Tester: Mardianto Soebagio Hadiputro [+886-928-942-094]

1) SQL
We have two tables **content_metadata** and **flight,** described as follows:

```
TABLE content_metadata
(
  content_id text NOT NULL,
  publisher_id text NOT NULL,
  wf_process_instance_id text,
  flight_id text,
  CONSTRAINT content_metadata_pk PRIMARY KEY (content_id),
  CONSTRAINT flight_id_fk FOREIGN KEY (flight_id)
      REFERENCES flight (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
```

And

```
TABLE flight
(
  id text NOT NULL,
  name text NOT NULL,
  CONSTRAINT flight_pk PRIMARY KEY (id),
  CONSTRAINT unique_flight_name UNIQUE (name)
)
```

Please select random 10 entries from content_metadata table with name of corresponding flight started with "TEST_FLIGHT".

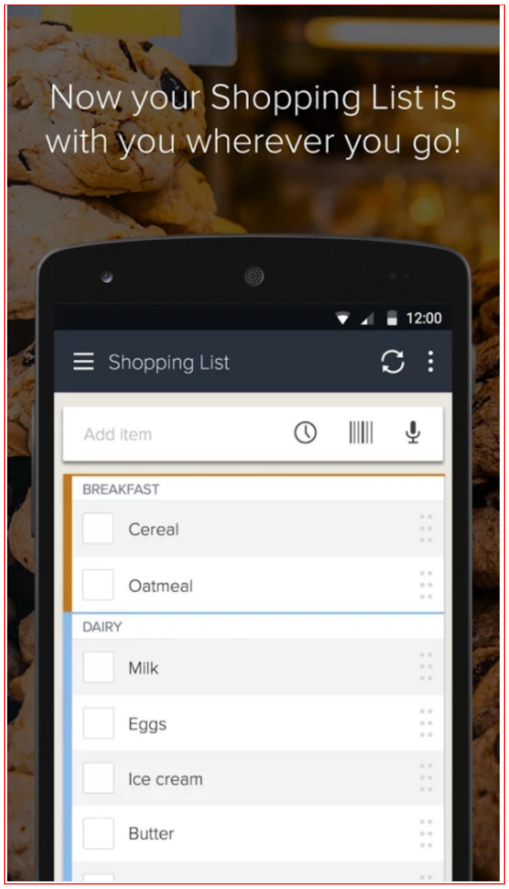
Answer:

```
SELECT * FROM content_metadata
WHERE flight_id IN (SELECT id FROM flight WHERE name LIKE 'TEST_FLIGHT%')
LIMIT 10;
```

References:

- SELECT … FROM: https://www.w3schools.com/sql/sql_select.asp
- WHERE … IN: https://www.w3schools.com/sql/sql_in.asp
- LIKE: https://www.w3schools.com/sql/sql_like.asp
- LIMIT: https://www.w3schools.com/sql/sql_top.asp

2) Test Planning
Download and install OutOfMilk application. Play a bit to understand whole idea of this shopping list application



2.1 How you will approach to test this application? Provide general ideas / check list
2.2 Take a deeper look into the adding new item in the list functionality (voice, search out of focus). Plan test activities for test this functionality.
2.3 Put test cases into priorities

Answer:
2.1 From testing scope approach:

- Basic Functional Test
  - Make sure all basic functions works normally, such as: installation/uninstallation, registration/login, create/delete/modify shopping list/pantry list/to-do list, etc. The expected result of each test item should refer to the Functional Specification defined by app product manager.
  - Make sure there is no abnormal UI, such as text truncation, object out of place, etc.
  - This test is performed for every release build
  - Each single test can be executed on different devices in order to make sure that all supported devices are tested.
- Advanced Functional Test
  - Error handling test: make sure that app handles error gracefully (i.e., app doesn't crash) for error condition, such as no Internet connection / slow Internet connection, user force to close the app, etc.
  - Multi-language test: make sure that UI shows normally in all supported languages (i.e., no truncated text, etc.)
  - Multi device test: make sure that all the test cases are run against all supported devices and OS versions:
    - iOS -> combination test of iPhone, iPad, iPod touch with different iOS 8, 9, 10 versions (e.g., iPhone with iOS 10, iPad with iOS 9, iPod touch with iOS 8, …)

        - Android -> combination with various supported Android version and screen resolution (suggest to choose the popular Android phones on the market)
    - Cross device test: make sure that when user migrates between iOS and Android devices, user can login successfully and user data can be synced successfully, etc.
    - This test is performed only for major milestone build release or if there's a request from the team (this test is not executed for every build release)
- Bug Verification Test
    - Verify that issue is already resolved. The expected result should be defined clearly.
    - Verify that there is no side-effect from the bug fix
- [Optional] Usability Test
    - Verify the usability of each function

