SI-10 - Information Input Validation:

Attack Method: SQL Injection (https://www.owasp.org/index.php/SQL_Injection)

- Attack Overview: “A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to effect the execution of predefined SQL commands.”
- Threat Modeling:
    - SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.
    - SQL Injection is very common with PHP and ASP applications due to the prevalence of older functional interfaces. Due to the nature of programmatic interfaces available, J2EE and ASP.NET applications are less likely to have easily exploited SQL injections.
    - The severity of SQL Injection attacks is limited by the attacker’s skill and imagination, and to a lesser extent, defense in depth countermeasures, such as low privilege connections to the database server and so on. In general, consider SQL Injection a high impact severity.
- Prevention Controls: Query Parameterization
    - APEX data management applications
        - Wizard-based data management forms are implemented using the default APEX functionality. When the standard “Process Row” process is executed the record operations are automatically handled for all form fields using Query Parameterization so that each form field value is treated as a value in the SQL statement and the content will never be executed as part of a larger SQL statement. When custom SQL code is used and contains a user-defined parameter (e.g. form field value) the statement is constructed to explicitly define the bind variables and define the values when the given query/PL/SQL code is executed.
        - Examples of security control implementation:
            - Fig. 2-1 shows an example of a common SQL injection attack type that was made on a typical APEX data management form. The attacker attempts to inject additional SQL statements that will be executed as part of the larger database query. Fig. 2-2 shows that the attack was unsuccessful and that the database object was not dropped and instead the SQL statement was treated as a form field value.

Attack Method: Cross Site Scripting
(https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))

- Attack Overview: “Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it. An attacker can use XSS to send a malicious script to an unsuspecting user. The end user’s browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page. For more details on the different types of XSS flaws, see: Types of Cross-Site Scripting.”
- Prevention Controls: Convert to HTML Entities Before displaying in web page
    - Each database value that is displayed in the web application is converted to its corresponding HTML entities so that the HTML will display correctly on the page but the code will be unable to execute because all of the HTML control characters will be converted to their corresponding HTML entity (e.g.converting & to &).
        - APEX data management applications
            - In APEX the “Escape special characters” security setting is enabled by default and it causes each of the corresponding database result set field values to be converted to their corresponding HTML entities before displaying the content in the page which renders potentially injected client-side code harmless.
            - Examples of security control implementation:
                - Fig. 3-1 shows a simple XSS attack by injecting executable HTML code in a standard APEX form with “Escape special characters” setting enabled. Fig. 3-2 shows the HTML code displayed correctly on the web page. Fig. 3-3 shows the underlying source code of the HTML page shown in Fig. 3-2 that has had its values converted to prevent the HTML code from actually being executed by the browser.

SI-11 - Error Handling:

- Attack Overview: Malicious users could experiment with form inputs that generate error messages in the given application. If these error messages provide technical details those could potentially be used to launch directed attacks at the underlying servers.
- Prevention Control: Global Custom Error Handling Module (Git: git@pichub.pifsc.gov:application-development/apex_tools.git in the "Error Handling" folder) implemented for all APEX application pages. There is a PL/SQL function that has been developed to suppress error message details on all application instances except

for midd.pic.gov since it is explicitly defined as a development environment only and therefore not subject to the same constraints.

- Examples of security control implementation:
  - The error handling function will check a user-defined table to see if there is a matching constraint name for the given database error and if so it will use that custom error message (shown in Fig 4-2). If there is no matching user-defined custom error message a generic error message is used instead (shown in Fig 4-3) that does not include structural information. The default APEX framework functionality is shown in Fig 4-1 which presents a security problem and the only application platform that will be allowed to view the detailed error code/description is midd.pic.gov because this information is useful for development purposes.

General APEX Checks:

- Each page in the application is evaluated using the APEX Advisor with all available checks enabled, if there are issues reported those are resolved. Otherwise, the results of the check are saved for future reference (show in Fig 5-1).

