

# Secure Perfect 6.0 Database Triggers

## Contents

- [“Overview” on page 1](#)
- [“Import Prerequisites” on page 2](#)
- [“Tables Overview” on page 2](#)
- [“Tables Defined” on page 3](#)
- [“Triggers” on page 13](#)
- [“Data Manipulation” on page 15](#)
- [“Database Connectivity” on page 21](#)
- [“Troubleshooting” on page 23](#)
- [“Printing a Report on Records Downloaded to Micros” on page 24](#)

## Overview

This document is intended to be a guide to importing and linking external database systems to personnel data in the Secure Perfect database.

Data must be extracted from a source, formatted to conform with the appropriate Secure Perfect record structure, and inserted into the `SecurePerfect` database. For example, if personnel information (such as name, address, department, identification number, and employment status) is already available in a Human Resources database, that information can be imported into the `SecurePerfect` database.

When importing data into the `SecurePerfect` database, a download to the micro will only occur in the following scenarios:

- When data is inserted into the `BadgeTable`, `BadgeUserField`, `PersonAccessRightMapTable`, or `PersonTable`
- When data is deleted from the `BadgeTable`, `BadgeUserField`, or `PersonAccessRightMapTable`
- When an update occurs in the `BadgeTable`, `BadgeUserField`, `BadgeAccessTable` or `PersonTable`

If data is inserted, updated, or deleted from an appropriate table, a database trigger will fire and call an extended stored procedure. The micro must be online to successfully complete a download to the micro database.

## NOTE



If the micro is offline, Secure Perfect will route the import data to the `OfflineDownload` table. When the micro returns to online, the information will be downloaded to the micro database.

Six extended stored procedures are used:

1. `xp_BadgeDelete`
2. `xp_BadgeInsert`
3. `xp_BadgeUpdate`
4. `xp_PersonAccessRightsMapDelete`
5. `xp_PersonAccessRightsMapInsert`
6. `xp_PersonTableUpdate`

The extended stored procedures are located in the master database of the SPSQL instance on the SQL 2000 Server computer. These extended stored procedures are bound to `xspDatabaseDownload.dll` located in the

`\Program Files\GE-Interlogix\Secure Perfect`

folder or your installed path. The `xspDatabaseDownload.dll` is registered to the Microsoft SQL 2000 services. If you need to uninstall Secure Perfect, you must shut down SQL 2000 services to release the `xspDatabaseDownload.dll` from use.

## Import Prerequisites

- To perform the required import/export procedures, the following knowledge base is essential:

- ☐ Structured Query Language (SQL) and SQL 2000 Server
- ☐ Programming skills in Visual Basic, C++, or other program that can import data into an SQL 2000 database
- ☐ Access control concepts

## Tables Overview

The database tables involved in importing, exporting, or linking processes are as follows:

### **PersonTable**

This table stores the minimum data needed to represent a person in the `SecurePerfect` database.

### **UserFieldTable**

This table stores user-definable characteristics for a specific person in the `PersonTable`.

### **BadgeTable**

This table stores data representing a badge that has been inserted into the system.

## NOTE



A badge can exist without being assigned to a person.

---

#### **BadgeUserField**

This table stores user-definable characteristics for a specific badge in the **BadgeTable**.

#### **BadgeAccessTable**

This table stores data representing a badge's last valid access that has been inserted into the system.

#### **PersonAccessRightMapTable**

This table maps the person to a specific access right in the **AccessRightTable**.

#### **FacilityTable**

This table stores data representing facility identification.

#### **DepartmentTable**

This table stores data representing department and facility identification.

#### **PersonTypeTable**

This table is an association table; it stores personnel-type information such as **Permanent** or **Contractor**. Nothing will be imported into it.

#### **NOTE**

Refer to **SQL 2000 Enterprise Manager** for detailed information about **Secure Perfect** database schema.



## **Tables Defined**

### **PersonTable**

The **PersonTable** Triggers are listed in [Table 12, "PersonTable Triggers," on page 14](#). A **PersonTable** record will be downloaded to the micro's database if there is an associated badge and an access right. If a **PersonTable** record has a badge but no access rights, a download to the micro will NOT occur. If a **PersonTable** record has an access right but no badge, a download to the micro will NOT occur.

When importing personnel data into the **SecurePerfect** database, start with the **PersonTable**. The **PersonTable** holds common identifying characteristics for individuals who will receive or have received badges.

[Table 1](#) lists all column names in the **PersonTable** along with the type, default value, and description.

See [page 15](#) for the triggers associated with the **PersonTable**.

---

**Table 1: PersonTable**

Column Name	Type	Nullable	Default	Description
Id (auto identity)	int	Not Null		Primary key
FirstName	nvarchar(32)	Null		Person's first name
LastName	nvarchar(32)	Not Null		Person's last name
MiddleName1	nvarchar(32)	Null		Person's first middle name
MiddleName2	nvarchar(32)	Null		Person's second middle name
Initials	nvarchar(3)	Null		First character from firstname, middlename, and middlename2
EmployeeNumber	nvarchar(12)	Not Null		Information that uniquely identifies person (Number must be at least four characters.)
Address1	nvarchar(64)	Null		Location of person or e-mail
Address2	nvarchar(64)	Null		Location of person or e-mail
Address3	nvarchar(64)	Null		Location of person or e-mail
Address4	nvarchar(64)	Null		Location of person or e-mail
Address5	nvarchar(64)	Null		Location of person or e-mail
Telephone	nvarchar(14)	Null		Person's phone number
PersonTypeId	int	Not Null		Foreign key to PersonTypeTable
DepartmentId	int	Null		Department to which person belongs (See DepartmentTable for Id.)
Traced	tinyint	Not Null	0	Is the person being traced? 0 = no 1 = yes
Photo	nvarchar(255)	Null		Name of file containing photographic image of person; name format is <personId.jpg>

