ACTIVITY **3.1.6**

Public vs. Private Trips

INTRODUCTION

In the previous activity, you finished the back-end functionality so users can successfully create, update, and delete trips. In this activity, you will modify the trip screens to accommodate the viewing of publicly shared trips. In either public or private view, users should be able to update and delete only the trips they own.

Materials

- Computer with Android™ Studio
- Android™ tablet and USB cable, or a device emulator
- Free Backendless account per student

RESOURCES

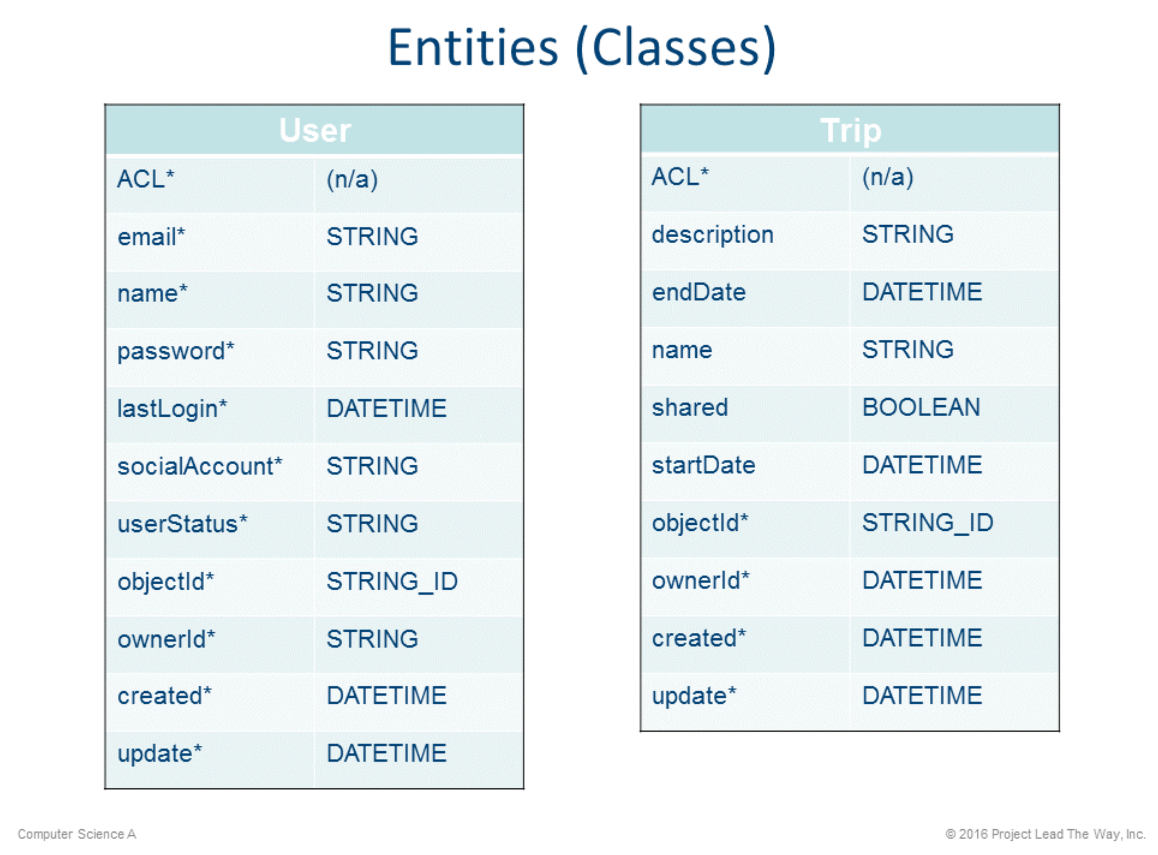
Lesson 3.1 Reference Card for Backendless
Resources available online

Procedure

Part I: Publicly Shared Trips

Your TripTracker app can keep track of trips and trip ownership using the Backendless database.

1. To understand how Backendless manages entity ownership, review the slideshow. Summarize it here and describe the entity relationship.



Entities (Classes)

User	
ACL*	(n/a)
email*	STRING
name*	STRING
password*	STRING
lastLogin*	DATETIME
socialAccount*	STRING
userStatus*	STRING
objectId*	STRING_ID
ownerId*	STRING
created*	DATETIME
update*	DATETIME

Trip	
ACL*	(n/a)
description	STRING
endDate	DATETIME
name	STRING
shared	BOOLEAN
startDate	DATETIME
objectId*	STRING_ID
ownerId*	DATETIME
created*	DATETIME
update*	DATETIME

Computer Science A

As it stands, the User and Trip entities look like this. The entities are listed in the order that you see them in the DATA BROWSER of each table.