Fig. 2-1 (example of a SQL Injection Attack) - This screenshot shows a common SQL injection attack that attempts to add an additional database query to be executed when a record save operation is attempted

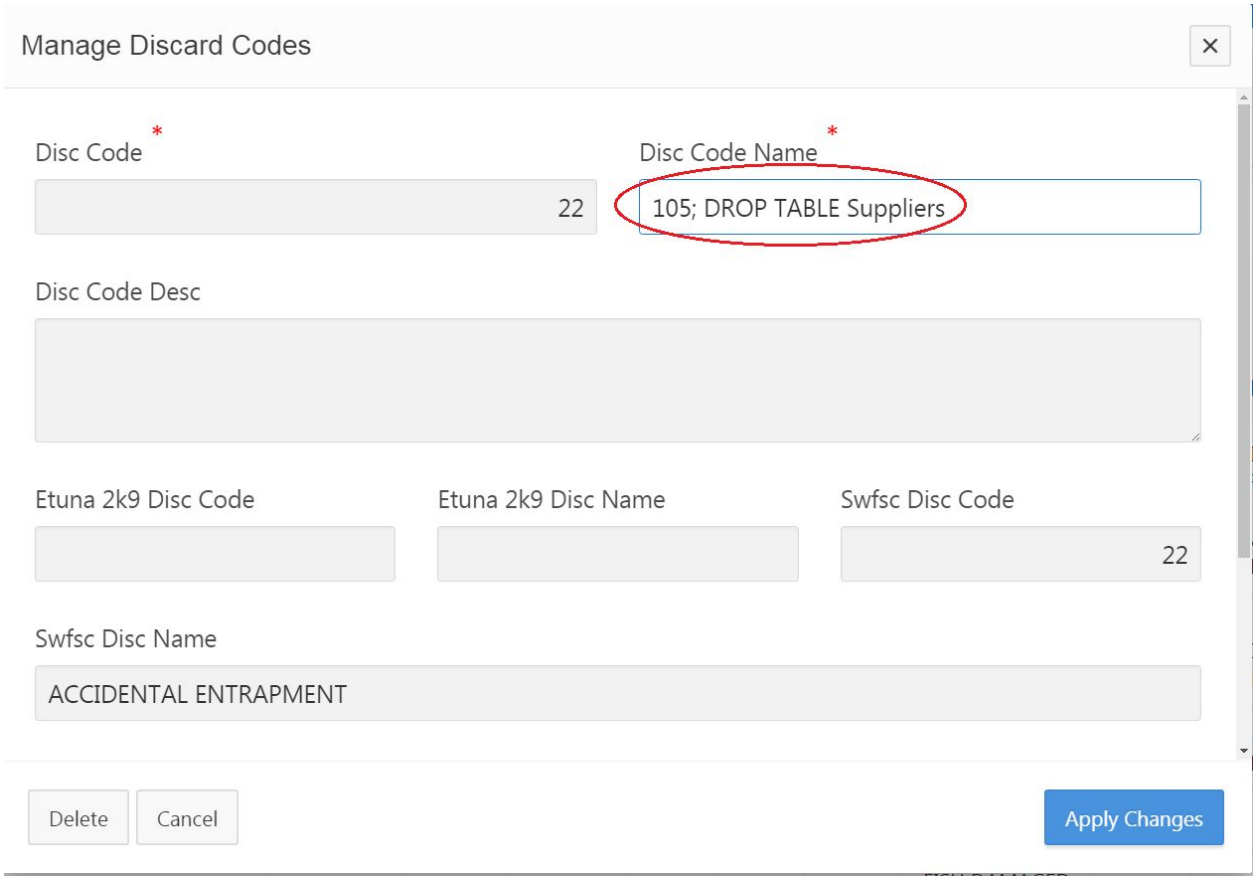

Fig. 2-2 (SQL Injection Attack Results) - This screenshot shows that the attack attempt was treated as a form field value that was used to set the database field value instead of actually executing the statement:

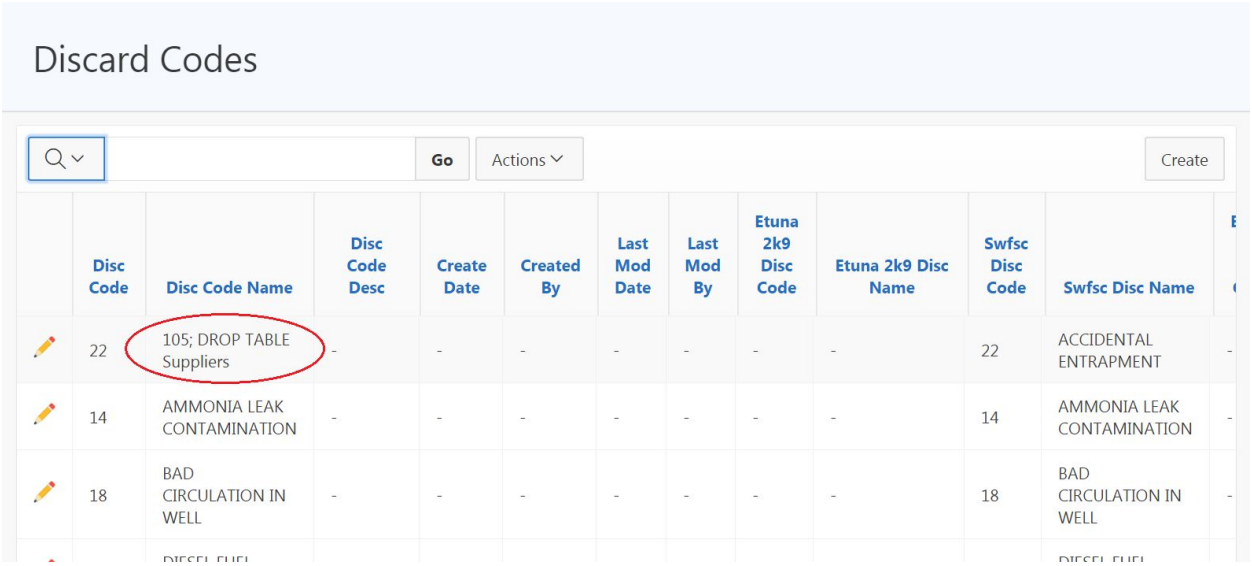

Fig. 3-1 (APEX XSS attack example) - This screenshot shows a simple XSS attack attempt to inject Javascript code into the database with the intent that the HTML code will execute each time it is displayed on the page and attempt to compromise the user's web browser.

