



Version 1: Spring '14

Database.com Workbook



Last updated: April 27, 2014

Table of Contents

About the Database.com Workbook.....	1
Database.com Workbook Concepts.....	2
Part 1: Database.com Fundamentals.....	3
Tutorial 1: Creating a Database and Test Database.....	3
Step 1: Log In to Database.com.....	3
Step 2: Create a Test Database.....	4
Step 3: Create an Object to Store Data.....	5
Step 4: Create Custom Fields for the Album Object.....	7
Summary.....	8
Tutorial 2: Using Workbench.....	8
Step 1: Launch Workbench.....	8
Step 2: Insert Data Records with Workbench.....	9
Step 3: Query Data Records with Workbench.....	10
Step 4: Update Data Records with Workbench.....	12
Summary.....	13
Tutorial 3: Creating and Relating Objects.....	13
Step 1: Create and Relate a Detail Object.....	13
Step 2: Insert and Query Detail Object Records.....	14
Step 3: Add a Validation Rule.....	16
Step 4: Test the Validation Rule.....	16
Step 5: Create a Roll-up Summary Field in a Master Object.....	17
Summary.....	18
Tutorial 4: Managing Users, Data Visibility, and Data Sharing.....	18
Step 1: Create a New User and Profile.....	18
Step 2: Toggle Object Visibility.....	20
Step 3: Set Field Level Security.....	21
Step 4: Specify Organization-Wide Default Sharing Rules.....	21
Using Hierarchies.....	23
Summary.....	23
Tutorial 5: Using Workflow Logic.....	23
Step 1: Create a Workflow Rule.....	24
Step 2: Send an Outbound Message.....	24
Step 3: Set Up Network Security.....	25
Step 4: Test the Outbound Messaging.....	26
Summary.....	27
Tutorial 6: Creating and Deploying Change Sets.....	28
Step 1: Connect to the Production Database.....	28
Step 2: Configure a Deployment Connection.....	33
Step 3: Create and Upload an Outbound Change Set.....	28
Step 4: Validate and Deploy Inbound Change Set.....	30
Step 5: Disallow Inbound Changes.....	31
Summary.....	31

Tutorial 7: Using Custom Business Logic and Database Triggers.....	31
Step 1: Create a Trigger.....	32
Step 2: Logically Test the Trigger.....	33
Step 3: Programmatically Test the Trigger.....	33
Step 4: View Recycle Bin Records and Restore Them.....	34
Step 5: Move the Trigger and Unit Tests to Production.....	35
Summary.....	36
Part 2: External Application Integration.....	37
Tutorial: Creating a Java Web Application to Access the Database.....	37
Prerequisites.....	37
Step 1: Configure OAuth on Database.com.....	37
Step 2: Download and Compile the Java Application.....	38
Step 3: Configure and Run the Java Application.....	39
Step 4: Run the Application.....	40
Step 5: Test the Application.....	40
Step 6: Investigate How It Works.....	42
Summary.....	44
Next Steps.....	45

About the Database.com Workbook

The Database.com Workbook walks you through creating a database via a series of tutorials. The tutorials are centered around building a very simple system for a music store. You'll start developing the database from basic building blocks; that is, you'll first build objects for keeping track of audio tracks, which are the store's merchandise. You'll continue by adding security profiles and the relationships between the objects, such as a master-detail relationship. Next, you'll add business logic: validation rules to ensure data integrity, workflow rules to spot unusual conditions, and data sharing settings to control access. Once the database and business logic are complete, you'll have a complete database system.

Each of the tutorials builds on the previous tutorial to advance the database's development and simultaneously showcase a particular feature of the platform. It might sound like a lot, but it's all quite easy—as you'll soon see.

What is Database.com?

Database.com is a cloud database that makes it easy to build collaborative, mobile enterprise applications. Secure sharing of data is at the heart of collaborative enterprise applications. The sophisticated identity and access management capabilities in Database.com ensure that you can get the right data, to the right people, at the right time, with the right access permissions. This workbook is intended to help you explore and learn about some of the unique capabilities of Database.com. Please look for other articles and future versions of this workbook which will highlight the Database.com unique social API set and the ability to create custom APIs using Apex REST.



Tip: You'll find the latest version of this workbook, as well as snippets of the code that you can easily copy, online at: www.database.com/workbook.

Intended Audience

This workbook is intended for experienced developers new to the Database.com platform. The workbook's code examples are in Java, so familiarity with Java is helpful, though all the programming tasks can also be done with .NET, Ruby, and other development tools.

Before You Begin

This workbook is designed to be used with a Base Edition organization, or *Base org* for short. Base orgs are multipurpose environments with all of the Database.com features and permissions that allow you to develop and test your database.

1. In your browser go to www.database.com.
2. Click **Signup**.
3. Fill in the fields about you and your company.
4. In the `Email Address` field, make sure to use a public address you can easily check from a Web browser.
5. The `Username` field is also in the *form* of an email address, but it does not have to be the same as your actual email address, or even an email that you use. It's helpful to change the username to something that describes the use of the organization. In this workbook we'll use `admin-user@workbook.db`.
6. Enter the Captcha words shown.
7. Read and then select the checkbox for the `Master Subscription Agreement` and supplemental terms.
8. Click **Sign Up**.
9. After signing up, you'll be sent an email with a link that you must click to verify your account. Click the link.
10. Now supply a password, and a security question and answer.

Database.com Workbook Concepts

To reflect the added functionality provided by metadata-driven features, Database.com uses different terminology compared to a relational database.

Relational Database Term	Equivalent Term in Database.com
Database	Organization
Table	Object
Column	Field
Row	Record

In a relational database, tables contain columns (to define the data types) and rows (to store the data). You relate tables to other tables by using primary keys and foreign keys, which map the rows of one table to the rows of another table.

In Database.com, an *organization* is the equivalent of a database, but with built-in user identity, security, and social features. *Objects* contain *fields* and *records*. You relate objects to other objects by using *relationship fields*, such as *lookup relationships* and *master-detail relationships*, instead of primary and foreign keys.

Part 1: Database.com Fundamentals

In this part of the Database.com Workbook, you learn how to create a database, a test database, and objects that store data within the database. You'll also learn how to create and manage users, set up user authentication, write database triggers, and configure workflow and data sharing.

To show you how the pieces work together, each tutorial builds on the same scenario: managing albums and tracks for an online music business. You'll create a simple database that stores album and track data in two related objects. The story unfolds as you continue working—so let's begin!

Tutorial 1: Creating a Database and Test Database

In this tutorial, you create a database in the cloud. After signing up for Database.com, you'll get your first look at the Database.com Console, and then use it to create both a test database environment for development and an object to store data.

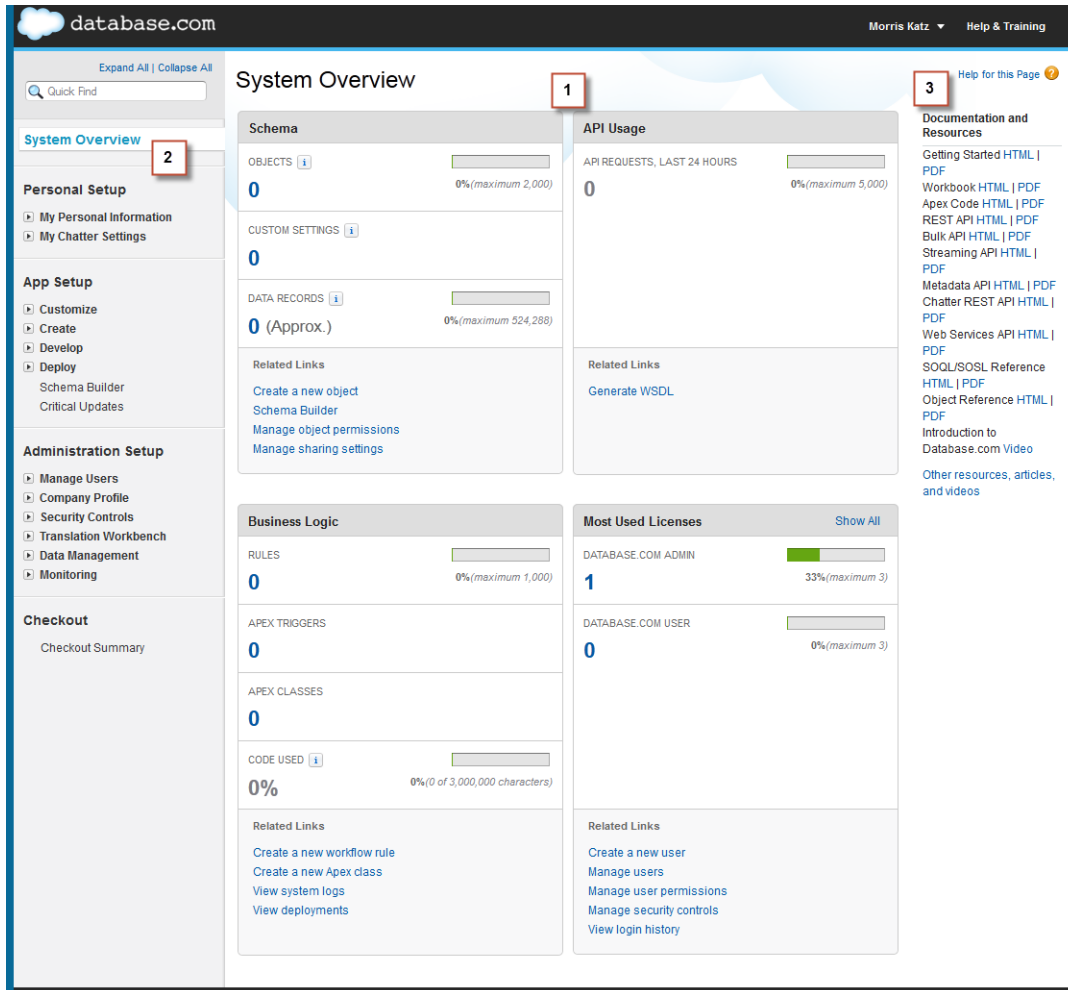
Step 1: Log In to Database.com

If you haven't already created a Database.com organization, do so now by following the procedure at [Before You Begin](#) on page 1.

If you're not already logged in to your production database, log in now.

1. Go to login.database.com.
2. Enter your user name and password.

That's it—you're now logged in to your production database. The first thing you'll see is the System Overview page.



Here's a quick guide to the System Overview page.

1. The primary section provides a dashboard of what's happening in your database. It provides quick summaries and handy links to your schema's objects and storage usage, business logic, such as database triggers and classes, API statistics, and your users.
2. The menus on the left provide access to the primary aspects of Database.com functionality, as well as administration configuration where you can manage users, data security, logs, and more.
3. The right-hand sidebar provides you with helpful links to documentation related to Database.com.

You can log in to the production database again at any time by navigating to login.database.com.

Step 2: Create a Test Database

Several databases are necessary during a typical Application Development Life Cycle (ADLC). These are development, test, and production databases. The cycle usually goes something like this:

1. Developers make a structural copy of the current production database, load it with a minimal amount of data, and then use this as a development database to support new development efforts.
2. Once the development iteration completes, Quality Assurance (QA) engineers make a copy of the development database and use it to test the latest version of the application.
3. Once QA testing completes, administrators roll the latest database changes into the production database to support the added application features.

4. The cycle repeats with each development effort.

After you sign up and first log in to Database.com, you are automatically working with your production database, also referred to as your production *organization*. To support the typical ADLC, Database.com allows your organization to create *test databases*.

Now create a test database to use in subsequent tutorials in this workbook.

1. Click **Data Management > Test Database**.
2. Click **New Test Database**.
3. For **Name**, enter `Workbook`. Ensure that **Type** is set to `QA Database`, and accept the defaults for everything else.
4. Click **Start Copy**.



Note: There are two types of test databases. A QA test database is a copy of your current production database configuration and code, but no data. A staging test database replicates all of your data as well. By default, you can create a single test database with Database.com. If you need additional test databases, please contact salesforce.com support.

Test database creation time varies. You'll get email from Database.com once the process completes.

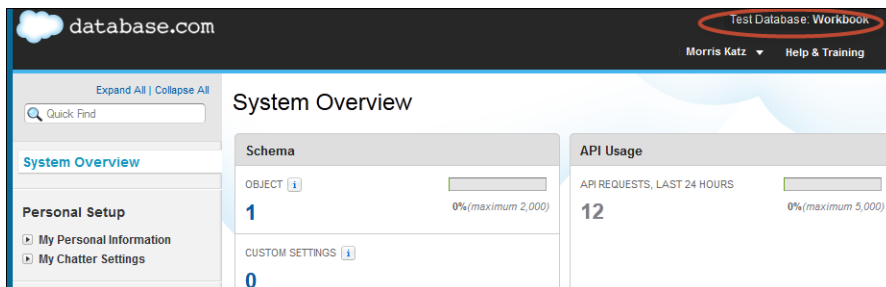
When Database.com creates a test database, it creates a user in the test database by appending `.<testdatabasename>` to your production database username. For example:

Production Username	Workbook Test Database Username
<code>admin-user@workbook.db</code>	<code>admin-user@workbook.db.workbook</code>

The password for the corresponding administrator account in the test database will be the same as it is in production. Once you receive an email that the test database is ready, you can use the Database.com Console to connect to the test database. Navigate to the default deployment connection.