**Table 1: PersonTable (Continued)**

Column Name	Type	Nullable	Default	Description
FacilityId	int	Not Null	1	Foreign key to FacilityTable
Modified <sup>1</sup>	float(53)	Not Null	0	Date and time record was last modified

1. The Modified column is a decimal number with a zero(0) date equivalent to the date 1899-12-30 00:00:00.000. SQL Server uses a zero(0) date equivalent date of 1900-01-01 00:00:00.000. If you use getdate() or any other date function in SQL Server, you must subtract 2 to make the time consistent with other tables in the database. If the modified value retrieved from a table is 37328.332404243825, you would get the correct date by using 'select cast(37328.332404243825 as datetime) - 2'.

## UserFieldTable

The **UserFieldTable** stores user-definable characteristics for a specific person in the **PersonTable**. The **PersonTable** and the **UserFieldTable** have a one-to-one relationship. For every record in the **PersonTable**, there must be a corresponding record in the **UserFieldTable**. The minimum information for the **UserFieldTable** will consist of the corresponding **PersonTableId** in the **PersonId** column of the **PersonTable** and the time the record was changed in the **Modified** column.

**Table 2** lists all column names in the **UserFieldTable** along with the type, default value, and description.

**Table 2: UserFieldTable**

Column Name	Type	Nullable	Default	Description
PersonId	int	Not Null		Foreign key to the PersonTable
Value1	nvarchar(32)	Null		User-definable data
:				
:				
Value 90	nvarchar(32)	Null		User-definable data
Modified <sup>1</sup>	float(53)	Not Null	0	Date and time record was last modified

1. The Modified column is a decimal number with a zero(0) date equivalent to the date 1899-12-30 00:00:00.000. SQL Server uses a zero(0) date equivalent date of 1900-01-01 00:00:00.000. If you use getdate() or any other date function in SQL Server, you must subtract 2 to make the time consistent with other tables in the database. If the modified value retrieved from a table is 37328.332404243825, you would get the correct date by using 'select cast(37328.332404243825 as datetime) - 2'.

---

## BadgeTable

The **BadgeTable** holds information on a badge that has been inserted into the system.

[Table 3](#) lists all column names in the **BadgeTable** along with the type, default value, and description. Following the table is a list of columns with additional import information.

See [page 14](#) for the triggers associated with the **BadgeTable**.

**Table 3: BadgeTable**

Column Name	Type	Nullable	Default	Description
Id	int	Not Null		Primary key
Description	nvarchar(64)	Null		Unique descriptive text
PersonId	int	Null		Person record assigned to the badge record (This is not a foreign key. A badge can exist without being assigned to a person.)
EncodedNumber	nvarchar(20)	Not Null		Number encoded into the card. (Only numbers and blanks are allowed and all numbers must be contiguous with any blanks leading all numbers.)
AliasNumber	nvarchar(20)	Not Null		Number that may be used to hide encoded number
Status	tinyint	Not Null	1	Badge's status (One of the following: 1 = Active 2 = Issuable 3 = Suspended 4 = Lost 5 = Remake
PIN	nvarchar(4)	Null		Personal Identification Number (Only numbers and blanks are allowed and all numbers must be contiguous with any blanks leading all numbers.)

**Table 3: BadgeTable (Continued)**

Column Name	Type	Nullable	Default	Description
ReturnDate <sup>1</sup>	float(53)	Null		Date badge was last unassigned from a person - Null when it was never assigned
IssueDate <sup>1</sup>	float(53)	Null	0	Date the badge was assigned or the date the badge will become active <sup>2</sup>
DueDate <sup>1</sup>	float(53)	Null	0	Date that badge is no longer allowed access
Modified <sup>1</sup>	float(53)	Not Null	0	Date and time when record was last modified
FacilityId	int	Not Null	1	Foreign key to the FacilityTable

1. The Modified column and all columns representing a date are decimal numbers with a zero(0) date equivalent to the date 1899-12-30 00:00:00.000. SQL Server uses a zero(0) date equivalent date of 1900-01-01 00:00:00.000. If you use getdate() or any other date function in SQL Server, you must subtract 2 to make the time consistent with other tables in the database. If the modified value retrieved from a table is 37328.332404243825, you would get the correct date by using 'select cast(37328.332404243825 as datetime) - 2'.

2. A badge set to Active before the issue date will result in a badge that will provide valid access.

Importing data into the **BadgeTable** to create a new badge without the use of Secure Perfect software can be accomplished. The sections that follows provides additional information required to successfully import data into the **BadgeTable**.

#### **EncodedNumber**

Number stored in the badge and used by readers to identify a specific badge. The encoded number is built into the card. When a card is issued, the number is read from the card and inserted into the **BadgeTable**. The encoded number type is nvarchar and must be between 4 and 20 contiguous numeric characters in length. If you plan to import encoded numbers, they must be unique and between 4 and 20 characters in length.

The EncodedNumber column cannot be updated. Once data is inserted into the column, it cannot be changed.