All columns with an asterisk (*) are maintained by Backendless.

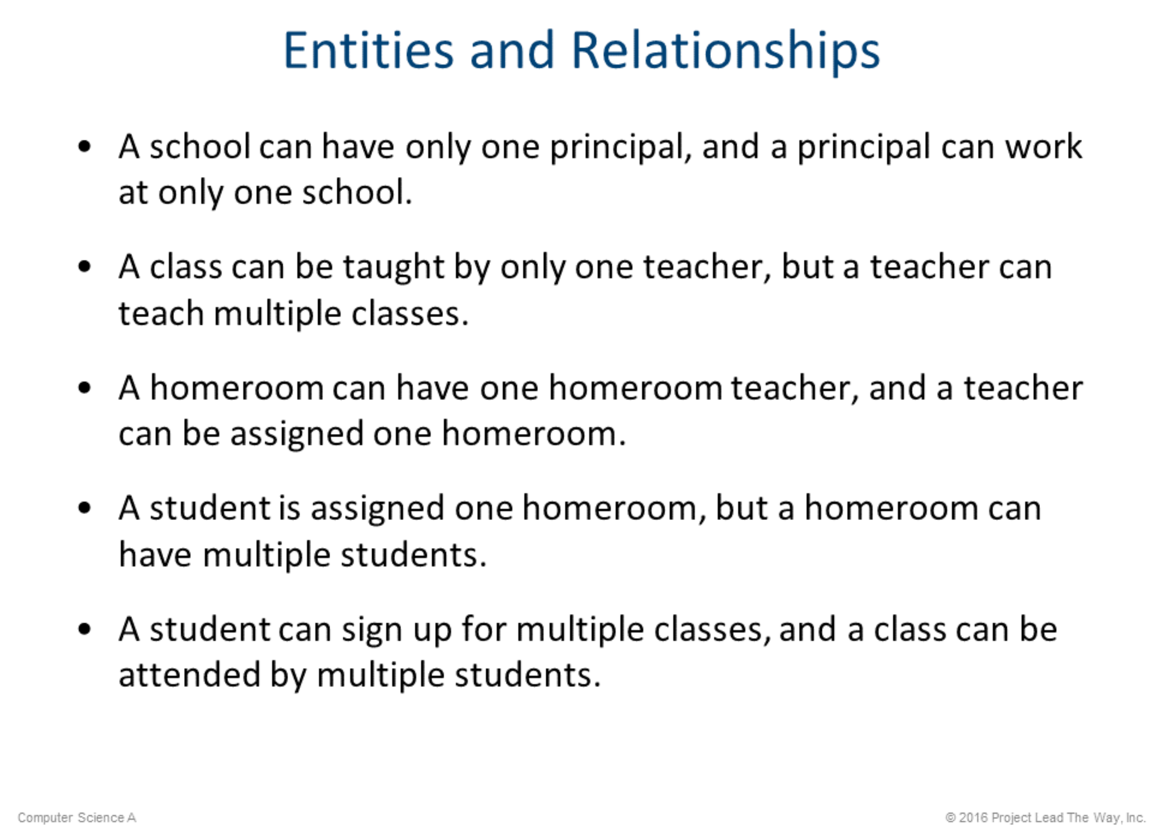
Most columns are self-explanatory. The ACL (Access Control List) defines user permissions including remove, find, update. You will not be modifying ACLs.

Columns in the User table used by TripTracker:

- email, name, and password

Columns used and created in the Trip class are:

- Trip: description, endDate, name, shared, startDate, and ownerId



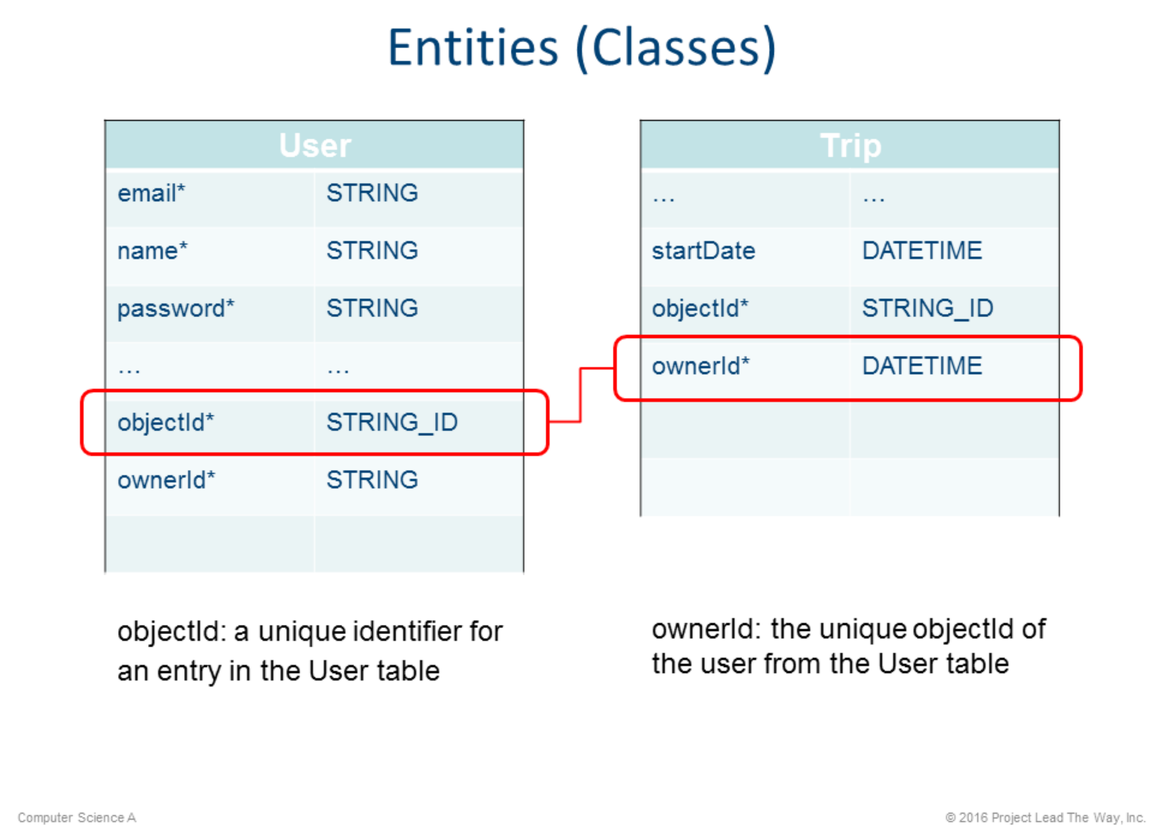
Entities and Relationships

- A school can have only one principal, and a principal can work at only one school.
- A class can be taught by only one teacher, but a teacher can teach multiple classes.
- A homeroom can have one homeroom teacher, and a teacher can be assigned one homeroom.
- A student is assigned one homeroom, but a homeroom can have multiple students.
- A student can sign up for multiple classes, and a class can be attended by multiple students.

Computer Science A

Entities (and even objects within the same entity) can be connected using database relationships. There can be three kinds of relationships between two objects:

- One-to-one relationships (1-1) associate one object with another object.
- One-to-many relationships (1-N) associate one object with many other objects.
- Many-to-many relationships (N-N) can associate multiple objects with multiple other objects, hence enabling complex relationships among many objects.
- The list on the slide provides some examples of relationships. Identify which relationships are 1-1, 1-N, and N-N.



Entities (Classes)

User	
email*	STRING
name*	STRING
password*	STRING
…	…
objectId*	STRING_ID
ownerId*	STRING

Trip	
…	…
startDate	DATETIME
objectId*	STRING_ID
ownerId*	DATETIME

objectId: a unique identifier for an entry in the User table

ownerId: the unique objectId of the user from the User table

Computer Science A

© 2016 Project Lead The Way, Inc.

From the various entity relationships (1-1, 1-N, and N-N) , you will implement a one-to-many or 1-N for the User and Trip tables.

A 1-N relationship means that one user can own many trips.

To establish a 1-N relationship:

- You need a unique identifier for the one entity. In this case, it is User: objectId (uniquely identifies a user).
- Refer to the unique identifier in the many entity of another table. Here it is Trip: ownerId (many trips can have the same ownerId,;it is not unique).

This creates a connection between the two entities, called a join.

2 Open your TripTracker app in Android Studio.

NOTE

If you were unable to complete *Activity 3.1.5 Updates and Deletes*, import `3.1.5TripTracker_Solution` as directed by your teacher. Recall that if you import the solution, you must update keys values in `strings.xml` Specifically,

- Change `be_app_id` your Backendless App ID key value
- Change `be_android_api_key` to your Android API key value

You can retrieve these from your **Backendless Console** (**Manage** icon).

3 Launch your app. Confirm that you have at least two users and a few trips created by each user.

Currently, you are viewing only the trips created by the user who is logged in to the app (the "current user" as Backendless calls this). The user refers to this view of trips as "My Trips".