1. Open a new browser window and load the URL test.database.com.
2. Enter the test database username and password, and log in.

Once you connect to your test database, look at the top right portion of your screen and confirm that you see an indicator that you are using your test database.



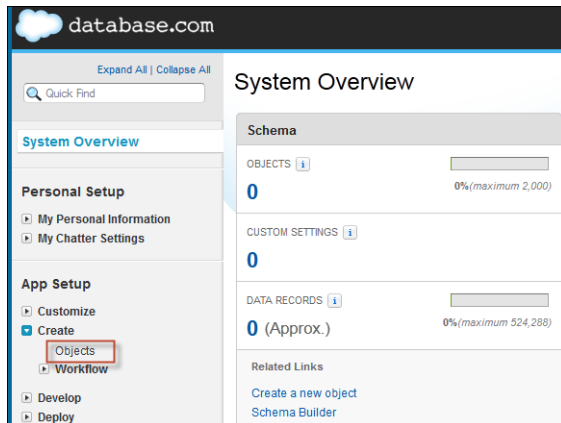
Step 3: Create an Object to Store Data

If you are familiar with relational databases, you'll understand the concept of *tables*—structures that store data. A table contains one or more *columns* (fields) and zero or more *rows* of data (records).

With Database.com, the traditional table data structure is referred to as an *object*. Every object in Database.com has a few standard fields, and can have custom fields that you create to store various types of data (numbers, text, dates, and so on).

Create an object in your test organization called `Album` that will be used to store data about music albums.

1. Click **Create > Objects**.



2. Click **New Custom Object**.
3. For **Label**, type Album, then press TAB to set the **Object Name**.
4. For **Plural Label**, type Albums.
5. Select **Starts with a vowel sound**.
6. For **Record Name**, type Album Name and set its **Data Type** to Text.

7. Leave all other values as they are.
8. Click **Save** to finish creating the object.

You now have an object that can store data. The Album object has a field called Album Name that can store text strings and that you can set.

Custom Object
Album

Custom Object Definition Detail [Edit](#) [Delete](#)

Singular Label	Album	Description	
Plural Label	Albums	Track Field History	<input type="checkbox"/>
Object Name	Album	Deployment Status	Deployed
API Name	Album__c		
Created By	Morris Katz, 7/11/2012 9:57 AM	Modified By	Morris Katz, 7/11/2012 9:57 AM

Standard Fields [Standard Fields Help](#)

Action	Field Label	Field Name	Data Type	Controlling Field
Edit	Album Name	Name	Text(80)	
	Created By	CreatedBy	Lookup(User)	
	Last Modified By	LastModifiedBy	Lookup(User)	
Edit	Owner	Owner	Lookup(User:Queue)	

All Database.com objects automatically include an ID field that contains a unique identifier for each record in the object. This field is analogous to a typical primary key in a relational database table. Objects also include other standard fields, such as Created By, Last Modified By, and Owner. Database.com automatically manages these fields.

Step 4: Create Custom Fields for the Album Object

The Album object already has an Album Name field. In this step, you add two custom fields to go along with the standard fields that are already in the object.

Create the Released On field.

1. If you're not already viewing the Custom Object–Album page, click **Create > Objects** and then click **Album** on the Custom Objects page.
2. Scroll down to the **Custom Fields & Relationships** section.
3. Click **New** to launch the New Custom Field wizard.

Notice the many different data types that Database.com supports.

4. For this field's **Data Type**, select **Date**, and click **Next**.
5. For **Field Label**, enter **Released On**, then press TAB to set the **Field Name**, and then click **Next**.
6. Click **Save** to accept the security defaults.

Create the Description field.

1. In the **Custom Fields & Relationships** section, click **New**.
2. For **Data Type**, select **Text Area (Long)**, and click **Next**.
3. For **Field Label**, enter **Description**.
4. For **Length**, enter 5000.
5. Click **Next** to accept the remaining defaults, and click **Save** to accept the security defaults.

Custom Fields & Relationships New Field Dependencies Custom Fields & Relationships Help					
Action	Field Label	API Name	Data Type	Controlling Field	Modified By
Edit Del	Description	Description__c	Long Text Area(5000)		Morris Katz, 7/11/2012 10:02 AM
Edit Del	Released On	Released_On__c	Date		Morris Katz, 7/11/2012 10:01 AM

You now have an object that can store the name, release date, and description of music albums, as well as other standard field data such as each data record's owner—which by default is the user who created the data record. Later on, we'll add another field to Album called Price and automatically calculate each album's price based on the aggregate price of individual music tracks.



Note: For each object and field in Database.com, there is a label (the name you assigned) and an *API name* (by default, a system-generated name). The API names for custom objects and fields end with a “__c” suffix to prevent naming

conflicts with system objects and fields. Standard fields do not have this suffix. The following tutorials reference these API names when building formulas and coding database triggers.

Leave the Database.com Console open so that you can quickly switch back to it in subsequent tutorials.

Summary

Congratulations—you've created your production and test database environments. You've also learned about objects, fields, records, and default values as you created your first object, Album, to store information about music albums.

Now let's learn how to browse and manage data using Workbench.

Tutorial 2: Using Workbench

In this tutorial, you learn how to manage and query records in Database.com objects using Workbench. Workbench is a powerful, web-based suite of tools that administrators and developers can use to interact with environments built on Database.com. You'll see how you can use Workbench to describe, query, and manipulate database objects. Along the way, you'll learn more about Database.com.



Note: Workbench is a free, open source, community-supported tool (see the Help page in Workbench). Salesforce.com provides a hosted instance of Workbench for demonstration purposes only—salesforce.com recommends that you do not use this hosted instance of Workbench to access data in a production database. If you want to use Workbench for your production database, you can download, host, and configure it using your own resources.

Step 1: Launch Workbench

Before launching Workbench, make sure you are logged in to your test database in only one browser tab.

To get started with Workbench, use the hosted version. In a new browser tab, navigate to: workbench.developerforce.com.

Once Workbench loads, it presents a login screen. Use this screen to connect to your database.

1. For **Environment** select **Sandbox**.
2. Accept the terms of service and click **Login with Salesforce**.

Once you successfully establish a connection to your test database using Workbench, you land on the Select page, which you can use to quickly jump to whatever interests you in your database.

workbench info queries data migration utilities

MORRIS KATZ AT PRODUCTION ON API 25.0

Select an action to perform:

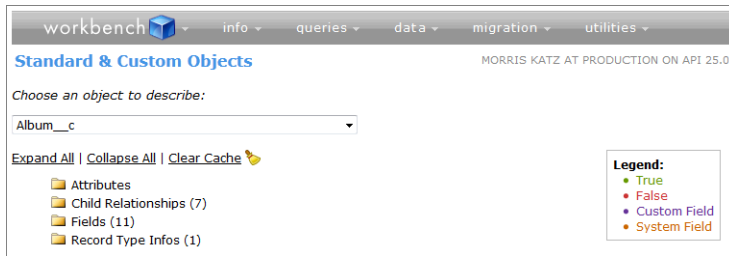
Jump to: [dropdown]

Object: [dropdown]

Select

For example, let's display summary information about our Album object.

1. For **Jump to**, select **Standard & Custom Objects**.
2. For **Object**, select **Album__c**, the API name for the Album object.
3. Click **Select**.



The Describe page reveals useful metadata that corresponds to the chosen object. Expand the component folders to review their contents.

- **Attributes** displays general information about an object, including its API name.
- **Child Relationships** lists information for all related objects, including relationships to system objects.
- **Fields** lists information for all standard and custom fields in an object, including each field's label, API name, and data type.
- **Record Type Infos** is an array of the record types supported by this object. You don't need access to all the returned record types to see them here. This object contains all of the existing fields of `RecordTypeMapping` except `layoutId` and `picklistForRecordType`.

Step 2: Insert Data Records with Workbench

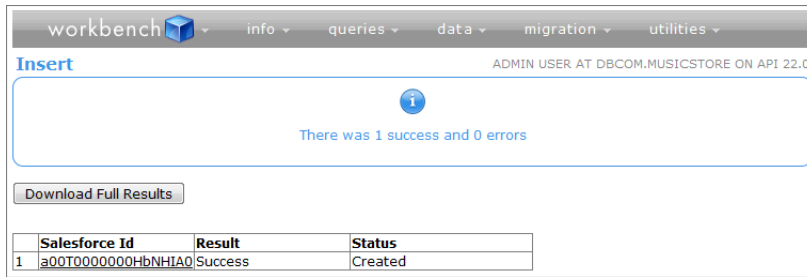
You can use Workbench to insert data into the Album object by creating a few records. To insert the first record, complete the following steps:

1. With the **Album__c** object selected on the Standard & Custom Objects page, click **data > Insert**.
2. On the Insert page:
 - a. For **Object Type**, select **Album__c**.
 - b. Select **Single Record**.
 - c. Click **Next**.
3. On the following Insert page:
 - a. For **Description__c**, enter `Great cover art`.
 - b. For **Name**, enter `Who's Next`.
 - c. Leave **OwnerId** empty so that Database.com auto-assigns your user Id to this field.
 - d. For **Released_On__c**, enter `1971-08-14`.
 - e. Click **Confirm Insert**.

Field	Value	Smart Lookup ?
Description__c	Great cover art	
Name	Who's Next	
OwnerId		
Released_On__c	1971-08-14	

Confirm Insert

Workbench returns a message upon completion. Your Salesforce Id value will be different than the one shown here.



Now repeat these steps to create more records with the following data so that you have some varying data to work with in the next part of this tutorial.

Description__c	Name	Released_On__c
A classic	Help!	1965-08-06
Great rock opera	The Wall	1979-11-30

Step 3: Query Data Records with Workbench

Now let's learn how to query Database.com with Workbench to retrieve records in an object. To build your first query, complete the following steps.

1. On the Workbench main menu, select **queries** > **SOQL Query** to display the Query page.
2. For **Object**, select Album__c.
3. Build a database query by selecting multiple fields in the **Fields** selection box. You can select more than one field by holding down the CTRL key and clicking the field names. For this example, select the following fields and notice how they appear in the query field.
 - Id
 - Description__c
 - Name
 - Released_On__c
4. Edit the query that Workbench builds to rearrange the order of the columns in the SELECT list. When you are finished, the query resembles the following:

```
SELECT Id, Name, Description__c, Released_On__c FROM Album__c
```

5. Click **Query** to execute the query.

Here's what your query results look like in Workbench (your Ids will be different).

workbench info queries data migration utilities

SOQL Query MORRIS KATZ AT PRODUCTION ON API 25.0

Choose the object, fields, and criteria to build a SOQL query below:

Object: View as: ☒ List ☐ Matrix ☐ CSV ☐ Bulk CSV ☐ Bulk XML Deleted and archived records: ☒ Exclude ☐ Include

Fields: CreatedById CreatedDate Description__c Id IsDeleted

Sort results by: A to Z Nulls First Max Records:

Filter results by:

Enter or modify a SOQL query below:

```
SELECT Id, Name, Description__c, Released_On__c FROM Album__c
```

Run: Save as:

Query Results

Returned records 1 - 3 of 3 total records in 0.362 seconds:

Id	Name	Description__c	Released_On__c
1a00TV000000HbNwYA0	Who's Next	Great cover art	1971-08-14
2a00TV000000HbNXYA0	Help!	A classic	1965-08-06
3a00TV000000HbNBAYAK	The Wall	Great rock opera	1979-11-30

The Database.com query language is SOQL. Notice that SOQL is similar to SQL, so getting started with SOQL is easy if you already understand SQL.



Note: In our scenario, we only have three records in the Album object. When querying larger data sets, consider using the LIMIT clause in a SELECT statement to limit the number of records in a query's return set.

Now let's add a few bells and whistles to your first query.

1. On the SOQL Query page, the current query should still be present. In **Sort results by:**, choose `Released_On__c`.

Your query now looks like the following (reformatted below for readability).

```
SELECT Description__c, Id, Name, Released_On__c
FROM Album__c
ORDER BY Released_On__c ASC NULLS FIRST
```

2. Click **Query** to run the new query and notice that the query results are now ordered by the `Released_On__c` field in ascending order.