#### **Alias Number**

Any artificially created number that is used to hide the encoded number, or a copy of the encoded number. If aliasing is turned off, both columns, Encoded Number and AliasNumber, will have the same value. The alias number is the number displayed to operators of Secure Perfect. This allows Secure Perfect to 'hide' the real badge encoded numbers from anybody using Secure Perfect.

The AliasNumber column cannot be updated. Once data is inserted into the column, it cannot be changed. For more information on aliasing, see the Secure Perfect Online Help.

---

**PIN**

PIN is only used with a badge and keypad reader. The PIN must be four digits.

**Status**

Can have values from 1 to 4

where:

- |   |   |
|---|---|
| 1 | Indicates badge is active with a valid PersonId in the PersonId column.             |
| 2 | Indicates badge is issuable and the PersonId column is null.                        |
| 3 | Indicates badge is suspended. A suspended badge will not have access to any reader. |
| 4 | Indicates badge is lost. A lost badge will not have access on any reader.           |

When creating a badge for the first time, you do not have to assign it to a person in the **PersonTable** (PersonId column is nullable). If the PersonId column is left null, the row is not downloaded to the micro.

All columns representing a date are decimal numbers with a zero(0) date equivalent to the date 1899-12-30 00:00:00.0000. SQL server uses a zero(0) equivalent date of 1900-01-01 00:00:00.000. If you use getdate() or any other date function in SQL server, you must subtract 2 to make the time consistent with other tables in the database.

**ReturnDate**

Date the badge will be unassigned from a person. The ReturnDate value is of type float and a decimal representation of a date.

The ReturnDate column is informational only which means that changing this column will not cause a download to the micro to occur. The status can be set to 2 to make the badge active and issuable. The status can be set to 3 (suspended) to remove badge access. Once the status is updated, the micro database will be updated.

**IssueDate**

Date the badge was assigned or the date the badge will become active. The IssueDate column is of type float and a decimal representation of a date. When the IssueDate and the current date are the same, the micro's database will automatically allow the badge to have access.

**DueDate**

Date a badge is no longer allowed access. The DueDate column is of type float and a decimal representation of a date. When the DueDate date and the current date are the same, the micro's database automatically invalidates that badge's access. The maximum DueDate cannot exceed the current date plus 9 years. The status column does not automatically change when the DueDate is greater than or equal to the current date.



---

## BadgeAccessTable

The **BadgeAccessTable** stores data indicating location and date/time of last valid access. The **BadgeAccessTable** and the **BadgeTable** are closely linked. For each record in the **BadgeTable** a record in the **BadgeAccessTable** must exist. For more information on Time and Attendance, refer to the Secure Perfect Online Help.

[Table 4](#) lists all column names in the **BadgeAccessTable** along with the type, default value, and description.

**Table 4: BadgeAccessTable**

Column Name	Type	Nullable	Default	Description
BadgeId	int	Not Null		Primary key
LastAccess <sup>1</sup>	float(53)	Null	0	Date and time when a valid badge accessed reader
APBStatus	int	Not Null	0	Current global anti-passback status: 0 = Neutral 1 = In 2 = Out
TASStatus	int	Not Null	0	Current global time and attendance status: 0 = Neutral 1 = In 2 = Out
ReaderId	int	Null	0	Reader that was last accessed

1. This date column is a decimal number with a zero(0) date equivalent to the date 1899-12-30 00:00:00.000. SQL Server uses a zero(0) date equivalent date of 1900-01-01 00:00:00.000. If you use getdate() or any other date function in SQL Server, you must subtract 2 to make the time consistent with other tables in the database. If the modified value retrieved from a table is 37328.332404243825, you would get the correct date by using 'select cast(37328.332404243825 as datetime) - 2'.

During an import, the following values are set:

### BadgeId

Corresponding BadgeId in the **BadgeTable**.

### Last Access

Will be zero(0) by default. The zero represents a badge that has never accessed a reader.

### APBStatus

### TASStatus

Will be set to zero(0) which is neutral.

## ReaderId

Will be null when importing in the initial record.

### NOTE



After the initial record has been imported, do not manipulate the **BadgeAccessTable** directly.

## BadgeUserField

The **BadgeUserField** stores user-definable characteristics for a specific person in the **BadgeTable**. The **BadgeTable** and the **BadgeUserField** have a one-to-one relationship. For every record in the **BadgeTable**, there must be a corresponding record in the **BadgeUserField**. The minimum information for the **BadgeUserField** will consist of the corresponding **BadgeTableId** in the **BadgeId** column of the **BadgeTable** and the time the record was changed in the **Modified** column.

[Table 2](#) lists all column names in the **BadgeUserField** along with the type, default value, and description.

**Table 5: BadgeUserField**

Column Name	Type	Nullable	Default	Description
BadgeId	int	Not Null		Foreign key to the <b>BadgeTable</b>
BadgeUserField1	nvarchar(32)	Null		User-definable data
:				
:				
BadgeUserField20	nvarchar(32)	Null		User-definable data
Modified <sup>1</sup>	float(53)	Not Null	0	Date and time record was last modified

1. The **Modified** column is a decimal number with a zero(0) date equivalent to the date 1899-12-30 00:00:00.000. SQL Server uses a zero(0) date equivalent date of 1900-01-01 00:00:00.000. If you use `getdate()` or any other date function in SQL Server, you must subtract 2 to make the time consistent with other tables in the database. If the modified value retrieved from a table is 37328.332404243825, you would get the correct date by using `'select cast(37328.332404243825 as datetime) - 2'`.

