Enforcing Sharing Rules

Apex generally runs in system context; that is, the current user's permissions and field-level security aren’t taken into account during code execution. Sharing rules, however, are not always bypassed: the class must be declared with the `without sharing` keyword in order to ensure that sharing rules are not enforced.

The only exceptions to this rule are Apex code that is executed with the `executeAnonymous` call and Connect in Apex. `executeAnonymous` always executes using the full permissions of the current user. For more information on `executeAnonymous`, see Anonymous Blocks.

Because these rules aren't enforced, developers who use Apex must take care that they don't inadvertently expose sensitive data that would normally be hidden from users by user permissions, field-level security, or organization-wide defaults. They should be particularly careful with Web services, which can be restricted by permissions, but execute in system context once they are initiated.

Most of the time, system context provides the correct behavior for system-level operations such as triggers and Web services that need access to all data in an organization. However, you can also specify that particular Apex classes should enforce the sharing rules that apply to the current user. (For more information on sharing rules, see the Salesforce online help.)

Enforcing sharing rules by using the `with sharing` keyword doesn’t enforce the user's permissions and field-level security. Apex code always has access to all fields and objects in an organization, ensuring that code won’t fail to run because of hidden fields or objects for a user.

This example has two classes, the first class (`CWith`) enforces sharing rules while the second class (`CWithout`) doesn’t. The `CWithout` class calls a method from the first, which runs with sharing rules enforced.
The `CWithout` class contains an inner classes, in which code executes under the same sharing context as the caller. It also contains a class that extends it, which inherits its without sharing setting.

```
public with sharing class CWith {
  // All code in this class operates with enforced sharing rules.

  Account a = [SELECT . . . ];

  public static void m() { . . . }

  static {
    . . .
  }

  {
    . . .
  }

  public void c() {
    . . .
  }
}

public without sharing class CWithout {
  // All code in this class ignores sharing rules and operates
  // as if the context user has the Modify All Data permission.
  Account a = [SELECT . . . ];
  . . .

  public static void m() {
     . . .

    // This call into CWith operates with enforced sharing rules
    // for the context user. When the call finishes, the code execution
    // returns to without sharing mode.
    CWith.m();
  }


  public class CInner {
    // All code in this class executes with the same sharing context
    // as the code that calls it.
    // Inner classes are separate from outer classes.
    . . .

    // Again, this call into CWith operates with enforced sharing rules
    // for the context user, regardless of the class that initially called this
inner class.
    // When the call finishes, the code execution returns to the sharing mode t
hat was used to call this inner class.
    CWith.m();
  }

  public class CInnerWithOut extends CWithout {
    // All code in this class ignores sharing rules because
    // this class extends a parent class that ignores sharing rules.
  }
}
```

There is no guarantee that a class declared as `with sharing` doesn't call code that operates as `without sharing`. Class-level security is always still necessary. In addition, all SOQL or SOSL queries that use PriceBook2 ignore the `with sharing`keyword. All PriceBook records are returned, regardless of the applied sharing rules.

Enforcing the current user's sharing rules can impact:

- SOQL and SOSL queries. A query may return fewer rows than it would operating in system context.
- DML operations. An operation may fail because the current user doesn't have the correct permissions. For example, if the user specifies a foreign key value that exists in the organization, but which the current user does not have access to.