Query Results

Returned records 1 - 3 of 3 total records in 0.430 seconds:

Id	Description__c	Name	Released_On__c
1a00T0000000HbNMIA0	A classic	Help!	1965-08-06
2a00T0000000HbNIIA0	Great cover art	Who's Next	1971-08-14
3a00T0000000HbNNIA0	Great rock opera	The Wall	1979-11-30

Let's enhance the query a bit more, so that it shows only specific records that meet a condition. Use **Filter results by:** to see if you can build the following query, one that shows albums that were released before 1970-01-01.

```
SELECT Description__c, Id, Name, Released_On__c
FROM Album__c
WHERE Released_On__c < 1970-01-01
ORDER BY Released_On__c ASC NULLS FIRST
```

When you execute the query, you'll only see one record in the result set, the data record for the album named "Help!" The description for this album can be more helpful (pun intended), and indicate that this particular record corresponds to the North American release of the album and not the British release.

workbench info queries data migration utilities

SOQL Query MORRIS KATZ AT PRODUCTION ON API 25.0

Choose the object, fields, and criteria to build a SOQL query below:

Object: **Album__c** View as: ☒ List ☐ Matrix ☐ CSV ☐ Bulk CSV ☐ Bulk XML Deleted and archived records: ☒ Exclude ☐ Include

Fields: **count()** **CreatedBy** **CreatedDate** **Description__c** **Id** **IsDeleted**

Sort results by: **Released_On__c** **A to Z** **Nulls First** Max Records:

Filter results by: **Released_On__c** **<** **1970-01-01**

Enter or modify a SOQL query below:

```
SELECT Description__c,Id,Name,Released_On__c FROM Album__c WHERE Released_On__c < 1970-01-01 ORDER BY Released_On__c ASC NULLS FIRST
```

Query **Reset** Run: Save as: **Save** **Clear All**

Query Results

Returned records 1 - 1 of 1 total record in 0.111 seconds:

Id	Description__c	Name	Released_On__c
1a00TV000000HbNXYA0	A classic	Help!	1965-08-06

Next you'll learn how to update a data record quickly using Workbench.

Step 4: Update Data Records with Workbench

Workbench makes it easy to manage data records in Database.com. You'll now use Workbench to improve the album description that we discovered in the previous step.

Picking up from where you left off in the previous step, notice that the Id field for each record in a query's result set is actually a hyperlink. If you hover over a link, text appears that provides you with links to Update, Delete, Undelete, or Purge the associated data record. The View in Salesforce link opens the record in the Database.com console. If you click the Id link itself, Workbench loads a detail page for the record.

Query Results

Returned records 1 - 1 of 1 total record in 0.111 seconds:

Id	Description__c	Name	Released_On__c
1a00TV000000HbNXYA0	A classic	Help!	1965-08-06

Choose an action:
[Update](#) [Delete](#) [Undelete](#) [Purge](#) [View in Salesforce](#)

In this step, we want to update the record.

1. Click the **Update** link to display the Update page.
2. Update the **Description__c** field so that it is North American release.

workbench info queries data migration utilities

Update MORRIS KATZ AT PRODUCTION ON API 25.0

Provide values for the Album__c fields below:

Field	Value	Smart Lookup
Id	a00TV000000HbNXYA0	
Description__c	North American release	
Name	Help!	
OwnerId	005D0000001LY8UIAW	
Released_On__c	1965-08-06	

Confirm Update

3. Once you've made the change, click **Confirm Update** to execute the update.

You've finished this tutorial! Leave Workbench open so that you can quickly switch back to it in subsequent tutorials.

Summary

In this tutorial, you learned how to work with data in your organization's data objects using Workbench, an open source tool that you can use to access Database.com databases. Specifically, you learned how to insert, query, and update records in an object. In the next lesson, you'll switch back to the Database.com Console and continue building your schema.

Tutorial 3: Creating and Relating Objects

In this tutorial, you create a second object, *Track*, representing music tracks in an album. You'll relate this new object to the *Album* object in a master-detail relationship, and learn about field validation and roll-up summary fields.

Step 1: Create and Relate a Detail Object

If you're familiar with relational databases, you'll notice a difference in Database.com when it comes to relating objects (tables). In Database.com, you don't work with primary and foreign keys, but instead with a higher-level abstraction of object relationships. You can declare a field in an object with a *relationship type*, and Database.com takes care of the underlying implementation for you.

In this step, you create a new object to represent music tracks in an album, and relate this new *Track* object to the existing *Album* object. Database.com supports two types of relationships: *lookup relationships* and *master-detail relationships*. In master-detail relationships, which we'll use for *Track* and *Album*, the detail (*Track*) derives its significance from the master (*Album*). In other words, tracks can't exist without albums.

From the Database.com Console using your test database, create and relate the new *Track* object.

1. Navigate to **Create > Objects**.
2. Click **New Custom Object**.
3. For **Label**, enter *Track*, and then press TAB.
4. For **Plural Label**, enter *Tracks*.
5. Click **Save** to accept the remaining defaults and create the *Track* object.

Custom Object Definition
Track

Custom Object Definition
Detail [Edit](#) [Delete](#)

Singular Label	Track	Description
Plural Label	Tracks	Track Field History <input type="checkbox"/>
Object Name	Track	Deployment Status Deployed
API Name	Track__c	
Created By	Morris Katz, 8/14/2012 4:03 PM	Modified By Morris Katz, 8/14/2012 4:03 PM

Standard Fields [Standard Fields Help](#)

Action	Field Label	Field Name	Data Type	Controlling Field
	Created By	CreatedBy	Lookup(User)	
	Last Modified By	LastModifiedBy	Lookup(User)	
Edit	Owner	Owner	Lookup(User,Queue)	
Edit	Track Name	Name	Text(80)	

Now create two fields—one for the price of the track, and another representing the relationship to the album.

1. In the **Custom Fields & Relationships** section, click **New**.
 - a. Select **Currency** as the data type and click **Next**.
 - b. For **Field Label**, type *Price*, and then press TAB.
 - c. For **Length**, type 5.
 - d. For **Decimal Places**, type 2.

- e. For **Default Value**, type 0.99.
 - f. Click **Next**, accept the security defaults, and click **Save**.
2. In the **Custom Fields & Relationships** section, click **New**.
 - a. For **Data Type**, select **Master-Detail Relationship**, and click **Next**.
 - b. In **Related To**, choose **Album**, and click **Next**.
 - c. For **Field Label**, leave the default as **Album**, and press TAB for the next field, ensuring that the **Field Name** has a value of **Album** too.
 - d. Click **Next**, accept the security defaults, and click **Save**.

The Field Label in the relationship field is how you will traverse to the related master record from any given detail record. You have now created a new object named **Track**, as well as two fields, **Price** and **Album**.

Custom Object Definition
Track

Custom Object Definition
Detail [Edit] [Delete]

Singular Label	Track	Description	
Plural Label	Tracks	Track Field History	<input type="checkbox"/>
Object Name	Track	Deployment Status	Deployed
API Name	Track__c		
Created By	Morris Katz	Modified By	Morris Katz

Standard Fields [Standard Fields Help ?]

Action	Field Label	Field Name	Data Type	Controlling Field
[Edit] [Del]	Created By	CreatedBy	Lookup(User)	
[Edit] [Del]	Last Modified By	LastModifiedBy	Lookup(User)	
[Edit] [Del]	Track Name	Name	Text(80)	

Custom Fields & Relationships [New] [Field Dependencies] [Custom Fields & Relationships Help ?]

Action	Field Label	API Name	Data Type	Controlling Field	Modified By
[Edit] [Del]	Album	Album__c	Master-Detail(Album)		Morris Katz
[Edit] [Del]	Price	Price__c	Currency(5, 2)		Morris Katz

Step 2: Insert and Query Detail Object Records

Now let's go back to Workbench and use it to insert some data records into our new **Track** object. In this step, you'll create three tracks for the Album named "The Wall." To create the first track, complete the following steps.

1. Return to the Workbench session that's connected to your test database.
2. On the Workbench main menu, select **data > Insert**.
3. For **Object Type**, select **Track__c**, select **Single Record**, then click **Next**.



Note: If you don't see the changes documented in this workbook when using Workbench, try clearing Workbench's cache to refresh your session—select **info > Session Information** and click **Clear Cache**. Retry your Workbench operation.

4. For the **Album__c Value** field, type **The Wall** and then select **Album__c.Name** in **Smart Lookup**.
5. For **Name**, type **01 - In The Flesh**.
6. For **Price__c**, don't enter anything — we'll use the default value of 0.99.
7. Click **Confirm Insert**.

Repeat the steps above using the following values for the **Name** and **Price** fields to create two more records in the **Track** object.

Name	Price
02 - The Thin Ice	0.59

Name	Price
03 – Another Brick in the Wall (Part 1)	1.49

We could have just skipped entering a price for each new record to defer to the default value of 0.99, but we'll put in some varying prices so that subsequent queries are a bit more interesting.

Now let's learn how to build some new queries with Workbench that target the records in the Track object.

1. Select **queries > SOQL Query**.
2. Execute the following query to summarize information about specific records in the Track object.

```
SELECT count(Id), max(Price__c), min(Price__c), sum(Price__c)
FROM Track__c
WHERE Album__r.Name = 'The Wall'
```

The results appear as follows.

The screenshot shows the Salesforce Workbench interface for running an SOQL query. The query is: `SELECT count(Id), max(Price__c), min(Price__c), sum(Price__c) FROM Track__c WHERE Album__r.Name = 'The Wall'`. The results table shows one record with the following values: `Unknown_Field__1` (1), `Unknown_Field__2` (1.49), `Unknown_Field__3` (0.59), and `Unknown_Field__4` (3.07).

Unknown_Field__1	Unknown_Field__2	Unknown_Field__3	Unknown_Field__4
1	1.49	0.59	3.07

In the query itself, notice the following new concepts to understand about SOQL.

- The **SELECT** clause uses several functions (count, max, min, and sum) to aggregate information about the selected records in the Track object.
- The **WHERE** clause condition employs an object traversal with dot notation to target information in the Name field of the related Album object. Notice the “__r” suffix to indicate that this is a relationship. This points out one difference between SQL and SOQL: With SQL, you can use a subquery in the WHERE clause condition to build an equivalent query.

One more example. Execute the following query.

```
SELECT Id, Name, Price__c
FROM Track__c
WHERE Album__r.Name = 'The Wall'
ORDER BY Name ASC
```

This SOQL query is the equivalent of an inner join in SQL. Notice again how it uses dot notation in the WHERE clause to specify an object traversal. Your results of this query will be similar to the following output.

Query Results

Returned records 1 - 3 of 3 total records in 0.205 seconds:

Id	Name	Price__c
1a01T0000000UWbqIAG	01 - In The Flesh	0.99
2a01T0000000UWbhIAG	02 - The Thin Ice	0.59
3a01T0000000UWbIIAG	03 - Another Brick in the Wall (Part 1)	1.49

Step 3: Add a Validation Rule

Database.com lets you create database-centralized validation rules to help ensure the integrity of data records. It's easy to create such rules because you simply declare them for an object (rather than build and maintain code to enforce them). In this step, modify the Track object to ensure that the price is never negative.

First, return to your browser window that has the Database.com Console connection to your test database. If you closed this window and don't remember how to get back to your test database with the Database.com Console, please review [Tutorial 1: Creating a Database and Test Database](#) on page 3. Once you have the Database.com Console available, complete the following steps.

1. Click **Create > Objects**.
2. Click **Track**.
3. In **Validation Rules**, click **New**.
4. For **Rule Name**, type `No Negative Prices`.
5. For **Error Condition Formula**, enter `Price__c < 0`.
6. Click **Check Syntax** and make sure you have no errors in your formula.
7. For **Error Message**, enter `Too cheap!`.
8. Click **Save** to finish creating the object.

The error condition formula specifies the condition under which the rule fires. In this case, it fires when the `Price__c` field is below zero.



Note: The Database.com formula language is very rich, letting you look up and manipulate values used in the rule. You'll use the same formula language in fields of type `Formula`, as well as in the Workflow functionality in [Tutorial 6](#) on page 28.

Once you save the validation rule, you'll see a summary page for the new rule. Notice that the rule is active, by default. Database.com enforces validation rules that are active, but allows you to deactivate them when you might feel it is necessary (for example, prior to a bulk data load).

Track Validation Rule [Help for this Page](#)

[Back to Track](#)

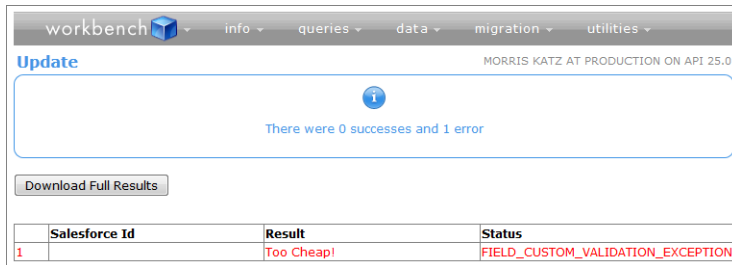
Validation Rule Detail		Edit	Clone
Rule Name	No_Negative_Prices	Active	✓
Error Condition Formula	Price__c < 0		
Error Message	Too cheap!	Error Location	Top of Page
Description			
Created By	Morris Katz, 7/11/2012 10:51 AM	Modified By	Morris Katz, 7/11/2012 10:51 AM
		Edit	Clone

Step 4: Test the Validation Rule

The Database.com architecture is such that you can make changes to your schema in real time and see those changes immediately. For example, the new validation rule is in effect, so if you try to insert or update a record in the Track object that doesn't comply with the new rule, you'll get an error. Let's try it out.

1. Return to the Workbench session that's connected to your test database.

2. Hover over the Id link for any data record, select **Update**, and set the Price__c field to a negative number.
3. Click **Confirm Update**.



Notice that the update is not successful this time—Database.com raises a validation rule exception (returns an error) because of our new validation rule. Perfect—that’s just the way we want it.