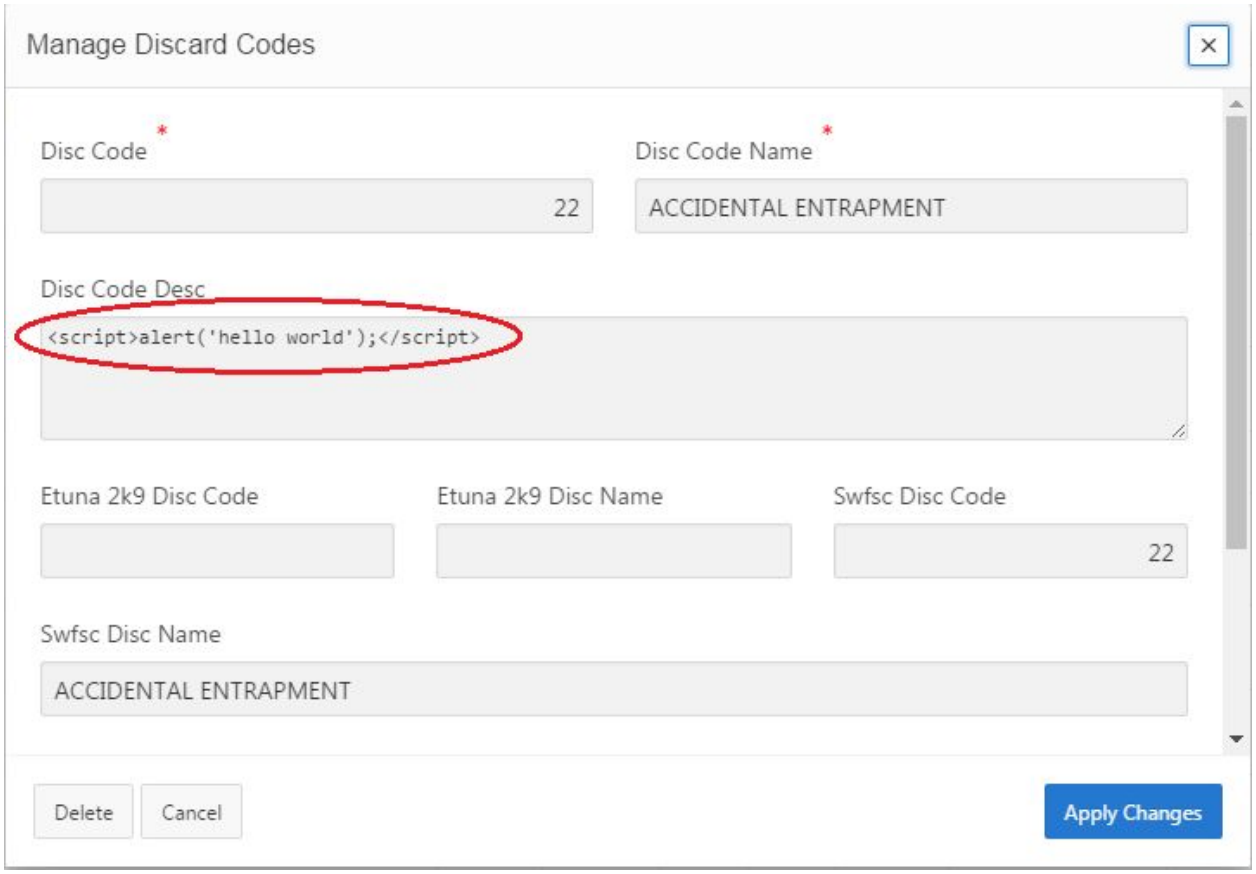

Fig. 3-2 (APEX XSS screenshot) - This screenshot shows the HTML entered by the XSS attack displayed on a web page and displayed correctly. This code does not actually execute in the web browser