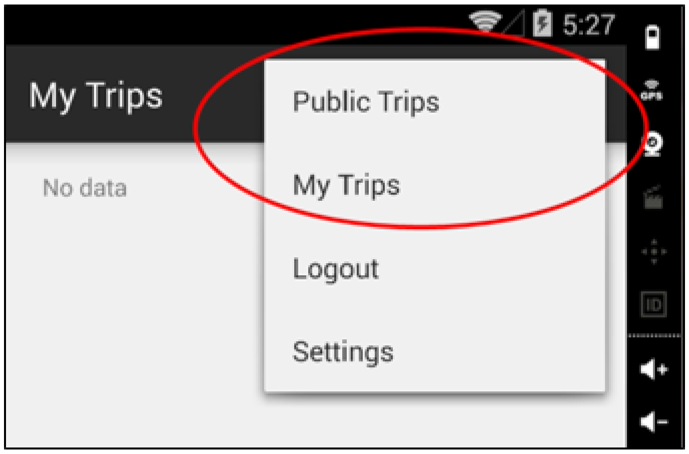
Suppose a user wants to make their trips publicly available, so anyone with this app can see them. The user would have ownership for editing and deleting, but other people should be able to view publicly shared trips.

4 Edit a few of your trips and select the **Shared/Public?** check box.

5 Save your trip, and in the Backendless Console, confirm that the trip was saved with Shared set to True. Also confirm that your app shows the trip as Shared/Public on the Trip Details screen.

Currently your app runs in My Trips mode, the mode that shows just the trips owned by you, the current user. You will be adding functionality to run the app in Public Trips mode, to see all trips that *any* user has shared publicly.

Note the menu options on the My Trips screen:



The menu shows both the Public Trips and My Trips options. You will change these menu options depending on whether the app is in public mode or private mode. If the app is in public mode, you will show the option to change to private; if the app is in private mode, you will show the option to change to public. This **toggle** behavior, switching between two modes or states, is a common computing feature. Specifically, you will make changes in your code so that

- In My Trips mode, the menu options should show:
  - Public Trips
  - Logout
  - Settings

- In Public Trips mode, the menu options should show:
  - My Trips
  - Logout
  - Settings

To control what is displayed in the menu, you can override the `onPrepareOptionsMenu` method in `TripListFragment`, which runs just before the `onCreateOptionsMenu` runs.

Open the class and add the following method *after* the `onCreateOptionsMenu` method in your code.

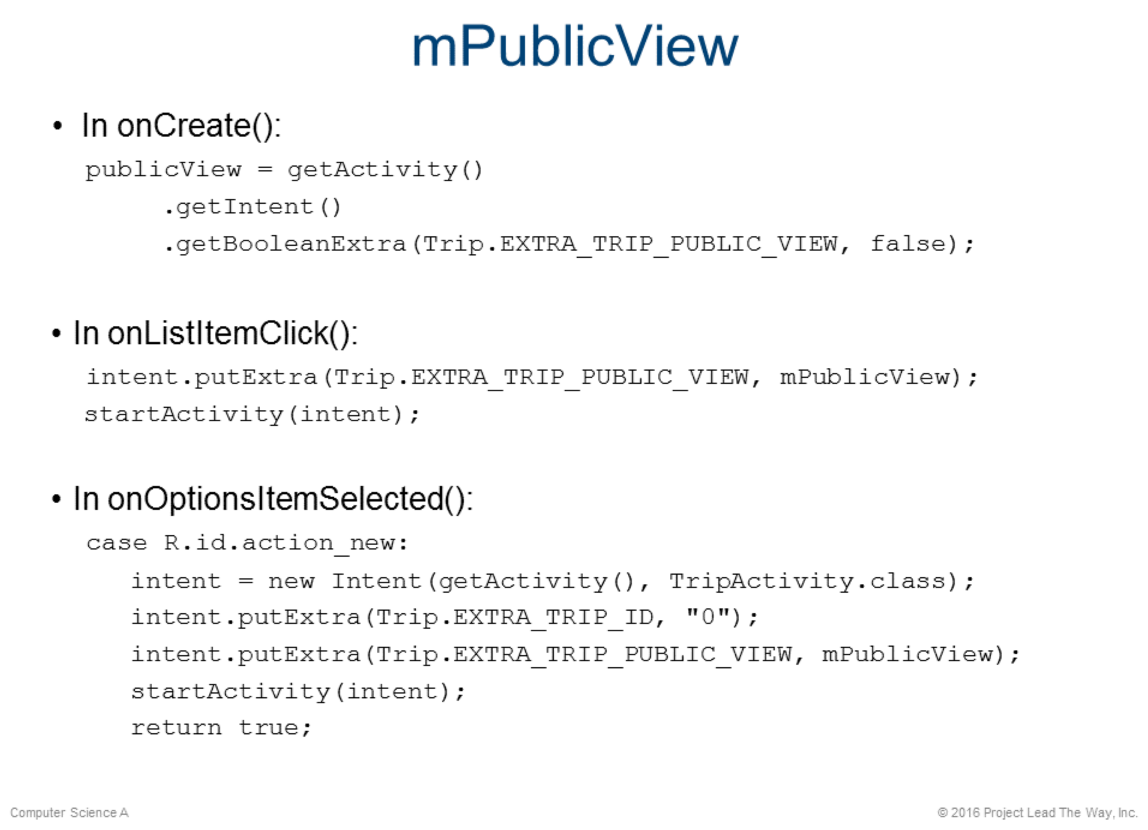
```java
@Override
public void onPrepareOptionsMenu(Menu menu){
    //toggle between Public Trips and My Trips action bar items
    if (mPublicView) {
        menu.findItem(R.id.action_public_trips).setVisible(false);
        menu.findItem(R.id.action_my_trips).setVisible(true);
    } else {
        menu.findItem(R.id.action_public_trips).setVisible(true);
        menu.findItem(R.id.action_my_trips).setVisible(false);
    }
}
```