Step 5: Create a Roll-up Summary Field in a Master Object

Database.com provides a lot of functionality around objects in a master-detail relationship. For example, deleting the master record also deletes the detail records. This is known as a *delete cascade*.

Master-detail relationships let you create fields of a special roll-up summary type—these fields automatically aggregate information found in the detail records associated with each master record. In this step, add a roll-up summary field to Album that calculates the total album price based on the prices of individual tracks.

1. Return to your Database.com session.
2. Click **Create > Objects** and then click **Album**.
3. In the **Custom Fields & Relationships** section, click **New**.
4. For **Data Type**, select **Roll-Up Summary** and click **Next**.
5. For **Field Label**, type *Price*, press TAB, and click **Next**.
6. For **Summarized Object**, select **Tracks**.
7. For **Roll-Up Type**, select **Sum**.
8. For **Field to Aggregate**, select **Price**.
9. The remaining defaults are perfect, so just click **Next** and then **Save**.

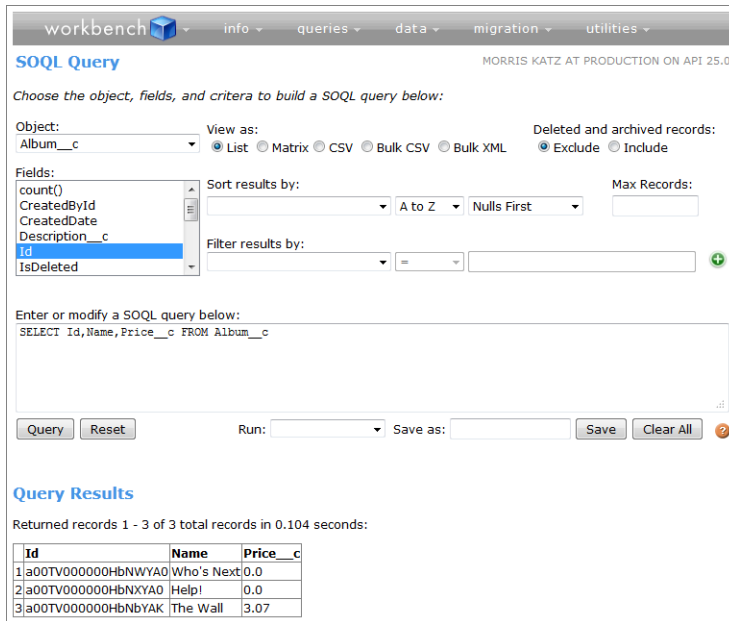
At this point, all of your Album records have an additional field, Price__c, which automatically calculates the total price of the album based on the sum of the prices for related tracks. As you saw during the creation of this field, you can use roll-up summary fields to count and find the minimum or maximum of particular fields.

To test that the new summary field is working properly, return to Workbench and build the following query.

```
SELECT Id, Name, Price__c FROM Album__c
```

If you don’t see **Price__c** in the list of Fields available for the query, clear the Workbench cache, and retry the query.

Your output shows the price value that is automatically calculated for each album.



workbench info queries data migration utilities

SOQL Query MORRIS KATZ AT PRODUCTION ON API 25.0

Choose the object, fields, and criteria to build a SOQL query below:

Object: Album__c View as: ☒ List ☐ Matrix ☐ CSV ☐ Bulk CSV ☐ Bulk XML Deleted and archived records: ☒ Exclude ☐ Include

Fields: count() CreatedById CreatedDate Description__c **Id** IsDeleted

Sort results by: A to Z Nulls First Max Records:

Filter results by:

Enter or modify a SOQL query below:
SELECT Id,Name,Price__c FROM Album__c

Query Reset Run: Save as: Save Clear All

Query Results

Returned records 1 - 3 of 3 total records in 0.104 seconds:

Id	Name	Price__c
1a00TV000000HbNwYAO	Who's Next	0.0
2a00TV000000HbNXXAO	Help!	0.0
3a00TV000000HbNbYAK	The Wall	3.07



Note: When you add a roll-up summary field to a master table, Database.com schedules a background job to calculate the summary data for all existing records. You might have to wait a few seconds to see results similar to those above.

Summary

In this tutorial, you discovered how to relate two objects, and how a master-detail relationship is reflected in SOQL queries. You then learned how to declare validation rules that help ensure objects contain records that adhere to your business rules. Finally, you saw how to take advantage of roll-up summary fields to automatically aggregate detail record information as fields in the master records.

Tutorial 4: Managing Users, Data Visibility, and Data Sharing

Up until now, you've been interacting with your database using a single user. The default user is an administrator, which has access to all records in the database. Database.com extends the traditional database models by offering users, permissions, data sharing, and other features that give you fine control over data visibility. You can easily hide complete objects, a field of an object, or just particular records from groups of users.

In this tutorial, you create a new user and security profile, and use this security profile to toggle field-level security and record sharing.

Step 1: Create a New User and Profile

One way to control data access for users is through their security profiles. In this step, create a new security profile for your vendors, add a new user to this profile, and discover how to toggle object visibility for the profile.

Begin by creating a new profile in your test database.

1. Using your Database.com Console, select **Manage Users > Profiles**.
2. Click **Clone** next to the **Database.com User** entry.

3. For **Profile Name**, enter `Vendor`, and then click **Save**.

The Database.com Console displays a page that lists the permissions that are available to users who have the `Vendor` profile. For example, you can use this to set permissible login hours, or the range of IP addresses from which Database.com permits sessions.

Profile
Vendor
[Back to List: Profiles](#) [Help for this Page](#)

Users with this profile have the permissions listed below. Administrators can change a user's profile by editing that user's personal information.

Profile Detail [Edit](#) [Clone](#) [Delete](#) [View Users](#)

Name	Vendor		
User License	Database.com User	Custom Profile	<input checked="" type="checkbox"/>
Description			
Created By	Morris Katz, 7/11/2012 11:32 AM	Modified By	Morris Katz, 7/11/2012 11:32 AM

Field-Level Security

Standard Field-Level Security
User [\[View\]](#)

Custom Field-Level Security
Album [\[View\]](#) Track [\[View\]](#)

Administrative Permissions

API Enabled	<input checked="" type="checkbox"/>	Edit Read Only Fields	<input type="checkbox"/>
API Only User	<input checked="" type="checkbox"/>	Invite Customers To Chatter	<input checked="" type="checkbox"/>
Chatter Internal User	<input checked="" type="checkbox"/>	Manage Public List Views	<input checked="" type="checkbox"/>
Create and Own New Chatter Groups	<input checked="" type="checkbox"/>	Send Outbound Messages	<input checked="" type="checkbox"/>

Custom Object Permissions

	Basic Access				Data Administration	
	Read	Create	Edit	Delete	View	Modify
Albums	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Tracks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

[Edit](#) [Clone](#) [Delete](#) [View Users](#)

For the new `Vendor` profile, note how the Custom Object Permissions don't have any check marks next to the Album and Track objects. This indicates that users with this profile do not have access to any records at all.

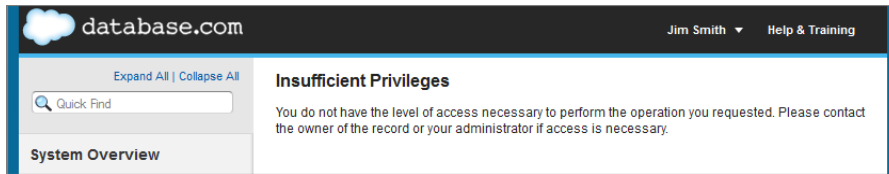
To test this, create a new user and assign him to this profile.

1. Select **Manage Users > Users**.
2. Click **New User**.
3. For **First Name**, type `Jim` and press **TAB**.
4. For **Last Name**, type `Smith`.
5. For **Email**, type *your* email address (so that you receive the activation email).
6. For **Username**, type something memorable in the form of an email address, related to your current user. Note that this doesn't need to be an authentic email address, just something in the form of an email address. For example, our administrator username is `admin-user@workbook.db`, so we can call this new user `jimsmith-user@workbook.db`.
7. For **User License**, select Database.com User (the default).
8. For **Profile**, select the new `Vendor` profile you created earlier.
9. Click **Save**.

You've now created a new user, and assigned the user to the `Vendor` profile.

Next, go to your own email account and look for the new user's activation email message. Click the link and activate the new user through the default landing page for the Database.com Console's set password page.

Once you set the initial password for a non-administrator, the Database.com Console redirects the session to the default landing page, which is the System Overview page. Because non-administrators don't have access to this page, you'll likely see an "Insufficient Privileges" message. Don't worry—simply log out as the non-administrator and continue to the next step.



Note: To support an application's user password management page in a production setting, make sure to configure an alternative home page for Database.com password management using **Password Policies > Alternative Home Page**.

Step 2: Toggle Object Visibility

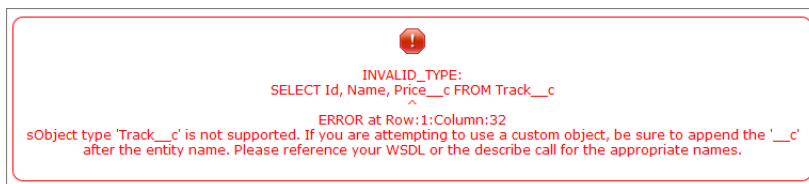
Now let's use Workbench, as the new user, to see what we can (and cannot) do.

1. Log out of Workbench with **workbench > Logout**.
2. Establish a new connection to the test database as your new user, starting with **workbench > Login**, and the login screen. Refer to [Tutorial 1, Step 2](#) on page 9 if you need to review the procedure.
3. Once you are connected, select **queries -> SOQL Query**.
4. In the Object list, notice that there are no Album__c and Track__c objects to choose. That's what we expect, because the user's Vendor profile doesn't provide its users with access to these custom objects.

Let's see if we can sneak in by hand-coding a query. Try executing the following query.

```
SELECT Id, Name, Price__c FROM Track__c
```

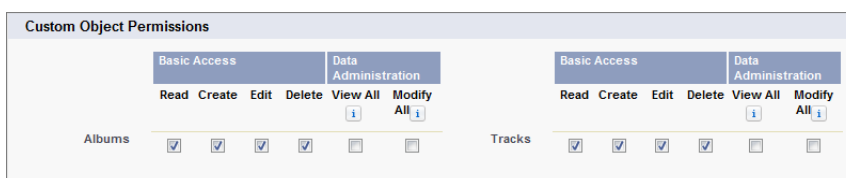
Workbench should return the following error.



In other words, because the new user has a profile that restricts object visibility, Database.com ensures that the object is not visible at all.

Switch back to the Database.com Console browser window. Because you logged out earlier (after setting the password for the new user), you'll have to log in again as the administrator.

1. Navigate to **Manage Users > Profiles**.
2. Click **Edit** next to the **Vendor** profile.
3. Check the boxes in the **Read, Create, Edit, and Delete** columns for both Album and Track, and click **Save**.



Any user with this profile will now be able to create, read, edit, and delete any record of the Album and Track object.

Now jump back over to your Workbench session (that's still tied to the Vendor profile) and simply re-execute the query. This time, it should successfully return all of the records in the Track object. If you still don't see all the records, clear the Workbench

cache and then retry the query. Profiles and object visibility are very powerful ways of granting access for entire objects to sets of users.

Step 3: Set Field Level Security

Database.com offers finer control than toggling object permissions. You can also specify field-level security, granting or denying access to sets of fields. In this step, you prevent vendors from accessing the price field on individual tracks.

1. If necessary, navigate to **Manage Users > Profiles**.
2. Click the **Vendor** profile.
3. In the **Field Level Security** section, click the **View** link next to **Track**.
4. Click **Edit**.
5. Deselect **Visible** for the Price field, and click **Save**.


Track Field-Level Security for profile **Vendor** [Help for this Page](#)

Save Cancel

Field Name	Field Type	Visible	Read-Only
Album	Lookup	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Created By	Lookup	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Last Modified By	Lookup	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Price	Currency	<input type="checkbox"/>	<input type="checkbox"/>
Track Name	Text	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Save Cancel

Now return to your Workbench session and re-execute the same query. This time, the application will throw an exception.



INVALID_FIELD:
SELECT Id, Name, Price__c FROM Track__c
^
ERROR at Row:1:Column:18
No such column 'Price__c' on entity 'Track__c'. If you are attempting to use a custom field, be sure to append the
'__c' after the custom field name. Please reference your WSDL or the describe call for the appropriate names.

If you reconnect as the administrator, which uses a more privileged profile, the query executes just fine. You can also remove Price__c from your query to make the query execute successfully.

Step 4: Specify Organization-Wide Default Sharing Rules

You've just toggled object visibility and field visibility. Database.com adds a third approach to data security: *record sharing*. Record sharing determines when a user can access records owned by another user. For example, you might want to allow vendors to only view the albums they own, but not view albums owned by other vendors.

Organization-wide (database-wide) sharing settings specify the default level of access to records, and can be set separately for different objects. To see how this works, let's do some work before and after making a configuration change.