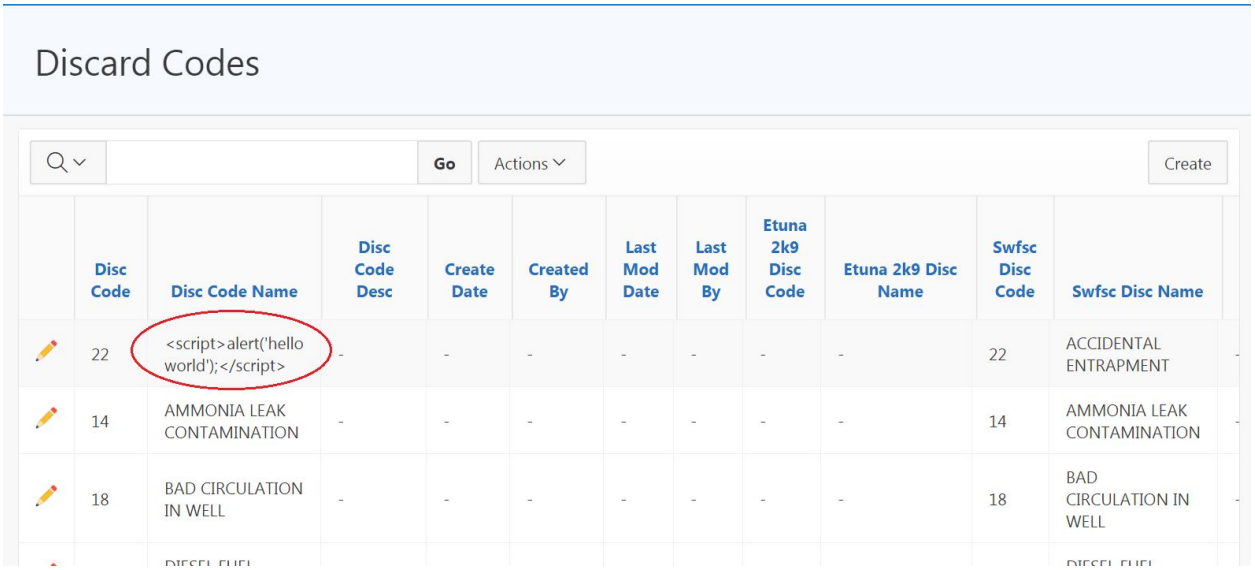

Fig. 3-3 (XSS attack mitigation by APEX security settings) - This screenshot shows the source code of the web page shown in Fig. 3-2 that has converted the HTML code into its corresponding HTML entities that prevents the injected HTML code from being executed in the web browser:

Fig. 4-1 (default error handling):

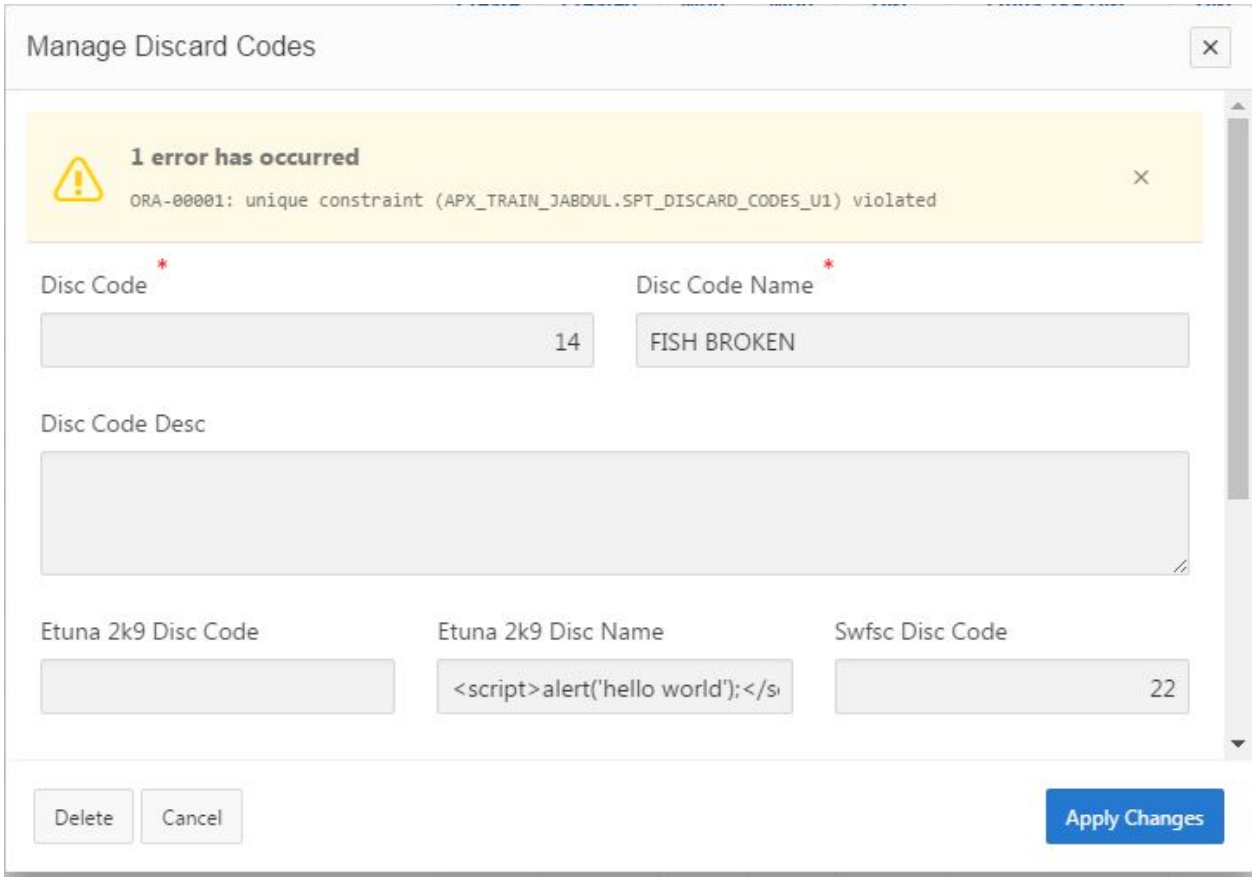

Fig. 4-2 (custom error handling):