The variable `mPublicView` is used to determine which menu item to show and which to hide; either My Trips will show or Public Trips will show, but not both.

6 For an overview of how the `mPublicView` attribute is used in the `TripListFragment` class, refer to slide 5 of *3.1.6 Public vs. Private Trips*.

a. What is the second parameter in `getBooleanExtra()` used for?

b. Explain how the `Trip.EXTRA_TRIP_PUBLIC_VIEW` is used in the code.



mPublicView

- In onCreate():

```
publicView = getActivity()
      .getIntent()
      .getBooleanExtra(Trip.EXTRA_TRIP_PUBLIC_VIEW, false);
```

- In onListItemClick():

```
intent.putExtra(Trip.EXTRA_TRIP_PUBLIC_VIEW, mPublicView);
startActivity(intent);
```

- In onOptionsItemSelected():

```
case R.id.action_new:
   intent = new Intent(getActivity(), TripActivity.class);
   intent.putExtra(Trip.EXTRA_TRIP_ID, "0");
   intent.putExtra(Trip.EXTRA_TRIP_PUBLIC_VIEW, mPublicView);
   startActivity(intent);
   return true;
```

Computer Science A

© 2016 Project Lead The Way, Inc.

The mPublicView indicates whether the user wants to:

- View the Public Trips, in which case the value of mPublicView would be true.
- See only his/her own trips, in which case the value would be false.

The value of the mPublicView attribute is set using the intent extra EXTRA_TRIP_PUBLIC_VIEW in the onCreate() method. The first time the user accesses the trips list (after logging in to the app), the intent extra will not be set. But the default value has been set to false with the call to getBooleanExtra() when you get the activity's intent. This means that by default, the view of the trips will be the user's own trips (i.e., My Trips).

The mPublicView attribute is also passed to the TripFragment (through the intent) when

- the user clicks on the list item to view a trip's details in onListItemClick
- the user creates a new trip and invokes the action_new menu item in onOptionsItemSelected

For further explanation of how getBooleanExtra() works, it often helps to use the online documentation summarized below and online at:

http://developer.android.com/reference/android/content/Intent.html

----

public boolean getBooleanExtra (String name, boolean defaultValue)

Retrieve extended data from the intent.

Parameters

name The name of the desired item.

defaultValue The value to be returned if no value of the desired type is stored with the given name.

Returns

The value of an item that was previously added with putExtra() or the default value if none was found.

----

7 After reviewing the notes on slide 5, modify the `onOptionsItemSelected` method in `TripListFragment`.

a. To start a new activity that loads the correct view of the trips list, add code for the menu items `action_public_trips` and `action_my_trips`.

b. When you create the intent, carefully consider the activity you want to specify. In other words, consider which activity (Trip or TripList) you should reference in the new intent.