As a baseline, switch to the Workbench session, which should still be connected as Jim Smith. Execute the following query:

```
SELECT Id, Name from Album__c
```

Query Results

Returned records 1 - 3 of 3 total records in 0.069 seconds:

Id	Name
1 a00T00000000HbNMIA0	Help!
2 a00T00000000HbNNIA0	The Wall
3 a00T00000000HbNIIA0	Who's Next

The result set includes all data records in the Album object, even though our vendor, Jim Smith, doesn't own any of them. That's because the default and current organization-wide sharing setting for the Album object is Public Read/Write.

We don't want vendors to see records that they don't own. If you change the organization-wide sharing setting for Album to Private, then all users (other than administrators) will only see the records they own. In our scenario, Jim Smith doesn't own any records, and so shouldn't have access to any. Change the setting to verify this.

1. In the Database.com Console, as the administrator, navigate to **Security Controls > Sharing Settings**.
2. Click **Edit**, and change the **Default Access** for Album to **Private**.
3. Click **Save**.

Organization-Wide Sharing Defaults Edit [Help for this Page](#)

Edit your organization-wide sharing defaults below. Changing these defaults will cause all sharing rules to be recalculated. This could require significant system resources and time depending on the amount of data in your organization.

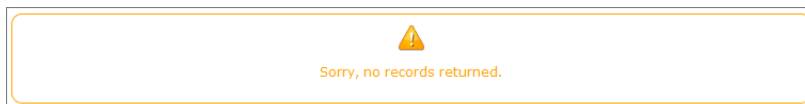
Object	Default Access	Grant Access Using Hierarchies
Album	Private	<input checked="" type="checkbox"/>

Save Cancel



Note: When you set a master object's organization-wide sharing default to Private, Database.com automatically cascades this default to all detail objects that reference the master object. This means that, by default, a user won't be able to work with detail records unless the user can also access the master record.

Now, go back to your Workbench session, still connected as Jim Smith, and re-execute the same query. Because Jim Smith doesn't own any records in the Album object, the query now doesn't return any records.



The records still exist, but the only user who can work with them is the person who owns them. To prove this last point, let's have Jim Smith create a new album and make sure that the new record is indeed visible to this user.

1. Back in your Workbench session, still connected as Jim Smith, click **data > Insert**.
2. For **Object type**, select **Album__c**, select **Single Record**, and click **Next**.
3. For the new album's fields, enter:
 - **Description__c:** Far out
 - **Name:** A Trick of the Tail
 - **Released_On__c:** 1976-02-20
4. Click **Confirm Insert**.

In the result, click the link for the new record to show its detail page.



The screenshot shows the 'workbench' interface with a navigation bar containing 'info', 'queries', 'data', 'migration', and 'utilities'. Below the navigation bar, the user is identified as 'JIM SMITH AT DBCOM.MUSICSTORE ON API 22.0'. The main content area displays a record for 'Album__c' with the title 'A Trick of the Tail'. Above the record table are buttons for 'Update', 'Delete', 'Undelete', 'Purge', and 'View in Salesforce'. The record table has two columns: 'Field' and 'Value'.

Field	Value
CreatedById	005T00000000rhufIAA
CreatedDate	2011-07-26T02:14:06.000Z
Description__c	Far out
Id	a00T00000000HbNRIAQ
IsDeleted	false
LastModifiedById	005T00000000rhufIAA
LastModifiedDate	2011-07-26T02:14:06.000Z
Name	A Trick of the Tail
OwnerId	005T00000000rhufIAA
Price__c	0.0
Released_On__c	1976-02-20
SystemModstamp	2011-07-26T02:14:07.000Z

And if you click the **OwnerId** link, you'll see that Jim Smith is the owner. Because in the current Workbench session you're connected as Jim Smith, you can see this new record even though the organization-wide record sharing default is Private.



Note: Users such as administrators, who use a profile with the **View All** or **Modify All** access rights for an object, can see all records in the object, despite the organization-wide record sharing setting of Private.

Organization-wide record sharing access is a powerful way to control data visibility at the record level. But Database.com sharing features are far more powerful than just this. For example, you can extend sharing by writing code (Apex Managed Shares) that explicitly grants control of certain records to groups of users under particular conditions.

Using Hierarchies

A refinement to organization-wide default record sharing is the ability to grant access to object records using a role hierarchy. In Database.com, you can create a hierarchical structure of roles, and assign users to these roles to give them a position in the hierarchy. Database.com then automatically opens access for a user to any records owned by other users lower down in the hierarchy.

For example, imagine a hierarchy in a company where a manager has access to records created by people that work for the manager. And above the manager, the CEO has access to all records that all managers have access to—all records! Role hierarchies let you easily manage data sharing policies like this efficiently with very little administrative overhead.

To set up a role hierarchy, click **Manage Users > Roles**. You can create a role hierarchy, add users, and run the queries in your application as each of the users to see that the record visibility holds.

Summary

Database.com offers powerful ways of controlling data visibility. In this tutorial, you created a user and security profile, and modified that profile to control object-level visibility. You then refined the visibility by controlling field-level visibility. Finally, you changed the organization-wide default sharing settings to ensure users only saw records that they own. Role hierarchies let you refine this still further, as do sharing rules, queues, and groups.

Tutorial 5: Using Workflow Logic

Database.com workflow rules are sets of actions that fire when conditions are met on a record. You can specify criteria based on formulas or field conditions, as well with a time-based mechanism. For actions, you can update a field or send an outbound message to a URL of your choosing.

In this tutorial, you create a workflow rule to send an email alert to your auditor whenever an expensive album record is created or modified. In the context of this tutorial, \$20 is expensive. You then add an additional action to the rule to send an outbound message as well.

Step 1: Create a Workflow Rule

Creating a workflow is purely declarative, and split into two parts. First create the rule, which defines the criteria under which it fires. Then define a set of actions that fire when the rule is met.

Using the Database.com Console connected to the test database as the administrator, create a rule that fires when someone creates an expensive album.

1. Navigate to **Create > Workflow > Workflow Rules**.
2. Click **Continue** if necessary, then click **New Rule**.
3. For **Select object**, choose **Album** and click **Next**.
4. For **Rule Name**, type **Expensive Album Alert**.
5. For **Evaluate the rule when a record is:**, choose **created, and every time it's edited**.
6. In the **Rule Criteria** section, set the **Field** to **Price**, the **Operator** to **greater than**, and enter **20** for the **Value**.

Field	Operator	Value	
Price	greater than	20	AND
--None--	--None--		AND
--None--	--None--		AND
--None--	--None--		AND
--None--	--None--		AND

7. Click **Save & Next**.

Although you've created the rule, it's not yet active, and it doesn't do anything when the criteria are met. So there's still some more work to do.

Step 2: Send an Outbound Message

In this step, configure an outbound message action for the workflow rule, which fires off a SOAP message to a configurable endpoint.

Testing this ordinarily requires creating a web service endpoint that can be invoked by the outbound message action. Instead, you're going to use an online service, RequestBin, that intercepts SOAP messages. RequestBin captures and logs the asynchronous requests made when actions happen.

Create your test endpoint first.

1. Navigate to requestb.in.
2. Click **Create a RequestBin**.
3. Record the resulting URL. For example, <http://requestb.in/srbqexsr>.

Back in the Database.com Console, configure the outbound message.

1. If necessary, return to the Edit Rule Expensive Album Alert page.
 - a. Navigate to **Create > Workflow > Workflow Rules**.
 - b. Click **Expensive Album Alert**.

- c. Click **Edit** in the **Workflow Actions** section.
2. In the **Immediate Workflow Actions** section, click **Add Workflow Action > New Outbound Message**.
3. For **Name**, enter `Album Price Outbound Alert` and press TAB.
4. For **Endpoint URL**, enter your RequestBin URL (for example, `http://requestb.in/srbqexsr`).
5. For **Selected Fields**, ensure that `Id`, `Name`, `Price__c`, and `SystemModStamp` are selected. These selections identify the field data that Database.com sends to the endpoint whenever it fires the rule.

New Outbound Message [Help for this Page](#)

Step 2: Configure Outbound Message **Step 2 of 2**

[Previous](#) [Save](#) [Cancel](#)

Enter the details of your outbound message and select the fields you want included in this message. Note that the fields available depend on the type of record previously selected.

Edit Outbound Message: Album ! = Required Information

Name

Unique Name [i](#)

Description

Endpoint URL

User to send as [i](#)

Send Session ID ☐

Album fields to send

Available Fields

- CreatedById
- CreateDate
- Description__c
- IsDeleted
- LastModifiedById
- LastModifiedDate
- OwnerId
- Released_On__c

Selected Fields

- Id
- Name
- Price__c
- SystemModStamp

[Add](#) [Remove](#)

6. Click **Save**, and **Done**.
7. Click **Activate** to make the new work flow rule active.

Workflow Rule [Help for this Page](#)

Expensive Album Alert

[Back to List: Workflow Rules](#)

Workflow Rule Detail [Edit](#) [Clone](#) [Deactivate](#)

Rule Name	Expensive Album Alert	Object	Album
Active	<input checked="" type="checkbox"/>	Evaluation Criteria	Every time a record is created or edited
Description			
Rule Criteria	Album: Price GREATER THAN 20		
Created By	Morris Katz, 7/11/2012 12:48 PM	Modified By	Morris Katz, 7/11/2012 2:10 PM

Workflow Actions [Edit](#)

Immediate Workflow Actions

Type	Description
Outbound Message	Album Price Outbound Alert

You can't test this rule yet though!

Step 3: Set Up Network Security

Database.com is locked down by default, and doesn't allow public access either into or out of your Database.com instance. If you're going to send data to an external Web address, a system administrator has to explicitly change the network permissions to enable such actions.

You need to explicitly allow communication to the RequestBin website.

1. Navigate to **Security Controls > Remote Site Settings**.
2. Click **New Remote Site**.
3. For **Remote Site Name**, type RequestBin.
4. For **Remote Site URL**, type `http://www.requestbin.in/`.
5. Check **Disable Protocol Security** (because RequestBin does not provide a secure service).
6. Click **Save**.

Remote Site Details [Help for this Page](#)

[Back to List: Remote Site Settings](#)

Remote Site Detail [Edit](#) [Delete](#) [Clone](#)

Remote Site Name	RequestBin	Modified By	Morris Katz, 7/11/2012 2:29 PM
Remote Site URL	http://www.requestbin.in		
Disable Protocol Security	<input checked="" type="checkbox"/>		
Description			
Active	<input checked="" type="checkbox"/>		
Created By	Morris Katz, 7/11/2012 2:29 PM		

[Edit](#) [Delete](#) [Clone](#)

Your Database.com instance will now be able to communicate with the RequestBin site.

Step 4: Test the Outbound Messaging

You're now ready to test the new workflow rule that triggers an outbound message using Workbench. Establish a new session connecting to the test database as your administrator account (vendors can't work with prices for tracks). Once you are active, insert a new track with a price greater than \$20, making the album price greater than \$20, triggering the work flow rule criteria and initiating an outbound message.

1. Click **data > Insert**.
2. For **Object Type**, select **Track__c**, **Single Record**, and click **Next**.
3. For **Album__c**, type **A Trick of the Tail**, and select **Album__c.Name** in **Smart Lookup**.
4. For **Name**, enter **01 - Dance on a Volcano**.
5. For **Price**, enter **22**.
6. Click **Confirm Insert**.

The record is inserted and fires the Expensive Album Alert workflow rule.

In another browser, visit the RequestBin URL you created in Step 2. This time, add a parameter, `?inspect`, to the end of the URL. For example:

```
http://requestbin.in/srbqexsr?inspect
```

You'll see the SOAP data, including the fields you specified.

RequestBin BETA <http://requestbin.in/14x18g41> Home Download API Source Discuss Donate

#1h6zx2 POST /14x18g41 Headers Content

2012-07-11 22:11:40.872385 10.232.12.16, 96.43.144.8

body <?xml version="1.0" encoding="UTF-8"?> <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <soapenv:Body> <notifications xmlns="http://soap.sforce.com/2005/09/outbound"> <OrganizationId>00DE0000000aJrdMAE</OrganizationId> <ActionId>04kE00000008PDolAM</ActionId> <SessionId xsi:nil="true"?> <EnterpriseUrl>https://na9-api.salesforce.com/services/Soap/c/25.0/00DE0000000aJrd</EnterpriseUrl> <PartnerUrl>https://na9-api.salesforce.com/services/Soap/u/25.0/00DE0000000aJrd</PartnerUrl> <Notification> <Id>04IE0000000SqmIAS</Id> <sObject xsi:type="sf:Album__c" xmlns:sf="urn:subject.enterprise.soap.sforce.com"> <sf.Id>a00E0000002q4IsIAA</sf.Id> <sf.Name>A Trick of the Tail</sf.Name> <sf.Price__c>22.0</sf.Price__c> <sf.SystemModstamp>2012-07-11T22:11:40.000Z</sf.SystemModstamp> </sObject> </Notification> </notifications> </soapenv:Body> </soapenv:Envelope>

text/xml; charset=utf-8 1039 bytes

This bin will keep the last 20 requests made to it and remain available for the next 48 hours. However, during beta the data store might be cleared at any time, so **treat bins as highly ephemeral**.

© Jeff Lindsay 2011

When you use asynchronous background processing operations with Database.com, it's a good idea to regularly monitor their status. You can monitor the status of Database.com outbound messages (and many other asynchronous operations) using the Database.com Console as follows.