From testing execution method approach:

- Manual test
    - This test is performed by QA manually interacting with the app as a normal user would interact with the app. QA should function as a very advanced user of the app, by executing all the test cases defined in the test plans.
- Automation test
    - This test is performed via automation tools, such as Appium for mobile device automation, or Selenium for web-based applications.
    - Some or most of the basic functional tests can be executed by way of automation, such as signing in, creating/adding/deleting list, etc.
    - Automation test script job can be integrated into the build server so that automation test would be executed automatically once the build is released.
    - If a bug/issue is found via automation test, QA still need to manually reproduce the bug/issue in order to make sure that the bug is not from the automation test error.
- Unit test (performed by RD)
    - This test is modular test performed by RD to make sure that the output of function within the app is as expected

2.2 Add item functionality test cases example. Below are two samples of test cases of Add item functionality. Many more samples can be added to fully coverage the test plan.

Test Item		Test Steps	Expected Result*
Voice search	With internet connection	1. Make sure internet is connected	"Sugar" item is created successfully
		2. Click [Mic] button	
		3. Say "sugar"	
		4. Choose "sugar" on the suggestion dialog	
		5. Click [OK] checkmark button	
	Without internet connection	1. Use Airplane mode in the device	[Mic] button should be disabled
		2. Click [Mic] button	
		3. Say "sugar"	
Text search	Special characters	1. Tap into [Add item] text box	Dialog should show up that item is not recognizable
		2. Type ";*&^"	
		3. Click [Done] on the onscreen keyboard	

*Expected result should be based on the Functional Specification defined by Product Manager

2.3 Take above test cases and add priorities to them

Test Item		Test Steps	Expected Result	Priority*
Voice search	With internet connection	1. Make sure internet is connected	"Sugar" item is created successfully	1
		2. Click [Mic] button		
		3. Say "sugar"		
		4. Choose "sugar" on the suggestion dialog		
		5. Click [OK] checkmark button		
	Without internet connection	1. Use Airplane mode in the device	[Mic] button should be disabled	1
		2. Click [Mic] button		
		3. Say "sugar"		
Text search	Special characters	1. Tap into [Add item] text box	Dialog should show up that item is not recognizable	2
		2. Type ";*&^"		
		3. Click [Done] on the onscreen keyboard		

*Priority:
1 – High; 2 – Normal; 3 - Low

3) Test Automation (optional)

This task can be hard, especially if you don't have previous experience with mobile testing. So please free to stop at any step we are not expecting to see 100% working setup! Our interest here mainly to see your problem solving skills in action.

Review several tutorials about Appium. Your task is move as far as possible along the way to setup working project with Appium. In case of unbearable problem arise, please describe this problem and solutions which you have tried. Also please free to return back with any questions and clarifications.

Prepare and setup environment for automated testing android application with Appium.

You can choose:
- platform you will focus on (Android or iOS)
- device / simulator / cloud based simulator
- Implement simple test case (from Task2) in Java/Groovy.

Please send your solution as a github link for a code and some document for project setup.

Answer:

In this Appium automation test sample project, I will use real Android device to execute the test script. The test script is written in Python language.

Below is the testing environment setting that I used:
Host OS: macOS Sierra 10.12.5
Device: Galaxy Note 4 (Android 5.0.1)

Test scenario:
Add item
Steps:
1. Launch OutOfMilk app
2. Click [Skip] at login page
3. [Optional] Remove the "What's New in v5.4.4…" dialog if it shows up
4. Type "sugar" in [Add item] box
5. Show onscreen keyboard
6. Press [Done] in onscreen keyboard