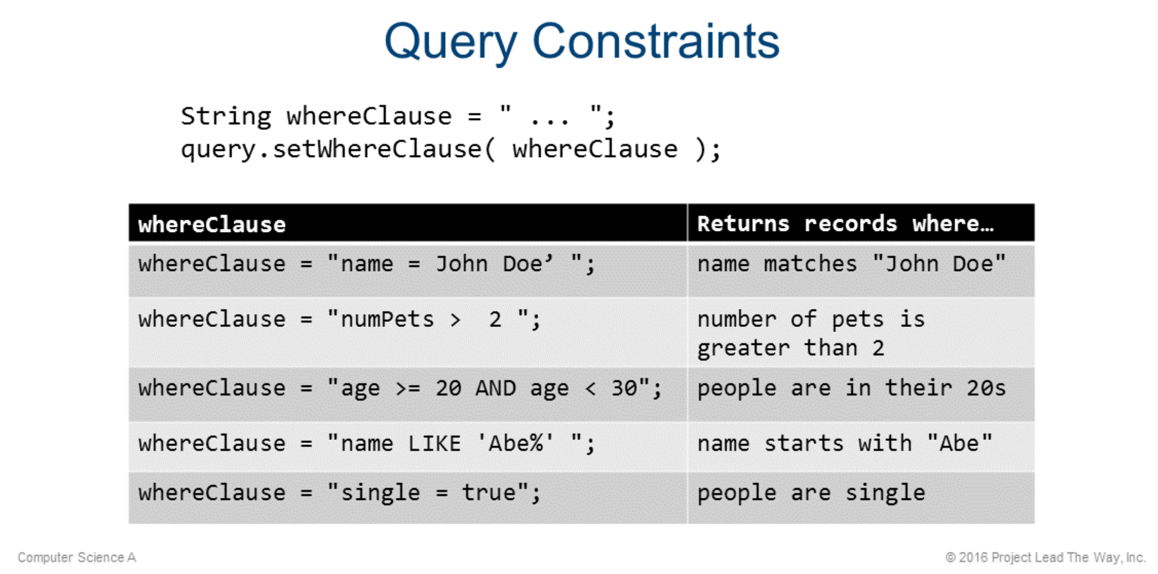
8 Test your app: Select the **My Trips** and **Public Trips** menu items and confirm that the screen title changes accordingly. Selecting the menu options will *not* cause the list of trips to change; that is coming up in subsequent steps.

Part II: Query Constraints and Options

9 For an overview of the different query constraints available in Backendless, refer to the slide below.

a. Below, write by hand a whereClause query constraint similar to the following Java code: `studentName.substring(0, 4).equals("Jill")`

b. Assuming “grade” is an integer column in a Student entity, which query constraint(s) would you use to retrieve all students in grades 9–12?



Query Constraints

```
String whereClause = " ... ";
query.setWhereClause( whereClause );
```

whereClause	Returns records where…
whereClause = "name = John Doe’ ";	name matches "John Doe"
whereClause = "numPets > 2 ";	number of pets is greater than 2
whereClause = "age >= 20 AND age < 30";	people are in their 20s
whereClause = "name LIKE 'Abe%' ";	name starts with "Abe"
whereClause = "single = true";	people are single

Computer Science A

© 2016 Project Lead The Way, Inc.

Without constraints, a query retrieves all records in the Trip entity.

Recall query statements you have used before, similar to:

String whereClause = “email='" + user.getEmail() + "'";

query.setWhereClause(whereClause);

String whereClause = “ownerId='" + user.getObjectId() + "'";

query.setWhereClause(whereClause);

The table shows other types of queries.

For further information, refer to Backendless documentation regarding Data Queries with the DataQueryBuilder object

10 In `refreshTripList()`, use a Backendless query constraint (a `whereClause`) based on the value of `mPublicView` to meet the following requirements:

- In the Public Trips screen, a `whereClause` that will show all shared/public trips, both trips the user owns and other user's public trips.
- In the My Trips screen, a `whereClause` that will show all of the user's trips, public or private.

Modification steps:

a. Create a `String` variable `whereClause`.

b. Based on the above requirements, assign a query constraint to the `whereClause` string.

c. Use `query.setWhereClause(whereClause)` to set the query constraint.

d. Issue the `find` passing the query as a parameter.

11 Test the app with public and private trips.

a. On the My Trips screen, confirm that all trips owned by the current user are shown.

b. On the Public Trips screen, confirm that public trips are shown.