## PersonAccessRightMapTable

The **PersonAccessRightMapTable** maps the person to a specific access right in the **AccessRightTable**.

[Table 6](#) lists all column names in the **PersonAccessRightMapTable** along with the type, default value, and description.

See [page 15](#) for the triggers associated with the **PersonAccessRightMapTable**.

**Table 6: PersonAccessRightMapTable**

Column Name	Type	Nullable	Default	Description
PersonId	int	Not Null		Foreign key to the PersonTable
AccessRightId	int	Not Null		Foreign key to the AccessRightTable
Modified <sup>1</sup>	float(53)	Not Null	0	Date and time the record was last modified

1. The Modified column is a decimal number with a zero(0) date equivalent to the date 1899-12-30 00:00:00.000. SQL Server uses a zero(0) date equivalent date of 1900-01-01 00:00:00.000. If you use getdate() or any other date function in SQL Server, you must subtract 2 to make the time consistent with other tables in the database. If the modified value retrieved from a table is 37328.332404243825, you would get the correct date by using 'select cast(37328.332404243825 as datetime) - 2'.

When inserting or updating into the **PersonAccessRightMapTable**, make sure that a **PersonId** from the **PersonTable** and an **AccessRightId** from the **AccessRightTable** exist. Access is granted to a person record, not a badge; keep this in mind when you are working with the **SecurePerfect** database.

A Person record can be associated with multiple badges; however, a badge CANNOT have multiple person records. The **PersonAccessRightMapTable** is the table that will grant an individual person access to specific areas already set up by Secure Perfect software.

**NOTE**



You must create Access Rights within the Secure Perfect system. Do not try to create and import new access rights.

---

## FacilityTable

The **FacilityTable** stores data representing facility identification.

**Table 7** lists all column names in the **FacilityTable** along with the type, default value, and description.

**Table 7: FacilityTable**

Column Name	Type	Nullable	Default	Description
Id (auto identity)	int	Not Null		Primary key
Description	nvarchar(64)	Not Null		Unique descriptive text
Modified <sup>1</sup>	float(53)	Not Null	0	Date and time the record was last modified

1. The Modified column is a decimal number with a zero(0) date equivalent to the date 1899-12-30 00:00:00.000. SQL Server uses a zero(0) date equivalent date of 1900-01-01 00:00:00.000. If you use getdate() or any other date function in SQL Server, you must subtract 2 to make the time consistent with other tables in the database. If the modified value retrieved from a table is 37328.332404243825, you would get the correct date by using 'select cast(37328.332404243825 as datetime) - 2'.

## DepartmentTable

The **DepartmentTable** stores data representing department and facility identification.

**Table 8** lists all column names in the **DepartmentTable** along with the type, default value, and description.

**Table 8: DepartmentTable**

Column Name	Type	Nullable	Default	Description
Id (auto identity)	int	Not Null		Primary key
Description	nvarchar(64)	Not Null		Unique descriptive text
FacilityId	int	Not Null	1	Foreign key to the FacilityTable
Modified <sup>1</sup>	float(53)	Not Null	0	Date and time record was last modified

1. The Modified column is a decimal number with a zero(0) date equivalent to the date 1899-12-30 00:00:00.000. SQL Server uses a zero(0) date equivalent date of 1900-01-01 00:00:00.000. If you use getdate() or any other date function in SQL Server, you must subtract 2 to make the time consistent with other tables in the database. If the modified value retrieved from a table is 37328.332404243825, you would get the correct date by using 'select cast(37328.332404243825 as datetime) - 2'.

---

## PersonTypeTable

The **PersonTypeTable** stores data representing personnel type such as **Permanent** or **Temporary**.

**Table 9** lists all column names in the **PersonTypeTable** along with the type, default value, and description.

**Table 9: PersonTypeTable**

Column Name	Type	Nullable	Default	Description
Id (auto identity)	int	Not Null		Primary key
Description	nvarchar(64)	Not Null		Unique descriptive text
BadgeDesignId	int	Null		Badge design used by default for printing badges for this person type
FacilityId	int	Null	1	Foreign key to the FacilityTable
Modified <sup>1</sup>	float(53)	Not Null	0	Date and time the record was last modified

1. The Modified column is a decimal number with a zero(0) date equivalent to the date 1899-12-30 00:00:00.000. SQL Server uses a zero(0) date equivalent date of 1900-01-01 00:00:00.000. If you use getdate() or any other date function in SQL Server, you must subtract 2 to make the time consistent with other tables in the database. If the modified value retrieved from a table is 37328.332404243825, you would get the correct date by using 'select cast(37328.332404243825 as datetime) - 2'.

## Triggers

A trigger is called to download changes to a micro whenever fields in certain columns are deleted, inserted, or updated. Only three tables contain triggers that deal with the import process: **BadgeTable**, **PersonAccessRightMapTable**, and **PersonTable**.

