitations', 9), ('oauth_access_tokens', 9), ('oauth_applications', 8), ('attendances', 7), ('comments', 7), ('petitions', 5), ('message_templates', 4), ('oauth_access_grants', 4), ('messages', 3), ('eeoc_profiles', 1), ('subscription_requests', 1)]

4) heroku/router is a router that forwards the calls accordingly to the specific links of the web application. Other than that, redirect_uri is also found at heroku/router

“443624133910355993 2014-09-02T18:36:12 2014-09-03T00:50:58Z 4147981 demo-workablehr 107.22.139.100 Local3 Info heroku/router at=info method=GET path="/oauth/authorize?client_id=3f044f0fec65fz30e347df1d8126c6aa7edfbeba8c8ecd4c4b63654f52c9a955&redirect_uri=http%3A%2F%2Fwww.workabledemo.com%2Faardvark%2Foauth2workable%2Fcode&response_type=code&scope=r_jobs%20r_candidates%20w_candidates" host=www.workabledemo.com request_id=9dc8c30c-ac97-4edf-9eb0-bd6d9dfa2904 fwd="71.193.227.127" dyno=web.2 connect=1ms service=125ms status=302 bytes=1865”

I found it by searching “redirect” with Search Function of Notepad++ .

5) Server error means “status=5xx”. Found 5 lines that include such an error and therefore they occur. Possible causes include:

Application bugs, Database issues, Resource exhaustion, Network problems, Configuration errors, Dependency failures, Security issues, Software updates or changes.
In our case, we got a Timeout Error and therefore one of the Network problems, Configuration errors or Dependency failures seem as the most probable causes.

I wrote the python script “task1/log_findings.py” in order to get the above results.
I ran it with this command: “python task1/log_findings.py --run all”

Task 2

1) Answer:

last_name	first_name	customer_id	rentals
Seal	Karl	526	45

SQL QUERY : “SELECT last_name, first_name, c.customer_id, COUNT(c.customer_id) AS rentals
FROM public.rental r JOIN public.customer c ON r.customer_id = c.customer_id
WHERE c.store_id = 2 GROUP BY c.customer_id ORDER BY rentals DESC LIMIT 1;”

2) Answer: The customer would be able to rent it if he had tried because there was one copy available, since ‘29/07/2005 3pm’ is only between rental and return dates of one of the 2 available copies.

SQL QUERIES:

“SELECT inventory_id FROM public.inventory
WHERE film_id =
(SELECT film_id FROM public.film WHERE film.title = 'Image Princess')
AND store_id = 2;” which gives us 2092 and 2093

“SELECT COUNT(inventory_id) FROM rental
WHERE '2005-07-29 15:00:00' BETWEEN rental_date AND return_date
AND (inventory_id = 2092 or inventory_id =2093);”

The second SQL QUERY returns 1 and therefore we understand that one copy was available and one rented.

3) Answer:

month_year	active_customers
May-2005	520
Jun-2005	590
Jul-2005	599
Aug-2005	599
Feb-2006	158

SQL QUERY: "SELECT TO_CHAR(rental.rental_date, 'Mon-YYYY') AS month_year,
COUNT(DISTINCT customer_id) AS active_customers
FROM rental
GROUP BY TO_CHAR(rental.rental_date, 'Mon-YYYY')
ORDER BY MIN(rental.rental_date) ASC;"

4) Answer: average_movie_length = 115.272

The whole list can be found at "task2/list_movie_length_more_than_avg.csv"

SQL QUERIES:

" SELECT AVG(length) AS average_movie_length
FROM public.film;"
"SELECT title FROM public.film WHERE length > 115.272;"

5) Answer: The whole list can be found at
"task2/list_customer_payments_more_than_avg.csv"

SQL QUERY: " SELECT c.last_name, c.first_name, c.customer_id, COUNT(p.customer_id) AS payments FROM public.payment p JOIN
public.customer c ON p.customer_id = c.customer_id
GROUP BY c.customer_id
HAVING COUNT(p.customer_id) >
(SELECT AVG(payments) AS avg_rn
FROM(SELECT c.customer_id, COUNT(p.customer_id) AS payments
FROM public.payment p JOIN public.customer c ON p.customer_id = c.customer_id
GROUP BY c.customer_id) AS payment_numbers) ORDER BY payments DESC;"

6) Top 100 customers*: "SELECT SUM(amount), customer_id FROM payment GROUP BY customer_id ORDER BY SUM(amount) DESC LIMIT 100;"*
in order to offer a reward to the well-paying customers.

Bottom 100 customers: *"SELECT SUM(amount), customer_id FROM payment GROUP BY customer_id ORDER BY SUM(amount) ASC LIMIT 100;"*
in order to send them a message and try to draw their attention again.

Top Categories: What the owner has in his store the most vs what people want. In order to realize which category of films he should bring more and which to lessen.

“SELECT c.category_id ,COUNT(fc.film_id) AS film_count
FROM category c
JOIN film_category fc ON c.category_id = fc.category_id
GROUP BY c.category_id
ORDER BY film_count DESC;”

“SELECT c.category_id AS cat_id, COUNT(rental.rental_id) AS rental_count
FROM category c
JOIN film_category fc ON c.category_id = fc.category_id
JOIN film f ON fc.film_id = f.film_id
JOIN inventory i ON f.film_id = i.film_id
JOIN rental ON i.inventory_id = rental.inventory_id
GROUP BY cat_id
ORDER BY rental_count DESC;”

Top 100 rented movies, in order to be stricter on the allowed rental days:

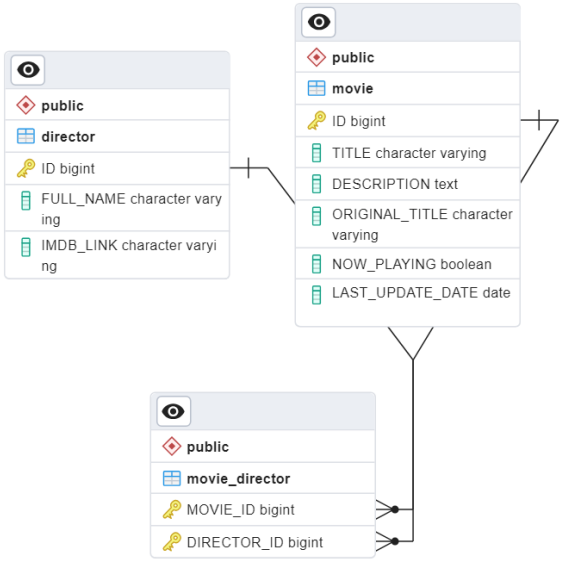
“SELECT f.title, COUNT(rental.rental_id) AS rental_count
FROM film f
JOIN inventory i ON f.film_id = i.film_id
JOIN rental ON i.inventory_id = rental.inventory_id
GROUP BY f.film_id
ORDER BY rental_count DESC limit 100;”

Bottom 100 rented movies, in order to pack some of them together with a top-rented movie as a sales promotion:

“SELECT f.title, COUNT(rental.rental_id) AS rental_count
FROM film f
JOIN inventory i ON f.film_id = i.film_id
JOIN rental ON i.inventory_id = rental.inventory_id
GROUP BY f.film_id
ORDER BY rental_count ASC limit 100;”

Task 3

1) The source code can be found at directory “task3” and more specifically at “movie_data.py”
2) DDL for Database Schema: can be found at “task3/ddl.sql” file.
Database name is movie_db and the Database Schema is:



3) Instructions on how to build and run the application can be found at “task3/Readme.txt”