1. Navigate to **Monitoring > Outbound Messages**.
2. Review items scheduled for delivery and failures.

Outbound Messaging Delivery Status [Help for this Page](#)

Total items in queue awaiting delivery : 1

Next items for delivery Refresh

Action	Outbound Message ID	Object	# Attempts	Created Date	Next Attempt	Delivery Failure Reason
Retry Del	04KT0000000CaUL	a00T0000000HbNR	1	7/25/2011 8:47 PM	7/25/2011 8:47 PM	(403)Forbidden

Oldest failures in queue Refresh

Action	Outbound Message ID	Object	# Attempts	Created Date	Next Attempt	Delivery Failure Reason
Retry Del	04KT0000000CaUL	a00T0000000HbNR	1	7/25/2011 8:47 PM	7/25/2011 8:47 PM	(403)Forbidden

Because RequestBin does not respond to our outbound message (authentic SOAP servers would do so), Database.com thinks there is an error. That's why you'll see one or more failures and an outbound message remaining for delivery. To end the test, simply click the **Del** link in the **Next items for delivery** section. You might want to disable the workflow rule as well.

If you were implementing this in a production environment, you'd want to use the WSDL that's automatically generated by Database.com to generate the web service endpoint. Access the WSDL as follows:

1. Navigate to **Create > Workflow > Outbound Messages**.
2. Click **Album Price Outbound Alert**.
3. In the **Endpoint WSDL** field, click **Click for WSDL**.

Summary

Workflow is a powerful, declarative, approach to executing actions when records meet specified conditions. In this tutorial you created a workflow rule that triggers an outbound SOAP message action in response to a field update.

Tutorial 6: Creating and Deploying Change Sets

In the next tutorial we'll be developing some code, which needs to be done in a test database. To prepare for this development effort, we'll need to replicate some of the objects that we have in our test database to our production database. To do this, we'll use a *change set*. A change set is the feature of Database.com that you employ to move database *metadata* changes (object definitions and modifications, Apex classes and code, profiles, and so on) from one database to another. Change sets do not contain data records.

When you want to send changes from your source database or organization to a destination organization, you create an outbound change set. Once you send the change set, the receiving organization sees it as an inbound change set.

Before you can send a change set between two organizations, you need to create a deployment connection. Currently, you can only send change sets between related organizations—for example, a production organization and a test database, or two test databases created from the same organization.

In this tutorial, you learn how to set up a deployment connection, create an outbound change set in your test organization, and validate and deploy it to your production database.

Step 1: Connect to the Production Database

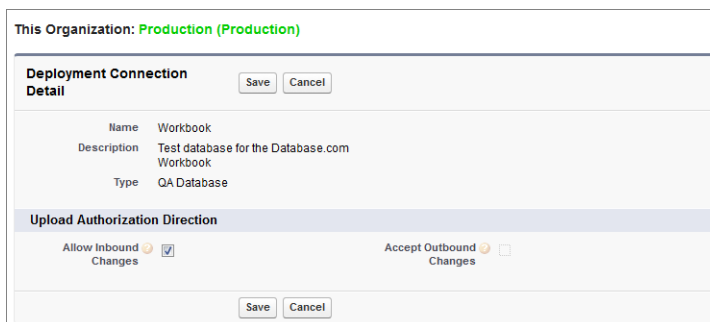
Use the Database.com Console to connect to the production database.

1. Open a new browser window and load the URL login.database.com.
2. Enter the production database username and password, and log in.

Step 2: Configure a Deployment Connection

In the Database.com Console session that's using your production database, navigate to **Deploy > Deployment Connections** (also click **Continue**, if necessary). You'll see that Database.com already created a deployment connection to your test database in the production database. However, the deployment connection does not allow inbound changes by default. Now that you are ready to send a change set to the production database, complete the following steps to accept inbound changes from the test organization.

1. Click **Edit** next to the **Test database** deployment connection.
2. Select **Allow Inbound Changes**.
3. Click **Save**.



The screenshot shows the 'Deployment Connection Detail' configuration page for a connection named 'Workbook'. The page is titled 'This Organization: Production (Production)'. It includes a 'Save' and 'Cancel' button at the top right. The configuration details are as follows:

Deployment Connection Detail	
Name	Workbook
Description	Test database for the Database.com Workbook
Type	QA Database

Below the details is the 'Upload Authorization Direction' section, which contains two checkboxes:

- Allow Inbound Changes**: ☒ (indicated by a green checkmark icon)
- Accept Outbound Changes**: ☐ (indicated by an orange circle icon)

At the bottom of the section are 'Save' and 'Cancel' buttons.

Step 3: Create and Upload an Outbound Change Set

In this step, create an outbound change set in the test organization and then upload it to the production database.

1. In a browser, log in to your test database at test.database.com as the administrator.
2. Navigate to **Deploy > Outbound Change Sets**.
3. Click **Continue**, if necessary, to bypass the description page.
4. Click **New**.
5. For **Name**, type `Sprint 1`.
6. Click **Save**.
7. Use the **Add** button next to **Change Set Components** repeatedly to add various metadata to your change set.

If you want, you can add everything you've created in the test database so far to the change set. However, you'll only need to iterate through Custom Object and Custom Field to build the change set necessary to support the following tutorial. Make sure your change set has the custom objects (Album, Track) and all custom fields for these objects, as shown in the following screen.

The screenshot shows the 'Change Set Detail' page for 'Sprint 1'. It includes buttons for Edit, Delete, Upload, and Clone. The 'Change Set Components' section shows a table of components to be added to the change set.

Action	Name	Parent Object	Type	API Name
Remove	Album		Custom Object	Album
Remove	Album	Track	Custom Field	Album
Remove	Description	Album	Custom Field	Description
Remove	Price	Album	Custom Field	Price
Remove	Price	Track	Custom Field	Price
Remove	Released_On	Album	Custom Field	Released_On
Remove	Track		Custom Object	Track

Previous (1 - 7 of 7) Next

Once you're sure you have everything in the change set, complete the following steps to send the change set to the production database.

1. In **Change Set Detail**, click **Upload**.
2. In **Upload Details**, select **Production**, the name of the production database.

The screenshot shows the 'Upload Details' page. It includes buttons for Upload and Cancel. The 'Target Organization' section shows a table with one row: 'Production'.

Name	Description	Type	Platform Version
Production	Production organization	Production	26.0

3. Click **Upload**.

When the upload completes, you'll see a confirmation page.


Change Set Video Tutorial | Help for this Page


Sprint 1

[Back to List](#) [Outbound Change Sets](#)

A change set contains customizations to components such as apps, objects, reports or email templates. You can use change sets to move customizations from one organization to another.

Once a change set has been uploaded, you can't add or remove components. However, you can clone the change set and then modify the component list in the new change set.

 **Your change set was uploaded successfully.**
It will be available shortly, so that an administrator can deploy it.

Change Set Detail			
Change Set Name	Sprint 1	Status	 Closed
Description			
Created By	Morris Katz, 7/13/2012 10:15 AM	Modified By	Morris Katz, 7/13/2012 10:23 AM

[Delete](#) [Upload](#) [Clone](#)

Change Set Upload History

Uploaded on	Target Organization	Uploaded by	Status
7/13/2012 10:23 AM	Test Dbcom Workbook	Morris Katz	Uploaded

Step 4: Validate and Deploy Inbound Change Set


Soon after you send a change set to your production database, you'll receive an email stating that the upload is complete. Once this happens, switch back over to your Database.com Console session that's connected to the production database. The next steps are to validate and deploy the inbound change set received from the test organization.

1. Navigate to **Deploy > Inbound Change Sets**.
2. Click **Continue** if necessary.
3. Click the link for the **Sprint 1** change set. There may be a delay before the change set becomes available.
4. Click **Validate** and **OK** to validate that when you deploy the change set, no errors arise.

Change Set Detail			
Change Set Name	Sprint 1	Status	Waiting for Deployment
Description	Expiration Date 9/15/2012		
Source Information			
Source Deployment Connection	Workbook	Uploaded By	Morris Katz @ Workbook, 7/18/2012 3:32 PM

[Validate](#) [Deploy](#) [Delete](#)

Deployment History

Action	Start Time	End Time	Validate Only	Status	Components	Apex Tests	Deployed By
View Results	7/18/2012 3:34 PM	7/18/2012 3:34 PM		Succeeded	7		Morris Katz

5. Upon successful validation, click **Deploy**, and then **OK** to deploy the change set.

Change Set Detail [Validate] [Deploy] [Delete]

Change Set Name	Sprint 1	Status	Deployed
Description	Expiration Date 1/13/2013		

Source Information

Source Deployment Connection	Workbook	Uploaded By	Morris Katz @ Workbook, 7/18/2012 3:32 PM
------------------------------	----------	-------------	---

[Validate] [Deploy] [Delete]

Deployment History

Action	Start Time	End Time	Validate Only	Status	Components	Apex Tests	Deployed By
View Results	7/18/2012 3:39 PM	7/18/2012 3:39 PM	<input type="checkbox"/>	Succeeded	7		Morris Katz
View Results	7/18/2012 3:34 PM	7/18/2012 3:34 PM	<input checked="" type="checkbox"/>	Succeeded	7		Morris Katz

And that's it! You've just migrated the custom objects and fields from your test database to the production organization. Now you're ready to develop some code in the next tutorial.



Note: Change sets never include users in the source database.

Step 5: Disallow Inbound Changes

Once you deploy an inbound change set to a database, it's a good idea to lock down the database so that you have absolute control over future change sets arriving. You can do this very simply using the Database.com Console:

1. Navigate to **Deploy > Deployment Connections**.
2. Click **Edit** next to the **Test** database deployment connection.
3. Clear **Allow Inbound Changes**.
4. Click **Save**.

Summary

In this tutorial, you learned about Database.com deployment connections and change sets, and how you use them to move changes from one database to another.

Tutorial 7: Using Custom Business Logic and Database Triggers

Database.com workflow rules can centralize many common types of business logic that an organization's applications need to accommodate, but there will always be custom workflows and business logic that declarative workflows cannot anticipate.

You can implement such logic in your application, but then you have to inefficiently repeat this process for every application that the database supports. A better way to address these cases with Database.com is to create centralized database triggers that enforce your custom business logic.

Database.com provides you with a strongly typed, object-oriented programming language, Apex, to implement such database-side logic requirements. The language will be familiar to anyone who knows Java. Apex supports:

- Data manipulation language (DML) calls, such as INSERT, UPDATE, and DELETE.
- Embedded Salesforce Object Query Language (SOQL) and Salesforce Object Search Language (SOSL).

- Looping constructs that provide for bulk processing of record sets.
- A locking syntax that prevents record update conflicts.

In this tutorial, you create a simple database trigger to change some default behavior of the master-detail relationship between the Album and Track objects. As it stands now, when someone deletes an Album record, Database.com also deletes related tracks. Let's say that, for our business, it's OK to delete albums that don't have any tracks, but we want to prevent the deletion of an album that does have tracks. We enforce this rule with a database trigger.



Note: In traditional relational database parlance, Database.com enforces the delete cascade referential action for every master-detail relationship. In this tutorial, what we are doing is implementing the delete restrict action with a trigger.

As a bonus, you learn about another great Database.com feature in this tutorial: the Recycle Bin. You'll see how to locate and recover deleted records from the Recycle Bin.

You can download the code for this tutorial at <https://gist.github.com/1132445> and <https://gist.github.com/1132449>.

Step 1: Create a Trigger

Use the Database.com Console to create a trigger. Connect to your test database as the administrator, and complete the following steps.

1. Navigate to **Create > Objects**.
2. Click **Album** on the Custom Objects page.
3. Scroll down, if necessary, to display the Triggers section and click **New**.

Notice that the Database.com Console provides you with an interface for declaring your trigger. The basic syntax for a trigger is as follows:

```
trigger name on object (event) {
  body
}
```

where:

- *name* is a unique name for the trigger.
- *object* is the object associated with the trigger.
- *event* is the event that determines when the trigger fires. Valid events include any combination of before/after and insert/update/delete/merge/upsert/undelete.
- *body* is the Apex code that executes when the trigger fires. Use this to implement your business logic.



Note: It's important to realize that a Database.com trigger can fire in the context of a single-row transaction or a bulk processing transaction. Consequently, you need to design all triggers with bulk processing in mind.

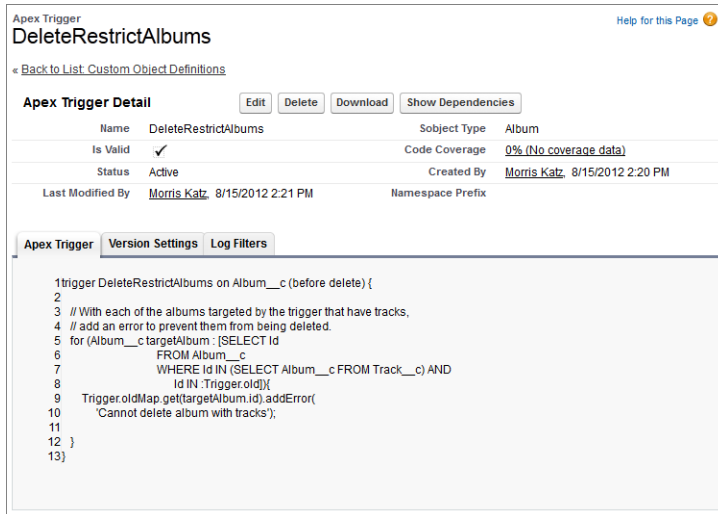
1. In your browser, navigate to <https://gist.github.com/1132445>.
2. Copy the code from the code window on this page into the Apex trigger window.