**Table 10: BadgeTable Triggers**

Action	Trigger Column(s)	Trigger Fired	Extended Stored Procedure
Delete	All columns in the BadgeTable All columns in the BadgeAccessTable	tr_Badge_Delete	xp_BadgeDelete
Insert	Minimum of: EncodedNumber, AliasNumber, Status, Modified, FacilityId, IssueDate	tr_Badge_Insert	xp_BadgeInsert
Update	status or duedate or pin or personId	tr_Badge_Update	xp_BadgeUpdate

**Table 11: PersonAccessRightMapTable Triggers**

Action	Trigger Column(s)	Trigger Fired	Extended Stored Procedure
Delete	All	tr_person_access_rights_map_delete	xp_PersonAccessRightMapDelete
Insert	All	tr_person_access_rights_map_insert	xp_PersonAccessRightMapInsert
Update	Not Applicable		

**Table 12: PersonTable Triggers**

Action	Trigger Column(s)	Trigger Fired	Extended Stored Procedure
Delete	Not Applicable		
Insert	FacilityId	tr_Facility_PersonTable	none
Update	Traced	tr_TracedStatus	xp_PersonTableUpdate

**NOTE**

The Triggers are detailed in the sections that follow.



## BadgeTable Triggers

The **BadgeTable** Triggers are listed in [Table 10](#).

### **tr\_Badge\_Delete**

A delete trigger on the **BadgeTable**. When the tr\_Badge\_Delete trigger fires, it calls the xp\_BadgeDelete extended stored procedure that calls the **xspDatabaseDownload.dll** to remove the badge from the micro's database.

### **tr\_Badge\_Insert**

An insert trigger on the **BadgeTable**. When the tr\_Badge\_Insert trigger fires, it checks if the facilityId exists. If the facilityId exists, the transaction proceeds normally. If the facilityId does not exist, a one(1) is inserted for the facilityId. A one(1) for facilityId means Ignore Facilities. After the facility check, the xp\_BadgeInsert extended stored procedure is called. If there is a personId and the person record has access rights, the badge information is downloaded to the micro. If the personId field is null and/or the person record has no access rights, a download to the micro will not occur.

When a record is inserted into the **BadgeTable**, you must insert a corresponding record into the **BadgeAccessTable**. The record will be the BadgeID of the record just inserted into the **BadgeTable**, LastAccess with a value of zero (0), APBstatus with a value of zero(0), TAsatus with a value of (0), and a ReaderId set to zero(0).

---

#### **tr\_Badge\_Update**

An update trigger on the **BadgeTable**. When the tr\_Badge\_Update trigger fires, it checks the EncodedNumber or AliasNumber, Status, DueDate, PIN, IssueDate, and PersonId. If the EncodedNumber or AliasNumber is updated, an error will be fired and the transaction will be rolled back. If the EncodedNumber and AliasNumber are untouched, then the trigger will check if the status, IssueDate, dueDate, PIN, or PersonId columns have been updated. If one or more of the previously noted columns have been updated, a download to the micro will occur. The download to the micro will only occur if the badge is associated with a valid personId and the person has access rights.

## **PersonAccessRightMapTable Triggers**

The **PersonAccessRightMapTable** Triggers are listed in [Table 11](#).

#### **tr\_person\_access\_rights\_map\_delete**

A delete trigger on the **PersonAccessRightMapTable**. When this trigger fires, it will call the xp\_PersonAccessRightsMapDelete extended stored procedure and the associated badge record will be removed from the micros.

#### **tr\_person\_access\_rights\_map\_insert**

An insert trigger on the **PersonAccessRightMapTable**. When this trigger fires, the xp\_PersonAccessRightMapInsert extended stored procedure inserts a badge record into the micros.

## **PersonTable Triggers**

The **PersonTable** Triggers are listed in [Table 12](#).

#### **tr\_Facility\_PersonTable**

An update and insert trigger on the **PersonTable**. The trigger checks if the updated or inserted facilityId exists. If the facilityId does not exist, the facilityId is set to 1 (ignore facilities). If the facilityId does exist, then the transaction proceeds normally.

#### **tr\_TracedStatus**

An update trigger on the **PersonTable** TRACE column. The trigger checks if the TRACE column has changed during the update. If the column has been updated, then the xp\_PersonTableUpdate extended stored procedure is called and a download to the micro will occur.

## **Data Manipulation**

There are five possible scenarios where data will be imported or updated into the **SecurePerfect** database:

1. A one-time mass import of people and badges
2. A one-time mass update of people and badges
3. Inserting individual records on a continuous basis from an application outside of Secure Perfect.

- 
4. Updating individual records on a continuous basis from an application outside of Secure Perfect
  5. Deleting records from the database

The micros must be online for any insert, update, or delete to complete successfully. If the micro is offline, Secure Perfect will route the import data to the `OfflineDownload` table. When the micro returns to online, the information will be downloaded to the micro database.

## Import Methods

Data can be imported into the database using SQL 2000 bulk insert commands or DTS. If bulk insert or DTS will be used, they must be configured to allow triggers to be fired. The setting for the bulk insert is `FIRE_TRIGGER`. For DTS, the `FastLoad` option must be set to false.

### NOTE



If you plan to import data into the `BadgeTable`, `PersonAccessRightMapTable`, or `PersonTable`, you must use a method to insert or update records that allows the triggers to be fired.

There are other methods to connect via Visual Basic, Visual C++.

Sources of additional information:

- SQL Books Online
- The following topics of Microsoft documentation: **ADO**, **OLEDB**, and **connect**.

If you choose to install the samples during installation of SQL Server, the following is a helpful sample:

```
<PATH>\Microsoft SQL Server\80\Tools\DevTools\Samples\ado\vb\intro
```

## Mass Import of People and Badges

Mass imports and updates should be executed only during off-peak hours.