Expected result:
"Sugar" item is created successfully

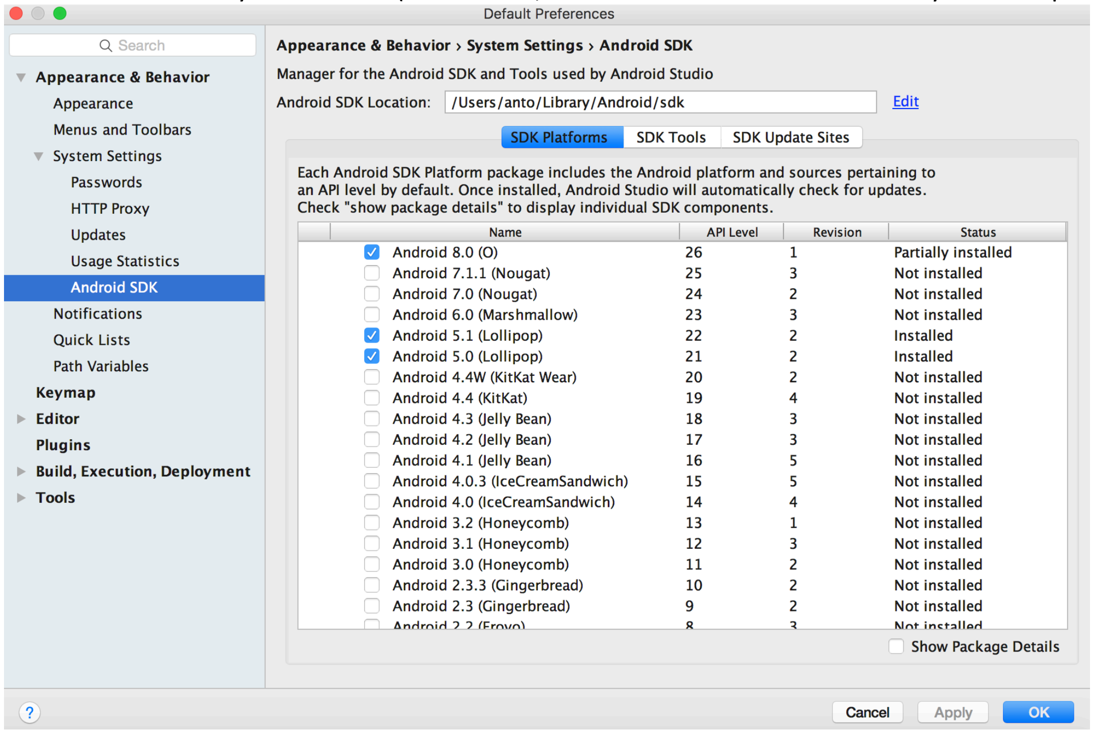
Delete item
Steps:
1. Perform "Add item" scenario above
2. Check the "sugar" item checkbox
3. Click [DELETE ALL] button
4. Click [DELETE ALL] button in Confirm page

Expected result:
"Sugar" item is deleted successfully

Pre-condition installation:

1) Install Android SDK
   https://developer.android.com/studio/index.html

   1.1. Download necessary Android kit (in this case, I downloaded Android 5.0 for my test sample project)



Appearance & Behavior › System Settings › Android SDK

Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: /Users/anto/Library/Android/sdk

Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, Android Studio will automatically check for updates. Check "show package details" to display individual SDK components.

	Name	API Level	Revision	Status
✓	Android 8.0 (O)	26	1	Partially installed
	Android 7.1.1 (Nougat)	25	3	Not installed
	Android 7.0 (Nougat)	24	2	Not installed
	Android 6.0 (Marshmallow)	23	3	Not installed
✓	Android 5.1 (Lollipop)	22	2	Installed
✓	Android 5.0 (Lollipop)	21	2	Installed
	Android 4.4W (KitKat Wear)	20	2	Not installed
	Android 4.4 (KitKat)	19	4	Not installed
	Android 4.3 (Jelly Bean)	18	3	Not installed
	Android 4.2 (Jelly Bean)	17	3	Not installed
	Android 4.1 (Jelly Bean)	16	5	Not installed
	Android 4.0.3 (IceCreamSandwich)	15	5	Not installed
	Android 4.0 (IceCreamSandwich)	14	4	Not installed
	Android 3.2 (Honeycomb)	13	1	Not installed
	Android 3.1 (Honeycomb)	12	3	Not installed
	Android 3.0 (Honeycomb)	11	2	Not installed
	Android 2.3.3 (Gingerbread)	10	2	Not installed
	Android 2.3 (Gingerbread)	9	2	Not installed
	Android 2.2 (Froyo)	8	3	Not installed

   1.2. Make sure ANDROID_HOME path is the path of your Android SDK. In Terminal, type

```
nano ~/.bash_profile
```

   add lines:

```
export ANDROID_HOME=/YOUR_PATH_TO/android-sdk
export PATH=$ANDROID_HOME/platform-tools:$PATH
export PATH=$ANDROID_HOME/tools:$PATH
```

   re-open terminal and check if it works:

```
echo $ANDROID_HOME
```

2) Install Java 8 JDK
   http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