Let's do a code walk-through so that you understand exactly what the trigger does and learn some Apex.

- The name of the trigger is `DeleteRestrictAlbums`.
- The trigger is associated with the `Album__c` object.
- The trigger fires before the execution of a triggering statement that deletes one or more records from the `Album__c` object.

- The trigger body consists of a SOQL for loop that processes a set of albums. The compound `WHERE` clause of the loop's defining query limits the return set to albums with tracks that the triggering statement is targeting for deletion (using `Trigger.old`).
- For each record in the result set, the body of the for loop adds an error to the corresponding album record, thus preventing Database.com from deleting these albums.

Click **Quick Save** to check your syntax before leaving the page. Once there are no errors, make sure to enable **Is Active** and then click **Save**.



Apex Trigger Detail

Name	DeleteRestrictAlbums	Subject Type	Album
Is Valid	✓	Code Coverage	0% (No coverage data)
Status	Active	Created By	Morris Katz 8/15/2012 2:20 PM
Last Modified By	Morris Katz 8/15/2012 2:21 PM	Namespace Prefix	

```

1 trigger DeleteRestrictAlbums on Album__c (before delete) {
2
3   // With each of the albums targeted by the trigger that have tracks,
4   // add an error to prevent them from being deleted.
5   for (Album__c targetAlbum : [SELECT Id
6     FROM Album__c
7     WHERE Id IN (SELECT Album__c FROM Track__c) AND
8     Id IN :Trigger.old]) {
9     Trigger.oldMap.get(targetAlbum.Id).addError(
10      'Cannot delete album with tracks');
11   }
12 }
13

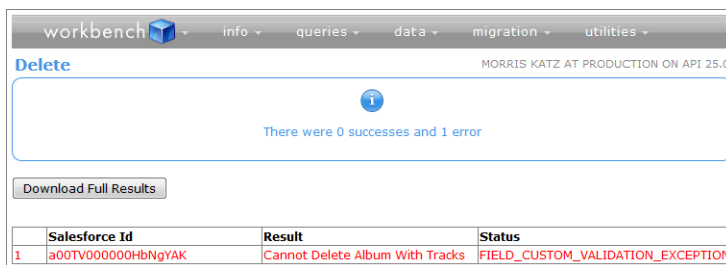
```



Note: As with declarative workflow rules, you can deactivate and activate triggers to meet your situational requirements.

Step 2: Logically Test the Trigger

To test that the trigger does what we intend, log in to Workbench for your test database and insert some albums and tracks. Then try to delete an album that has at least one track. When you do, you should see an error such as the following.



workbench info queries data migration utilities

Delete MORRIS KATZ AT PRODUCTION ON API 25.0

There were 0 successes and 1 error

Download Full Results

	Salesforce Id	Result	Status
1	a00TV000000HbNgYAK	Cannot Delete Album With Tracks	FIELD_CUSTOM_VALIDATION_EXCEPTION

Next, try to delete an album that does **not** have any tracks to make sure that the trigger does not prevent the deletion of such albums. Don't worry, we'll recover the deleted album later in this tutorial.

Step 3: Programmatically Test the Trigger

Apex provides built-in support for unit test creation and execution, including test results that indicate how much code is covered. Before you can add Apex classes and database triggers in your production database, you must create unit tests that programmatically validate at least 75% of the code in your organization.

This testing helps verify that your code executes as you expect it to, and that it doesn't consume unnecessary or extraordinary amounts of system resource. As a side effect, it also helps ensure the integrity of Database.com releases.

Unit test methods take no arguments and commit no data to the database. To create unit tests for the `DeleteRestrictAlbums` trigger, complete the following steps.

1. Navigate to **Develop > Apex Classes**.
2. Click **New**.
3. In a separate browser window, navigate to <https://gist.github.com/1132449>.
4. Copy the code from the code window on this page into the **Apex Class** field.

The comments in the code explain the gist of the test methods. Notice that it's important, when building and testing triggers, to keep in mind that triggers can fire as the result of both single-row and bulk triggering statements.

Here are a few important points to understand about building unit tests.

- Use the `@isTest` annotation to define classes or individual methods that only contain code used for testing. Classes annotated with `@isTest` don't count against your Apex limit.
- Test classes must be top-level classes.
- Unit test methods are static methods that are defined with the `@isTest` annotation or the `testMethod` keyword.

After you enter the `TestDeleteRestrictAlbums` test class, test, save, and run it.

1. Click **Quick Save** to test the code for compilation errors.
2. Once the class compiles without errors, click **Save**.
3. Click **Run Test** on the **TestDeleteRestrictAlbums** class page to run the test class and see what happens.

Apex Test Result Help for this Page	
Summary	
Test Class	TestDeleteRestrictAlbums
Tests Run	3
Test Failures	0
Code Coverage Total %	100
Total Time (ms)	26824.0
Debug Log	Download
Test Successes	
Method Name	Total Time (ms)
TestDeleteRestrictAlbums.verifyAlbumNoTracksDelete	23763.0
TestDeleteRestrictAlbums.verifyBulkAlbumDeleteRestrict	1317.0
TestDeleteRestrictAlbums.verifyAlbumTracksRestrict	1744.0
Code Coverage	
▼ Trigger Code Coverage	
Trigger Name	Coverage %
DeleteRestrictAlbums	100

When you run a test class, Database.com executes all of the unit test methods in the class and returns a report for the test run.

Step 4: View Recycle Bin Records and Restore Them

When someone deletes a record from an object, Database.com “soft deletes” the record by setting its `IsDeleted` field to true. Effectively, such records exist in what is called the *Recycle Bin*. The Recycle Bin keeps your database's deleted records for 15 days before permanently deleting them. So, if you change your mind, you have a way to gracefully restore recently deleted records.

In this step, use Workbench to view and restore the record that you deleted in Step 2 of this tutorial. Using your Workbench session that's connected to the test database as the administrator, execute the following query after making sure **Deleted and archived records** is set to the default **Exclude** option.

```
SELECT Id, IsDeleted, Name FROM Album__c
```

This first result set contains three records, all with `IsDeleted` equal to false. Now re-execute the query after setting **Deleted and archived records** to the **Include** option. The new result set includes an extra record, a record with `IsDeleted` equal to true. This is the record you deleted in Step 2—a record that's in the Recycle Bin.

workbench info queries data migration utilities

SOQL Query MORRIS KATZ AT PRODUCTION ON API 25.0

Choose the object, fields, and criteria to build a SOQL query below:

Object: Track__c View as: ☒ List ☐ Matrix ☐ CSV ☐ Bulk CSV ☐ Bulk XML Deleted and archived records: ☐ Exclude ☒ Include

Fields: count() Album__c CreatedById CreatedDate Id IsDeleted

Sort results by: [] A to Z Nulls First Max Records: []

Filter results by: [] = []

Enter or modify a SOQL query below:

```
SELECT Id, IsDeleted, Name FROM Album__c
```

Query Reset Run: [] Save as: [] Save Clear All

Query Results

Returned records 1 - 4 of 4 total records in 0.093 seconds:

Id	IsDeleted	Name
1a00TV000000HbNgYAK	false	The Wall
2a00TV000000HbNwYAO	false	Who's Next
3a00TV000000HbNXYAO	false	Help!
4a00TV000000HbNbYAK	true	The Wall

You have a few options for working with the Recycle Bin.

- Wait 15 days for Database.com to permanently delete the record.
- Use Workbench to manually restore or undelete the record.
- Use Workbench to manually purge or permanently delete the record.

Let's learn how to restore the record.

1. Hover over the link for the deleted record.
2. Click **Undelete**.
3. Click **Confirm Undelete**.

If you re-execute the previous query, you can confirm that Database.com restored the record and updated the record's `IsDeleted` field to false.



Note: The Database.com APIs provide complete access to Recycle Bin functionality, including bulk processing of Recycle Bin records.

Step 5: Move the Trigger and Unit Tests to Production

As an optional exercise, use what you learned in the previous tutorial about change sets to move your trigger and unit test from the test database to production. Here's a list of things you'll need to accomplish.

1. In the production organization, allow inbound changes for the test database deployment connection.
2. In the test database, create a new outbound change set with the trigger and Apex unit test class.
3. In the test database, upload the new outbound change set to the production organization.
4. In the production organization, validate and deploy the inbound change set received from the test database.

Summary

Database.com triggers provide a way for you to centralize logic that should be executed under various conditions associated with a record. In this tutorial, you learned how to write triggers in Apex, the object-oriented programming language supported by Database.com. You also discovered how to write test classes for Apex code, as well as how to use the Recycle Bin.

Part 2: External Application Integration

In this part of the Database.com Workbook, you learn how to access your database from an external client application that uses the Database.com REST API. Database.com exposes many APIs, and the primary API used to access your data is the Database.com REST API. There is also a Metadata API, which is used to access the configuration of your database, and a Chatter REST API, which is used to access the Chatter feeds. In this part, we'll focus exclusively on the Database.com REST API.

Database.com uses OAuth 2.0, an open standard for authorization. OAuth lets you construct applications that delegate the actual authentication to the Database.com platform. In the following tutorial, we'll use pre-built OAuth libraries to do the heavy lifting for us.

The following tutorial shows you how to integrate using Java. You can just as well use other languages such as C#, PHP, or Ruby. If you're not a Java programmer, look through the Java tutorial and you'll see that the same principles apply in your own language—interacting with a REST API is just issuing HTTP actions to the correct URLs. As a result, the principles are the same in any language. For additional integration samples using a different schema, see the [Integration Workbook](#).

You can also create mobile applications that access Database.com. See the [Mobile SDK Workbook](#) for tutorials that demonstrate Database.com connections on iOS and Android platforms.

Tutorial: Creating a Java Web Application to Access the Database

In this tutorial, you create and configure a Java Web application. You won't have to code it, though—instead, you'll create it using a Maven template, which ensures all library dependencies are installed. The final step in this tutorial performs a code walk-through so that you can see how it works under the hood. Feel free to make changes to the code and experiment.

Prerequisites

1. You need to have Maven installed. You can install the latest version from maven.apache.org.
2. You need to have completed Tutorials 1-3 of [Part 1](#) on page 3.

Step 1: Configure OAuth on Database.com

You are going to use a local web application, your client, to invoke OAuth 2.0 for authenticating users against your Database.com instance. With OAuth 2.0, a client application delegates the authentication to a provider, in this case Database.com, which in turn issues an access token if the user successfully authenticates.

As a result, your application doesn't need to handle authentication. It simply needs to ensure that a valid access token accompanies all interactions with the API.

To configure OAuth on Database.com, log into the production database as an administrator and configure a remote access application.

1. Navigate to **Develop > Remote Access**.
2. Click **New** on the Remote Access page.
3. For **Application**, enter `Test Client`.
4. For **Contact Email**, enter your own email address.
5. For **Callback URL**, enter `http://localhost:5000/_auth`.

6. Click **Save**.

Remote Access [Help for this Page](#)

[Back to List: Remote Access](#)

Remote Access Detail [Edit](#) [Delete](#)

Basic Information ! = Required Information

Application: Test Client

Description:

Logo Image URL:

Info URL:

Contact Phone:

Contact Email: mkatz@dealmaker.com

Integration ! = Required Information

Callback URL: http://localhost:5000/_auth

Policies ! = Required Information

No user approval required for users in this organization ☐ [i](#)

Authentication ! = Required Information

Use digital signatures

Consumer Key: 3MVG9PhR6g6B7ps6vmOTZBloN7KJzmX1toRM.dlymaG0JIB8QmIKH2w_26RD0K5nCLXWur6wkcZJIE7JKe0V

Consumer Secret: [Click to reveal](#)

Created Date: 7/13/2012 1:10 PM

Created By: [Morris Katz](#)

The detail page for your remote access configuration displays a consumer key as well as a consumer secret (which you have to click to reveal). You'll need these in the next step.

The callback URL is particularly important—as part of the OAuth 2.0 authentication dance, this URL will be invoked upon successful completion. In the next step, you'll create an application that does all the OAuth work, including implementing this URL.

Step 2: Download and Compile the Java Application

You create an application by executing a Maven template. This does all the hard work for you, and produces a working Java application. To retrieve the application:

1. Go to www.database.com/workbook and download the Java template associated with this workbook.
2. Unzip the template file and create a folder named `Maven-Workbook-Template`.
3. Using a command prompt:
 - a. Change the working directory to `Maven-Workbook-Template`. For example, `cd Maven-Workbook-Template`.
 - b. Run the command `mvn install`.

Maven will download all of the dependencies of the application, compile it, and place all the run-time assets in the target directory.