### NOTE



Previous versions of Secure Perfect required an individual update for each record; this is no longer necessary. Statements such as *UPDATE PersonTable SET Traced = 1 WHERE [ID] IN (3,5,67,43,126,78,96)* will now work.

#### ► To mass import people and badges:

1. Insert records into the `PersonTable`. If your person records contains `DepartmentId` or `FacilityId`, make sure that each ID exists in the `DepartmentTable` and `FacilityTable` respectively.
2. For every record inserted into the `PersonTable`, a corresponding record needs to be added to the `UserFieldTable`. The minimum columns needed are the `PersonId` and the `Modified` columns. The modified column is the date and time the record was last modified.
3. Insert the badge records into the `BadgeTable` and `BadgeUserField`. If the insert is missing a `PersonId`, there will be no download to the micro. If aliasing is turned on,

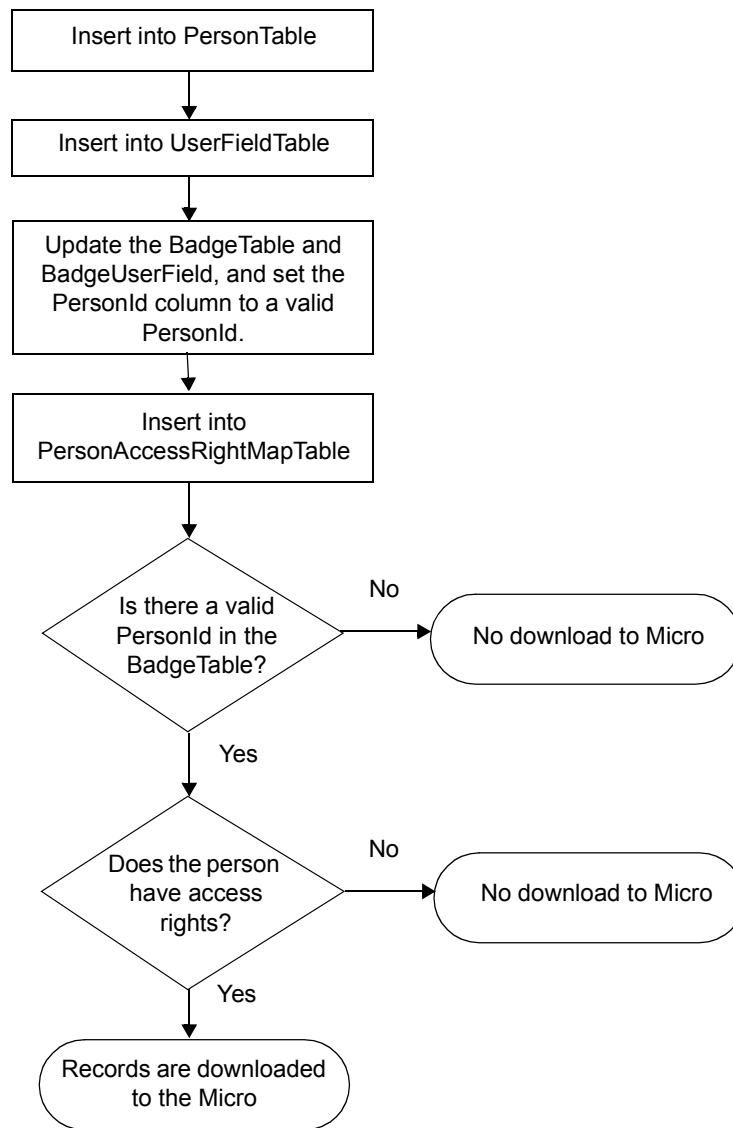


---

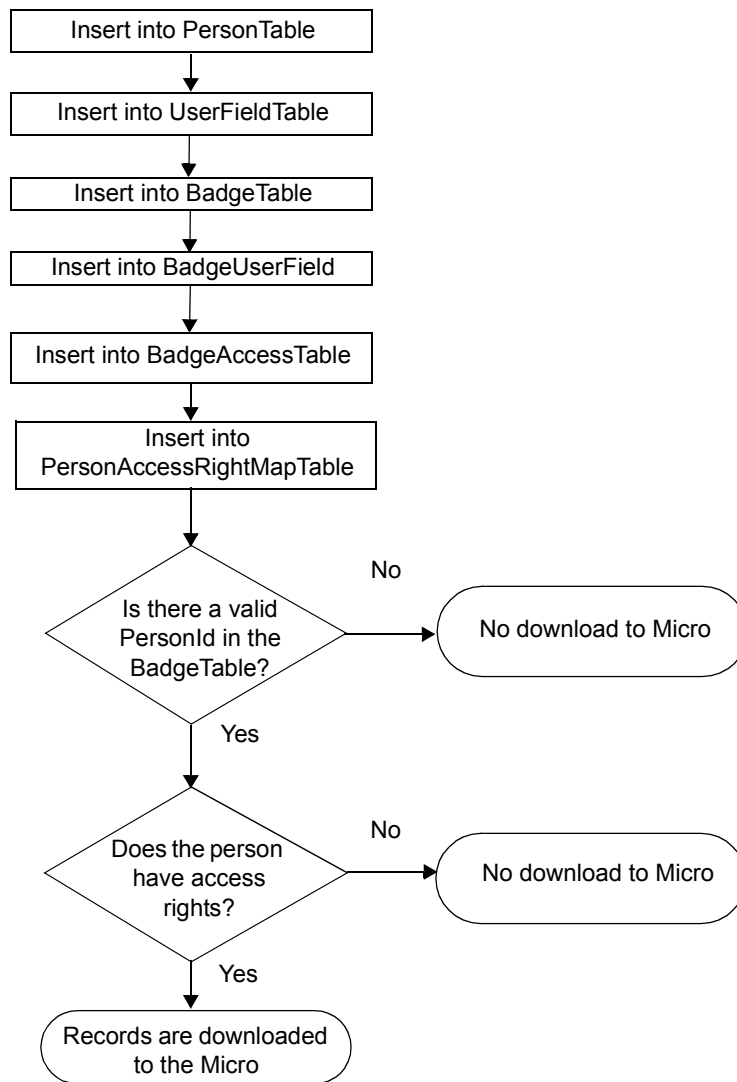
the alias number and the encoded number will be different. If aliasing is turned off, the alias number and the encoded number will be the same.