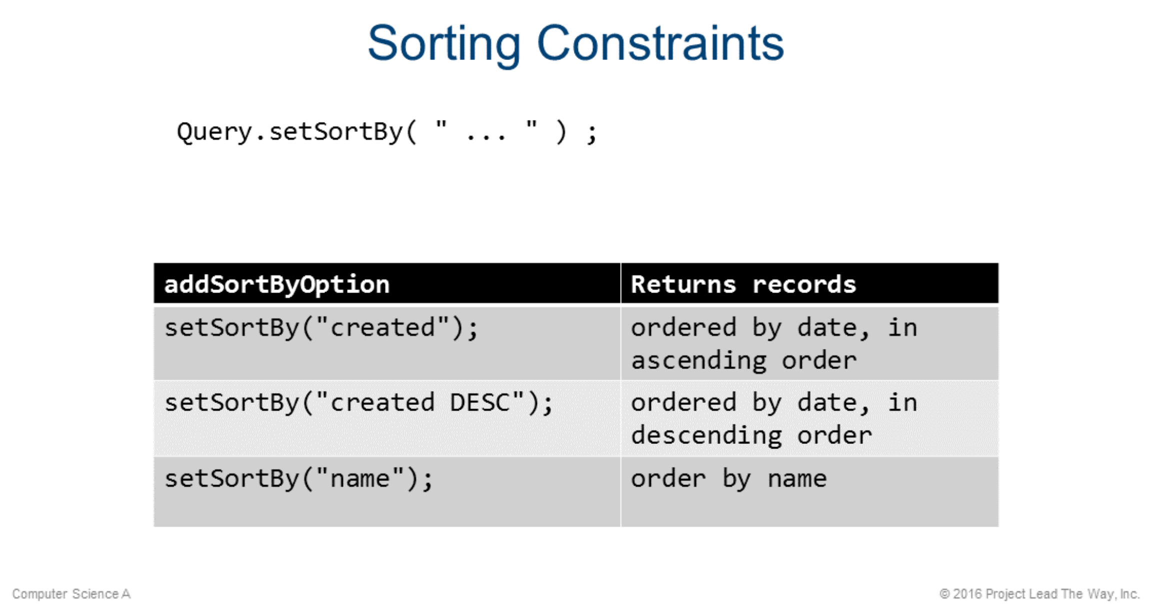
12 Log in as another user and create some public and private trips.

a. In My Trips, make sure you can only view the trips for the user who is currently logged in.

b. In Public Trips, make sure you can view all publicly shared trips (regardless of who owns them).

Perhaps you want to change the order in which your trips are shown—maybe by start date, alphabetically by name, or some other order.

13 Review Slide 7 in *3.1.6 Public vs. Private Trips.*



Sorting Constraints

```
Query.setSortBy( " ... " ) ;
```

addSortByOption	Returns records
setSortBy("created");	ordered by date, in ascending order
setSortBy("created DESC");	ordered by date, in descending order
setSortBy("name");	order by name

Computer Science A

© 2016 Project Lead The Way, Inc.

14 Add a **query option** to sort the results based on `startDate` (using the default, ascending order).

15 Test your app again to make sure the new sort order works.

Part III: Ownership and Trips

Try to delete a public trip that does not belong to the current, logged-in user. Users should not be able to delete other users' trips!

16 Recall the database column that tracks the owner of the trip in Backendless. What is its name?

Check your answer

ownerID

17 Fix this ownership bug.

a. In *`Trip.java`*, create an `ownerId` instance variable and a getter method for it. You will not be changing trip ownership, so do not provide a setter.

b. In *`IntentData.java`*, create a new constant, `EXTRA_TRIP_OWNER_ID`, to track the trip's owner.

c. *`InTripListFragment.java,`* put the `EXTRA_TRIP_OWNER_ID` on the intent along with other trip data.

d. Also in `TripListFragment` *a*, modify `deleteTrip`:

- Get the `CurrentUser()` from the `Backendless.UserService` object
- Now you will use your new `Trip` method, `getOwnerId`. Compare the current user's objectId to the ownerId of the trip you are trying to delete.
- If the current user owns the trip, proceed with the delete.
- If the user does not own the trip, display an AlertDialog to inform them they cannot delete a trip they do not own. You have string attributes for this in *`strings.xml`*.

e. Lastly, in *`TripFragment.java`* `onCreate`:

- Similar to other intent data, set a new variable `ownerId` to the owner ID you retrieve from the intent.
- Get the `CurrentUser()` from the `Backendless.UserService` object:

```
BackendlessUser user = Backendless.UserService.
CurrentUser();
```

- Set the `mEnabled` variable to false, if the app is in public mode and if the current user is not the owner of the trip:

```
if (mPublicView && !(user.getObjectId().equals(ownerId))) {
    mEnabled = false;
}

```

This will disable all of the Trip screen views.

18 Test your app again to make sure that you cannot delete other users' trips. Test from the Trip List and Trip screens. You may notice that you get the same "Permission Denied!" error if you try to save or delete another user's public trip in the Trip screen.

Starting in the `onListItemClick` in `TripListFragment`, trace your code into the Trip Details screen and explain how it prohibits non-owners from editing the trip.