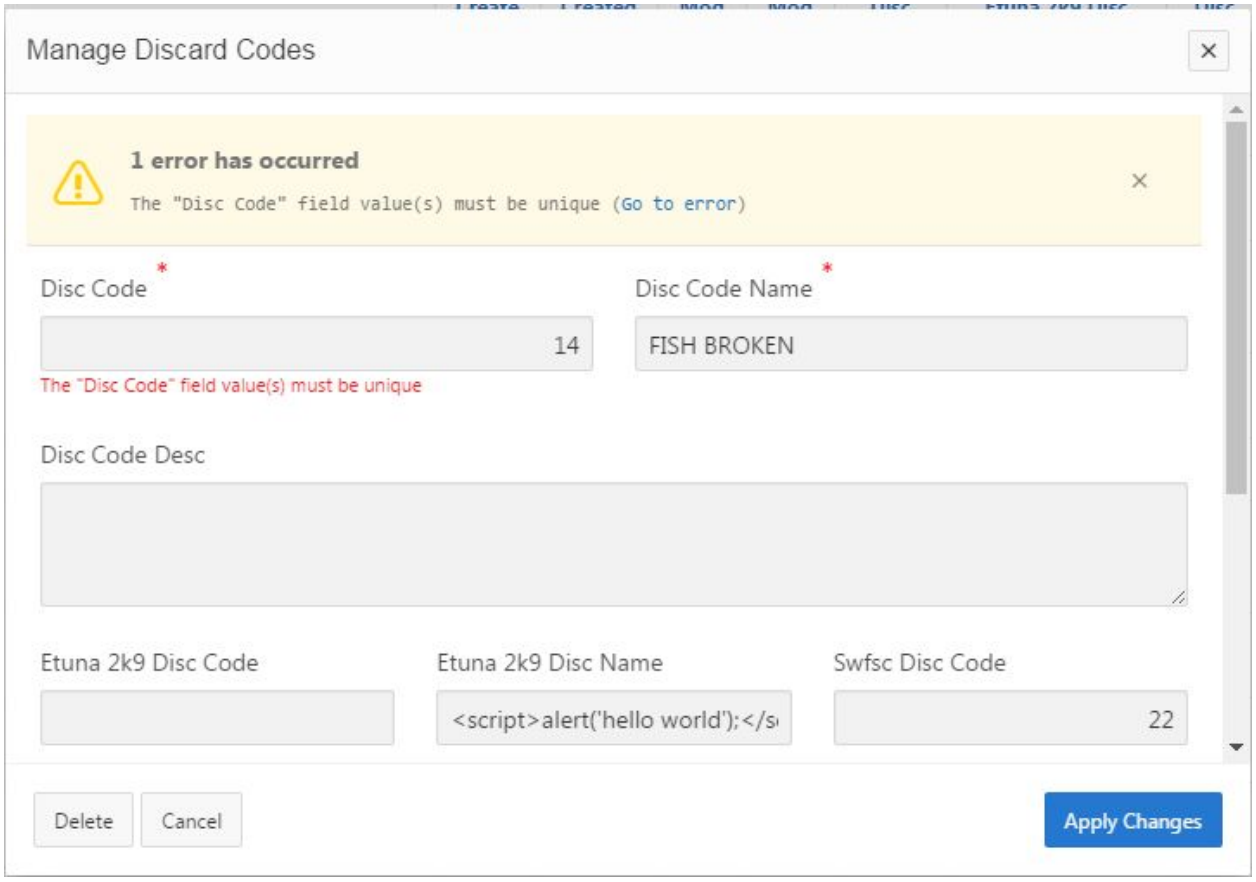

Fig. 4-3 (custom error handling without an explicit matching custom error message):