4. Insert a corresponding **BadgeAccessTable** record. Each record in the **BadgeTable** must have a corresponding record in the **BadgeAccessTable**.
5. Grant the Person access rights by inserting records into the **PersonAccessRightMapTable**. The **AccessRightId** must exist in the **AccessRightTable**. Access rights are created in Secure Perfect only, do not try to create access rights outside of Secure Perfect.

Refer to [Figure 1](#) and [Figure 2](#) for examples of importing.



**Figure 1. Flowchart - Importing Person Records Only**



**Figure 2. Flowchart - Importing Person Records and Badge Records**

## Mass Update of People and Badges

► **To mass update people and badge records:**

1. Update the records in the **PersonTable**. If you update the **DepartmentId** or **FacilityId**, make sure that the ID exists in the **DepartmentTable** and the **FacilityTable**, respectively.
2. Update the **UserFieldTable** if needed or desired. To update a **UserField** record, you will need the corresponding **PersonID** from the **PersonTable**.

- 
3. Update the badge records in the `BadgeTable` and `BadgeUserField`, if needed. If the updated record is missing a `PersonId` or an access right, there will be no download to the micro. The encoded number and the alias number cannot be updated.
  4. Update the Person's access rights by inserting records into the `PersonAccessRightMapTable`. The `AccessRightId` must exist in the `AccessRightTable`.

**NOTE**



**Do not update the `BadgeAccessTable`.**

Access rights are created only within Secure Perfect.

**CAUTION**



**Do not try to create access rights outside of the Secure Perfect system.**

## Continuous Insert or Update of Records

A continuous insert or update of records from a system outside of Secure Perfect can be accomplished by writing a custom program to read the data from the outside application, connect to the `SecurePerfect` database, and insert or update the data into the appropriate table(s).

It is recommended that the outside application and the database have some sort of commonality. For example, the employee number of the `PersonTable` in the `SecurePerfect` database matches up to some identification number from the outside application. If the outside application is a Human Resources system and Person X has been assigned the identification number 234567, then 234567 may be inserted into the `PersonTable` as the employee number.

### Continuous Insert

➤ **To insert individual records on a continuous basis:**

1. Decide how the record in the `SecurePerfect` database will be able to link to the record outside of the `SecurePerfect` database.
2. Write a program to connect to the `SecurePerfect` database and import the data. The order of operations for an import are:
  - a. Insert the record into the `PersonTable`.
  - b. Insert a corresponding record into the `UserFieldTable`.
  - c. Insert a record into the `BadgeTable` and corresponding record in the `BadgeAccessTable`.
  - d. Insert an access right into the `PersonAccessRightMapTable`, if applicable.

---

## Continuous Update

➤ **To update individual records on a continuous basis:**

1. Decide how the record in the `SecurePerfect` database will be able to link to the record outside of the `SecurePerfect` database.
2. Write a program to connect to the `SecurePerfect` database and update the data. The order of operations for an update will vary but keep the following concepts in mind:
  - a. The micro must be online for any insert, update, or delete to complete successfully.
  - b. A download to the database will occur only if a badge record has a valid `PersonId` and the corresponding person record has access rights.

If the micro is offline, Secure Perfect will route the import data to the `OfflineDownload` table. When the micro returns to online, the information will be downloaded to the micro database.

## Deleting Records from the Database

### Deleting a Person

Deleting a person from the database can be accomplished by calling the `usp_delete_person` stored procedure from the `SecurePerfect` database and passing the appropriate `PersonId`. The `usp_delete_person` stored procedure deletes the badge, access rights, as well as other records associated to the person.

**CAUTION** Do not attempt to delete a person directly from the table.



Example: `exec usp_delete_person 24` will delete the person with record ID 24 from the `PersonTable`.

### Deleting a Badge

Deleting a badge from the database can be accomplished by calling the `usp_delete_badge` stored procedure from the `SecurePerfect` database and passing the appropriate `BadgeId`. The `usp_delete_badge` stored procedure deletes the badge, access rights, as well as other records associated to the badge.

**CAUTION** Do not attempt to delete a badge directly from the table.



Example: `exec usp_delete_badge 264` will delete a badge record with record ID 264 from the `BadgeTable`.

---

## Database Connectivity

The following sections detail two methods of connectivity.

### ODBC Connection

If you use ODBC, you should have the SQL server administrator set up a dedicated login and password, and assign the login to the appropriate rights and privileges. Your own dedicated connection to the database will allow easier debugging of any application you build and more flexibility in development. **DO NOT USE** the Secure Perfect ODBC connection.