In disabling the edit capability in public mode, you have disabled editing the trip for *everyone*, even if you own the trip. Owners should be able to edit their trips from either the private or public view.

19 Fix this error.

Hint: In `TripFragment onCreate`, you use intent data to get the owner of the trip. However, you did not *set* the intent data in `TripListFragment`. Add code so that the owner of the trip is also placed on the intent when a user selects a trip from the Trip List.

Part IV: Navigating Public and Private Modes

You might have noticed that when you select a trip in the Public Trips screen and save it, the view switches to the private My Trips. The same occurs when you select Up/Back on the action bar. The switch in view modes could be very confusing to your user.

Consider *why* the switch back to the private My Trips screen is happening.

The Intent is lost when the `TripFragment` activity finishes and returns to `TripListActivity`.

20 To fix this bug, when you save a trip, you need to allow the `TripFragment` activity to return the intent data to `TripListActivity`.

a. In `TripListActivity onListItemClick`, replace `startActivity(intent)` with the following:

```
startActivityForResult(intent, Activity.RESULT_OK);
```

b. Then, prepare `TripListFragment` to *get* the results by overriding the `onActivityResult` method.

```
@Override
public void onActivityResult(int requestCode,int
resultCode, Intent intent) {
    if (resultCode == Activity.RESULT_OK) {
        mPublicView = intent.getBooleanExtra(Trip.EXTRA_
        TRIP_PUBLIC_VIEW, false);
    }
}
```

c. In `TripFragment`, before the app returns to Trip List when you update the trip, pass the intent back by *setting* the results.

```
// set the intent to return the public/private view mode
// this activity must be started with startActivityForResult
Intent intent = getActivity().getIntent();
intent.putExtra(Trip.EXTRA_TRIP_PUBLIC_VIEW, mPublicView);
getActivity().setResult(Activity.RESULT_OK, intent);
```

With this addition, the `mPublicView` data will be carried over, via the intent, back to the activity when you save a trip.

However, when you use the Up/Back button instead of saving, the intent is *not* passed back. In the `TripFragment onOptionsItemSelected` method, the case for `R.id.home` navigates the app up/back to the parent activity.

d. Replace the line `NavUtils.navigateUpFromSameTask(getActivity());` with the same code you just used in step c.

e. Since you are using the same code two times, create a new method. Don't forget to include `getActivity.finish()`. This call to `finish` destroys the current activity and removes it from the Activity stack. Consequently, the previous activity (`TripListActivity`) resumes.

21 Where else do you need to provide this "finish with results" functionality? Search for `getActivity().finish()` and replace it with your new method.

22 Thoroughly test your app to ensure the private/public modes and trip ownership are working properly.

Part V: Cyber-sharing Means Cyber-responsibility

When you shared a trip to be public, everyone can see the trip data. Apps that allow open sharing of information like this are susceptible to misuse, including cyber bullying. People could post “ugly” data in a public trip to bully someone. Taking a personal stand against cyber bullying is more important than ever. Make the decision now, to never cyber bully, to stand up for your friends, and to always behave online as if you are face-to-face with the person.

23 With this “pledge” in mind, write the contents of an anti-bully agreement that a user must **accept** when they register for your app. They must agree to behave properly while posting on public trips.

24 (Optional) Create an anti-bully checkbox with an anti-bully message on the Sign-up page at registration time. Require the user to check it before they can sign up.

Part VI: (Optional) Themes and Other Improvements

Consider implementing some *optional* steps that will enhance and improve your app:

- In *`AndroidManifest.xml`*, experiment with different themes for a different look and feel to your app. For example, in the `application` tag, set the `theme` attribute to use another theme, such as `android:theme="style/Theme.AppCompat"`. (The exact theme depends on which API version you have, so choose one that is available to you.)
- When the user signs up, add a user-friendly feature that auto-fills the email and password on the login screen. This will require new intent data and constants.
- You have many alert dialogs throughout your app, especially if you continue to perform extra data validation. Make a static class with a static alert dialog that all methods can use. Change `LoginActivity` to use this new version in place of the local method that uses `AlertDialog`.
- Do some extra data validation
  - Do not allow a password to equal the user’s name.
  - Provide an extra TextEdit for a “confirm password” field.
  - Validate that the start date does not occur after the end date, or does not occur sometime in the past. Explore the `after` method of the Date class.

CONCLUSION

1. Using the relationships in TripTracker as an example, explain why entity relationships are necessary in a database. Give examples of entity relationships that can exist in a real-life computer system of your choice.
2. If you wanted to retrieve all trips you own and all of the trips of your best friend Jose, how would you define the query?