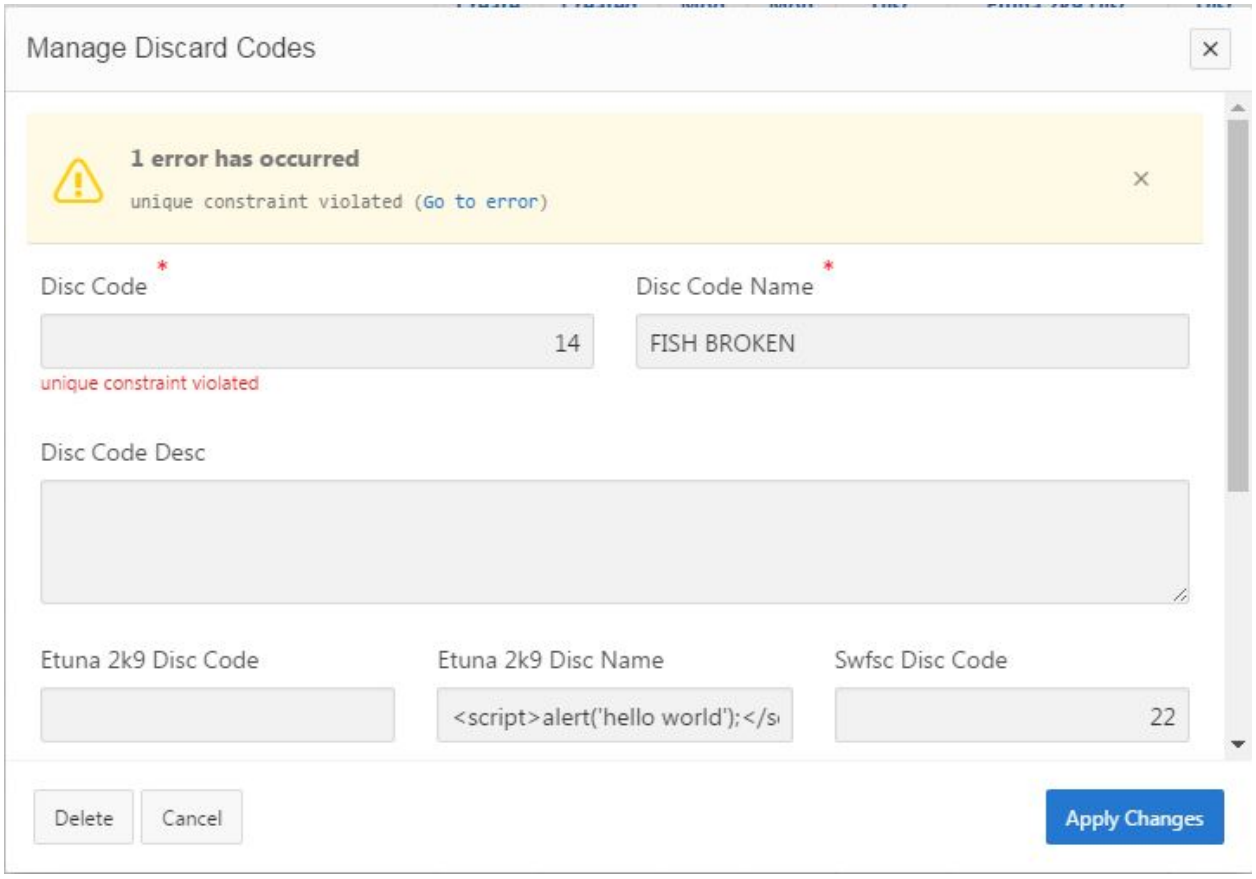

Fig. 5-1 (The results of an APEX Advisor check on a given application that shows there are no problems that Advisor identified)