Note: Recent versions of Maven install resource files under the current user's system directory (for example, `C:\Users\<username>\.m2\repository` on Windows; `/Users/<username>/\.m2/repository` on Mac OS.) This placement can prevent your application from being able to find essential files. To fix the problem, you can do either of two things:

- Copy the files in the `repository` folder to the `Maven-Workbook-Template\repo` folder. You will probably need to create the `repo` folder.

OR

- In the command prompt window where you test your application, set a `REPO` environment variable to the repository path:

```
set REPO=c:\Users\<username>\.m2\repository
```

Step 3: Configure and Run the Java Application

There are many implementations of the OAuth 2.0 standard in most current programming languages. Salesforce.com provides one as part of the Java SDK, which your Java application uses. It's a Java servlet filter that takes care of the hard authentication parts, and leaves you with the results of authentication—an access token and an instance URL.

Look at the `src/main/webapp/WEB-INF/web.xml` file and you will see the OAuth filter in operation.

```
<filter>
  <filter-name>AuthFilter</filter-name>
  <filter-class>com.force.sdk.oauth.AuthFilter</filter-class>
  <init-param>
    <param-name>connectionName</param-name>
    <param-value>FORCEDATABASE</param-value>
  </init-param>
</filter>
```

A servlet filter named `AuthFilter` is created. This has a `connectionName` parameter, the value of which needs to be defined in an environment variable. This variable ultimately references the login end point the application uses, and your OAuth credentials. Because the parameter has a value of `FORCEDATABASE`, you have to create an environment variable called `FORCE_FORCEDATABASE_URL` that holds these credentials.

The format of the `FORCE_FORCEDATABASE_URL` environment variable is as follows:

```
force://<LOGIN_ENDPOINT>;oauth_key=<OAUTH_KEY>;oauth_secret=<OAUTH_SECRET>
```

Because we're logging into the production database, `<LOGIN_ENDPOINT>` will be `login.database.com`. The OAuth key and secret must be set to the values of the consumer key and consumer secret in the remote application in [Step 1](#) on page 37. As a result, your final environment variable value will look something like this:

```
force://login.database.com;oauth_key=
3MVG9CVKiXR7Ri5qcjfWDZa0rvFSordUA5LfVJe0GIZ4mwUvFzM6WIPEiYCNcYS.gzFtFEjEWrfMFvEku1.FY;
oauth_secret=341524404139967490
```

You're almost there! From the same command prompt, create an environment variable with this same value. In Unix-like operating systems, you can execute something like this in the command line:

```
export FORCE_FORCEDATABASE_URL="force://login.database.com;oauth_key=
3MVG9CVKiXR7Ri5qcjfWDZa0rvFSordUA5LfVJe0GIZ4mwUvFzM6WIPEiYCNcYS.gzFtFEjEWrfMFvEku2.FY;
oauth_secret=391524404139967490"
```

On Microsoft Windows operating systems:

```
set FORCE_FORCEDATABASE_URL=force://login.database.com;oauth_key=
3MVG9CVKiXR7Ri5qcjfWDZa0rvFSordUA5LfVJe0GIZ4mwUvFzM6WIPEiYCNcYS.gzFtFEjEWrfMFvEku2.FY;
oauth_secret=391524404139967490
```

The final bit of configuration needed is to set the port number. We want the Web server to run on the same port you configured in the remote application Callback URL, which is 5000. From the command prompt, execute the following line (on Unix):

```
export PORT=5000
```

For Microsoft Windows operating systems:

```
set PORT=5000
```

Step 4: Run the Application

Now that you've configured the application, you can run it. The Maven command you executed earlier created two executable shells: one for Unix and one for Microsoft Windows. On Unix, run:

```
sh target/bin/webapp
```

On Windows, run:

```
target/bin/webapp.bat
```

The Java application now runs. The OAuth servlet filter picks up the environment variables you set for the OAuth credentials, as well as the port number. If all goes well, you'll see something like this in your log window:

```
INFO : org.springframework.web.servlet.DispatcherServlet - FrameworkServlet 'appServlet': initialization completed in 549 ms
INFO : org.eclipse.jetty.util.log - Started SelectChannelConnector@0.0.0.0:5000 STARTING
```

Step 5: Test the Application

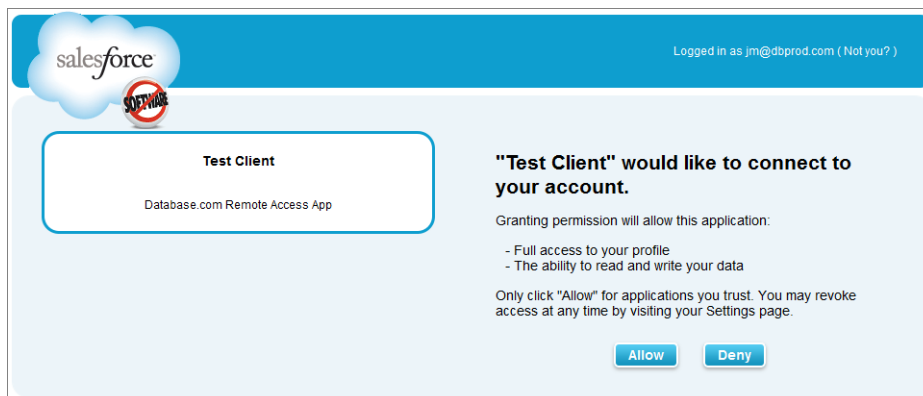
You are now running a local Java Web application server. When you visit your local web application, the OAuth servlet filter immediately kicks in because `web.xml` configures the application server to always run the authorization filter for all URLs:

```
<filter-mapping>
  <filter-name>AuthFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

As a result, as soon as you visit the website created by your application, you are redirected to Database.com, and invited to log in. To see this in action:

1. Open a browser.
2. Go to `localhost:5000/`.

Once logged in (use your admin credentials for now), you'll be asked whether you'd like to grant your application permission to access the database. This is part of the typical OAuth flow—you have to explicitly grant an application permission to access your data.



3. Click **Allow**.

You'll now be presented with the home page of the Web application, which allows you to query and create records on your Database.com instance. The application is basic, but provides you with a way to create a record, as well as perform a query. To query, insert a query in the text field and click **Query**.

Perform a Query

Insert your query:

For example, enter `select name,description__c from album__c`. The application creates an HTTP request (a GET request) to the appropriate resource on Database.com for queries, and then displays the raw output. Here's an output example.

```
{
  "done": true,
  "records": [
    {
      "Description__c": "Reggae rock",
      "Name": "Ghost in the Machine",
      "attributes": {
        "type": "Album__c",
        "url": "/services/data/v20.0/subjects/Album__c/a00T0000004xb2TIAQ"
      }
    },
    {
      "Description__c": "Swamp rock",
      "Name": "Green River",
      "attributes": {
        "type": "Album__c",
        "url": "/services/data/v20.0/subjects/Album__c/a00T0000004xb2UIAQ"
      }
    },
    {
      "Description__c": "Great cover art",
      "Name": "Who's Next",
      "attributes": {
        "type": "Album__c",
        "url": "/services/data/v20.0/subjects/Album__c/a00T0000004xb2PIAQ"
      }
    },
    {
      "Description__c": "North American release",
      "Name": "Help!",
      "attributes": {
        "type": "Album__c",
        "url": "/services/data/v20.0/subjects/Album__c/a00T0000004xb2QIAQ"
      }
    },
    {
      "Description__c": "Great rock opera",
      "Name": "The Wall",
      "attributes": {
        "type": "Album__c",
        "url": "/services/data/v20.0/subjects/Album__c/a00T0000004xb2RIAQ"
      }
    },
    {
      "Description__c": "Far out",
      "Name": "A Trick of the Tail",
      "attributes": {
        "type": "Album__c",
        "url": "/services/data/v20.0/subjects/Album__c/a00T0000004xb2SIAQ"
      }
    }
  ]
}
```

```

    }
  ],
  "totalSize": 6
}

```

Note how the query must use the "API name" for fields and objects that you created. In most cases, this is found by appending `__c` to the names. The API names are also listed in the administration console of Database.com when you navigate to fields and objects.

The output is in the *JSON* (JavaScript Object Notation) format. REST API can also use an XML format.

You can also create new records in the **Create New Record** section:

1. The first field is the resource name. Enter `album__c`.
2. For the text box, enter the value of the fields in JSON format:

```

{ "name": "Out of the Blue",
  "description__c": "The best ELO album ever" }

```

3. Click **Create**.

At this point, the application performs an HTTP POST on a URL that includes the resource name, `album__c`. The body of the POST message includes the JSON. The Database.com REST API automatically takes this data and creates the resource—in this case, a new record.

If all goes well, you'll see the response (also JSON) formatted and displayed like this.

```

{"id":"a003000000HGUp1AAH","errors":[],"success":true}

```

Step 6: Investigate How It Works

Open up each of these files to view the application source code and understand a little more about how the application works.

src/main/webapp/WEB-INF/web.xml

This file contains the configuration of the Web application. It ensures that the OAuth filter is in place and mapped to all URLs.

src/main/webapp/WEB-INF/servlet-context.xml

The Web application uses the Spring Framework's MVC container to display the web page with forms and handle the interaction. This file configures the Spring container. In particular:

- It configures an `HttpClientFactory` instance. This Java class provides the HTTP connection that is used to interact with the Database.com REST API.
- It configures a `RESTHelper` instance. This Java class performs the actual query and create calls.

src/main/java/com/developerforce/dbworkbook/HelloController.java**src/main/webapp/WEB-INF/views/home.jsp**

These files contain the user-facing Web application home page. The JSP page simply contains two forms. The Java file displays the home page, and then responds to the forms. For example, here's the method that runs when the query form is used:

```
@RequestMapping(value = "/q", method = RequestMethod.POST)
public ModelAndView performQuery(ModelAndView mv, @RequestParam String query) {
    mv.setViewName("dumpresponse");
    mv.addObject("json", rest.query(query));
    return mv;
}
```

This method ensures that the `dumpresponse` view is called to display output. The hard work—the actual query—is a result of a call to the `RESTHelper` instance, `rest.query(query)`.

src/main/java/com/developerforce/dbworkbook/HttpClientFactory.java

This Java class creates a singleton HTTP client. All REST APIs work on the basis of issuing simple HTTP actions. As a result, you need an HTTP workhorse to issue these actions.

The only tricky bit here is the establishment of a request interceptor. Successful OAuth authentication results in two things:

- An Instance URL. This URL should be used for all subsequent REST calls.
- An OAuth access token. This token should be inserted into every HTTP header of every call to the REST API. It effectively identifies you as a valid client for the period the token is valid.

Both of these data are made available by the `ForceSecurityContextHolder`, part of the OAuth framework. You'll notice how the request interceptor ensures that an HTTP header is automatically set, and uses the access token (using `getSessionId()`):

```
request.setHeader("Authorization", "OAuth " + sc.getSessionId());
```

src/main/java/com/developerforce/dbworkbook/RESTHelper.java

This class contains the meat of the interaction with the Database.com REST API. We'll walk through the `create()` method here, line by line:

```
String createURL = httpClientFactory.getInstanceURL()
+ "/services/data/" + VERSION + "/objects/" + resource + "/";
```

This line establishes the URL that we want to POST to in the Database.com REST API. Every URL is based on the Instance URL, concatenated with `/services/data,` concatenated with a version number of the API you want to use, concatenated with `/objects/`, and finally, the name of the resource.

For example, if you created a new object called `Foo`, it would be automatically represented by a URL such as `https://na1.salesforce.com/services/data/v20.0/objects/Foo__c/`.

The next few lines create an HTTP POST request to this URL, specifying the JSON description in the payload:

```
HttpPost httpPost = new HttpPost(createURL);
httpPost.setEntity(new StringEntity(recordDescription, "application/json", null));
```

Finally, we ask the HTTP Client to execute the POST.

```
HttpResponse response = httpClientFactory.getHttpClient().execute(httpPost);
```

The final set of lines just check for an error. Because you're simply performing HTTP actions, the response codes (we've all seen 404s!) determine success or not. If all goes well, we simply return the body of the response from the server, which is what's displayed in your Web application:

```
if (response.getStatusLine().getStatusCode() != 201)
    throw new Exception(EntityUtils.toString(response.getEntity()));
return EntityUtils.toString(response.getEntity());
```

That's it! Interacting with the Database.com REST API is a matter of:

- Using OAuth to perform authentication, which returns an instance URL and access token.
- Writing code to perform standard HTTP actions to a well-defined set of URLs defined by the Database.com REST API.

Summary

In this tutorial, you created, configured, and ran a Java Web application, and set up a Remote Access configuration on Database.com. The Web application used OAuth to authenticate against the database, and the Database.com REST API to execute HTTP GETs and POSTs to create records, and perform a query. These same techniques can be applied in other languages and platforms. The complete description of REST API, which also includes the full URL scheme, can be found in the documentation. See the [Database.com REST API Developer's Guide](#) for more information.

Next Steps

Now that you've completed this workbook, you should have a good introduction to Database.com concepts and a general idea of how to build applications using the Database.com REST API. To learn how to develop mobile applications for Database.com on iOS and Android, work through the [Mobile SDK Workbook](#).

To continue exploring, visit the Dev Center at www.database.com and view the official documentation for Database.com and the APIs, read the articles and tutorials, and check out the other resources there to help you build awesome applications.