ODBC connections are configured using the **Control Panel** ODBC setup applet. ODBC connections can be made against any data source for which an ODBC driver has been installed. Visual C++ 6.0 and Visual Basic ship with drivers for Text files, Access, FoxPro, Paradox, dBase, Excel, SQL Server, and Oracle. When you create an ODBC connection, it automatically receives a Data Source Name (DSN). The DSN is subsequently used to identify connections in data-source controls, such as ADO Data Control and RDO Remote Data Control.

► **To configure an ODBC data source:**

1. For Windows 2000 Server, click on **Start**, select **Settings**, **Control Panel**, **Administrative Tools**, then **Data Sources (ODBC)**.
2. Select the **User DSN** or **System DSN** tab. The **User DSN** tab lets you create user-specific Data Source Names and the **System DSN** tab lets you create data-sources available to all users.
3. Click **Add** to display a list of locally installed ODBC drivers.
4. Select the SQL server data source for your current driver.
5. Follow the instructions specific to the driver. After closing, the DSN is now available for use.
6. You should have a valid login and password. Make the login a user of the **securePerfect** database with appropriate rights and privileges to select, insert, update, and delete on the database. You can assign the new user the db\_owner role (see SQL server books online) to give total access of the database to the new user.

### ADO Connection Example

You can connect to the **securePerfect** database via Visual Basic and ADO.

The example in [Figure 3](#) uses the Microsoft ActiveX Data Objects. Add the ADO reference from Visual Basics Project toolbar, then select References and check the ADO reference. Add a text box and a command button.

There are other methods to connect via Visual Basic, Visual C++. Additional information can be obtained in the following topics of Microsoft documentation: ADO, OLEDB, and connect. If you choose to install the samples during installation of SQL Server, the following is a helpful sample:

```
<PATH>\Microsoft SQL Server\80\Tools\DevTools\Samples\ado\vb\intro
```

---

```

Option Explicit
Dim cn As New ADODB.Connection
Private Sub Command1_Click()
    On Error GoTo ErrHandler:
    Dim UserName As String
    Dim Password As String
    Dim ServerName As String
    Dim DBName As String

    UserName = "<any valid login>"
    Password = "<password for the valid login>"
    ServerName = "<server name>\<instance name>" 'YourServerName\SPSQL
    DBName = "SecurePerfect"

    'Set connection properties.
    cn.ConnectionTimeout = 25 'Set the time out.
    cn.Provider = "sqloledb" 'Specify the OLE DB provider
    cn.Properties("Data Source").Value = ServerName 'Set SQLOLEDB connection properties.
    cn.Properties("Initial Catalog").Value = DBName 'Set SQLOLEDB connection properties.
    cn.Properties("Integrated Security").Value = "SSPI" 'Set SQLOLEDB connection properties.

    'Change mousepointer while trying to open database.
    Screen.MousePointer = vbHourglass

    'Open the database.
    cn.Open
    Screen.MousePointer = vbDefault
    Text1.Text="Connected"

Exit Sub
ErrHandler:
    'Change mousepointer back to the default after open.
    Screen.MousePointer = vbDefault
    'Display the error message.
    MsgBox Err.Description, "Error "
    'End the program.
End
End Sub

```

**Figure 3. Sample Program - ADO Connection**

---

## Troubleshooting

Use the DBtrigger diagnostic object to troubleshoot the external import, update, or delete process.

### Turning on the DBtrigger Diagnostic

Before you can capture and view diagnostic information, the appropriate diagnostic component must be turned on.

➤ **To turn on the DBtrigger diagnostic:**

1. In the Secure Perfect system, select the **Administration** menu, then **Diagnostic Settings**.
2. Click **Search** in the toolbar to display a list of components that you can monitor.
3. Select the **DBtrigger** diagnostic on the Server computer.

**NOTE**

All diagnostic objects are prefixed with a machine name.



4. Select **Enable debug messages** check box and click **Save**.
5. When you are finished troubleshooting the system, don't forget to go back and **DISABLE** debug messages.

### Viewing the Diagnostics Log

For each client, there is a default logfile (others can be created) for each day of the week such as **SPEEFriday.spl**.

DiagView operates in “real time.” To access DiagView in the Secure Perfect system, select Diagnostic Viewer from the Administration menu. That is, every time Secure Perfect writes an entry to the log file, DiagView automatically displays the latest message. By default, DiagView displays only the latest 1000 messages. The number displayed can be changed on the Preferences Form.

All log files should be saved in the logs folder; it will be easier to locate for backups and upgrades. It is a shared folder which means other clients can gain access to the log files.

➤ **To view the diagnostics log:**

1. In the Secure Perfect system, select the **Administration** menu, then **Diagnostic Viewer**.
2. Select the current day's logfile.
3. When examining the log, you will see messages similar to:

**BadgeTable record update**

This means the database trigger has fired and has successfully called the **xspDatabaseDownload** table.

---

## Printing a Report on Records Downloaded to Micros

You can print a report that lists all records (imported, updated, or deleted) that were uploaded to micros.

➤ **To print a report listing all records downloaded to micros:**

1. In the Secure Perfect system, select the **Reports** menu, then **Operator History**.
2. In the **Filter** field, enter the login name **Trigger**.

**Result:** The resulting report lists all records that were passed to the `xspDatabaseDownload.dll`.