3) Install Homebrew from https://brew.sh/

4) Get Node.js. In Terminal, type

```
brew install node
brew update
```

5) Get Appium. In Terminal, type `npm install -g appium`

6) Get Appium Client. In Terminal, type `npm install wd`

7) Install Python 2.7

7.1 My test script is written in Python, so you will need to install python:
https://www.python.org/downloads/mac-osx/

7.2 Install pip (pip is package installer for python)
in Terminal, type `sudo easy_install pip`

7.3 Install package dependency for this project. In Terminal, type

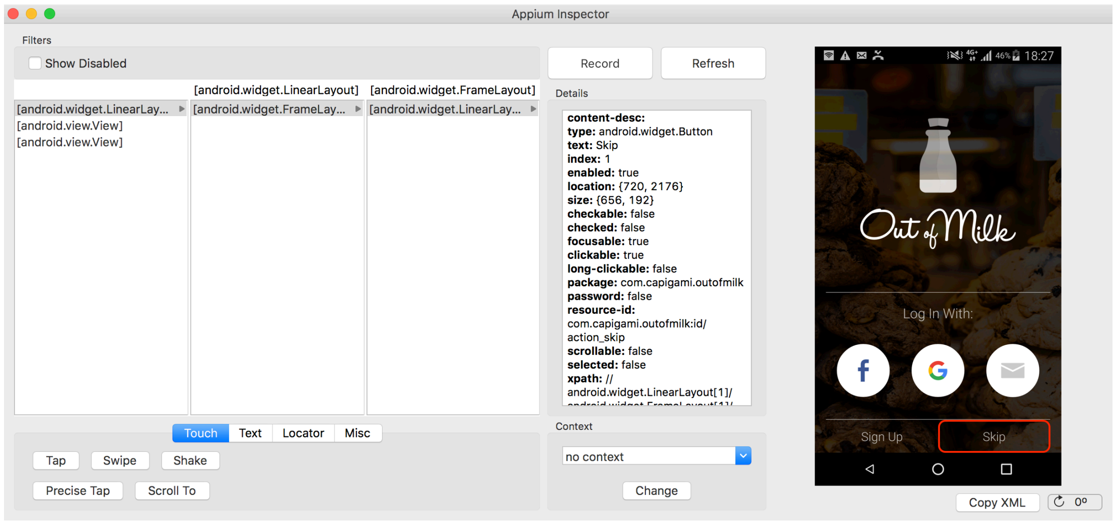
```
sudo pip install Appium-Python-Client
sudo pip install -U selenium
```

7.4 Download Selenium Java Server from:
http://selenium.googlecode.com/files/selenium-server-standalone-2.7.0.jar

and type (in Terminal) in the download path

```
java -jar selenium-server-standalone-2.7.0.jar
```

2 [OPTIONAL] Get Appium Desktop (this is for Appium Inspector to inspect App object XML property)
https://bitbucket.org/appium/appium.app/downloads/appium-1.4.0.dmg

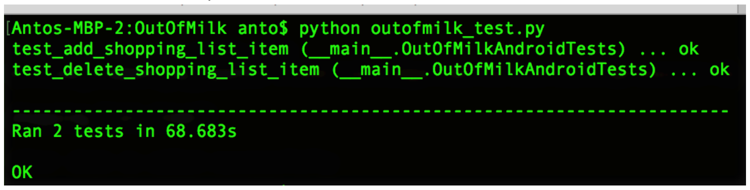


content-desc:
type: android.widget.Button
text: Skip
index: 1
enabled: true
location: {720, 2176}
size: {656, 192}
checkable: false
checked: false
focusable: true
clickable: true
long-clickable: false
package: com.capigami.outofmilk
password: false
resource-id: com.capigami.outofmilk:id/action_skip
scrollable: false
selected: false
xpath: //android.widget.LinearLayout[1]/

Testing steps:

1) Get the project files from GitHub (download the "apps" folder and the "outofmilk_test.py")
   https://github.com/mantoshka/OutOfMilk

2) Connect Android device (with Android 5.0.1) and make sure "USB debugging" mode is enabled on the device
3) Start Appium: in Terminal, type `appium &`
4) Run the test script: open new Terminal tab, navigate to the project files, and type `python outofmilk_test.py`
5) The script should automatically install the App if it's not installed yet, then proceed with the test scenario
6) If the result is OK, the Terminal should show



```
[Antos-MBP-2:OutOfMilk anto$ python outofmilk_test.py
test_add_shopping_list_item (__main__.OutOfMilkAndroidTests) ... ok
test_delete_shopping_list_item (__main__.OutOfMilkAndroidTests) ... ok

----------------------------------------------------------------------
Ran 2 tests in 68.683s

OK
```