OFXtoDB

# Overview

Most large financial institutions offer their clients the ability to generate and download account statement information, typically in either a .CSV file format or in a format used by the financial software Quicken. The .CSV formats pose many problems to be useful, but the two greatest are the lack of standardized format across financial institutions and the combining of positions and transactions into a single file, requiring an extra processing step to separate before they become useful. The Quicken format has the benefit of adhering to a standard maintained by an organization called Open Financial Exchange, or OFX, so any process that can translate one institution’s files into data should be able to do the same for all of them.

OFXtoDB is that program to read a downloaded .qfx file (Quicken format), find the data records within it, and accumulate them into a data base of records in a variety of formats. The program currently supports writing to PostgreSQL tables, EXCEL worksheets or a collection of .csv files. A major benefit is its ability to identify and eliminate duplicate records, so you do not need to be as careful about the date ranges you run or worry about whether the same file is processed multiple times.

OFXtoDB is designed to be run in ‘silent’ mode, so it is a console (command line) application that assumes itself to be a part of an automated sequence that is run over and over to capture and merge the latest statement data in a form that the user can further analyze.

Finally, the process of translating OFX data into destination data records is controlled by an external mapping specification that can reside either in the OFXtoDB.ini file or within a table in the Postgres database. The OFXtoDB.ini file that comes with the installation already has a pretty complete mapping of the INVESTMENT STATEMENT, BANKING STATEMENT and CREDIT CARD STATEMENT, so you might be able to run OFXtoDB with minimal changes to OFXtoDB.ini after the initial installation, especially to an EXCEL spreadsheet.

# Running OFXtoDB

OFXtoDB.exe is a standalone Python command-line program. It can be run at a command prompt on Windows in either cmd or PowerShell. In the same way it can be added as a step in an automated script. The format of the command line is:

OFXtoDB *[file path to OFX input]*

where the square brackets denote optional entry

If you wish to run OFXtoDB manually on Windows, a good way to do it is to drag the downloaded OFX file and drop it on top of the OFXtoDB shortcut that was installed with the distribution. The shortcut runs OFXtoDB in a cmd.exe window that won’t close after the program finishes, so you can still read the final statistics. That OFXtoDB shortcut can be copied to the desktop if desired.

OFXtoDB can also be run inside Python if you download and unpack the \*.py source files.

### Running OFXtoDB for the first time

After installing the OFXtoDB.exe there will be an OFXtoDB.ini configuration file in the same directory as the .exe file. Edit that file to provide the necessary parameters for the program to write its data. (Entries starting with # only need to be present if your value is different from the displayed default.)

|  |  |  |
| --- | --- | --- |
| OFXtoDB.ini entries to write to a PostgreSQL database: |  | OFXtoDB.ini entries to write to an Excel file: |
| [common]  TimeZone=America/New\_York  # OFXFile=*<.qfx file specification>*  Writer=Postgres  [Postgres]  # host=localhost  # port=5432  # dbname=postgres  # user=postgres  user=limited  password=plugh  schema=sandbox  mapping=UseDataBase  OFXmapping=OFXtoTables |  | [common]  TimeZone=America/New\_York  # OFXFile=*<.qfx file specification>*  Writer=Excel  [Excel]  ExcelFile=*<filepath to .xlsx file>* |

Before running for the first time, Postgres users will need to create their target tables in a single schema that they will designate and create and populate the mapping table as a database table in the same schema. See “[How To Create a Schema Sandbox in Postgresql](#_How_To_Create)” for database suggestions and “[Mapping Specification](#_Mapping_Specification_1)” for formatting the mapping table.

# OFXtoDB.ini Configuration File

Most of the program parameters are taken from the OFXtoDB.ini file which resides in the same directory as the OFXtoDB program. (On Windows it can also reside in the %APPDATA%\OFXtoDB directory.) This section documents only the basic parameters, but the .ini file can also contain the mapping specification, which is documented in the section [Mapping Specification](#_Mapping_Specification).

The file is read with Python’s ConfigParser package, so the format is very much like a Microsoft .ini file. It has sections whose names are in square brackets, followed by keyword, value pairs separated by either : (colon) or = (equal sign). Pay attention to capital letters in keywords, they must match exactly.

### Section [common]

|  |  |  |  |
| --- | --- | --- | --- |
| TimeZone | = | IANA time zone name | This is the desired time zone that all date-timestamps will be cast into before being written to the output. In the OFX standard all date fields are in date-timestamp format, with time zone, and there is considerable leeway as to which time zone is used even within a single file. This conversion ensures consistency of time-of-day treatment within your database. *(default:America/New\_York)* |
| Writer | = | Postgres or Excel or CSV | The desired output format. *(default:Postgres)* |
| OFXFile | = | File specification | The name and location of the downloaded OFX file from the financial institution. You can also specify it on the command line, just after the OFXtoDB program name and it will override anything you specify in the .ini file. |

### Section [Postgres] - Parameters specific to the Postgres writer

|  |  |  |  |
| --- | --- | --- | --- |
| host | = | Postgres hostname | The next five parameters are used to establish a database connection. *(default:localhost)* |
| port | = | Postgres listener port | *(default:5432)* |
| dbname | = | Postgres database name | *(default:postgres)* |
| user | = | Postgres user ID | *(default:postgres)* |
| password | = | Password for the user ID | For security purposes, it is assumed that the user ID is a Postgres role that has been suitably restricted to working solely within a single schema. See the section “[How To Create a Schema Sandbox in Postgresql](#_How_To_Create)” for instructions on how to set up such an ID. |
| schema | = | Schema name | All tables written by OFXtoDB must reside within this schema. Additionally, the mapping tablename specified by the OFXmapping keyword must be in this schema as well. |
| Mapping | = | UseDB or UseIniFile | The Postgres output writer can take its mapping specification from either the OFXtoDB.ini file or from a table in the database itself. UseDB is the recommended choice, because the data field formats and primary keys are read directly from the data base and do not need to be repeated in the specifications. |
| OFXmapping | = | Table name | If the UseDB option is chosen this parameter specifies the table containing the OFX-tag-to-DBColumn mapping. Such a table must have the four columns named OFXList, DBTable, OFXTag, and DBColumn and it must reside in the schema specified above. See the [Mapping Specification](#_Mapping_Specification) section for details. |
|  |  |  |  |

### Section [Excel] - Parameters specific to the Excel writer

|  |  |  |  |
| --- | --- | --- | --- |
| ExcelFile | = | File specification | Name and path of the .xlsx file to receive the OFX data. |

### Section [CSV] - Parameters specific to the Excel writer

|  |  |  |  |
| --- | --- | --- | --- |
| WriteToDirectory | = | Path specification | Path to store the .csv file(s). |
| Headers | = | YES/NO or TRUE/FALSE | Include the alpha column names as a first row. |
| WhenToQuote | = | SeparatorOnly or AllStrings | SeparatorOnly adds quotes only as needed if the string contains the column separator character. AllStrings adds quotes to all string-type data fields. *Default:* *SeparatorOnly* |
| ExcelCompatibility | = | YES/NO or TRUE/FALSE | This option should be selected if the .csv file is to be read in to EXCEL. It ensures that all string fields remain in character format after being read into EXCEL. *Default:YES* |
| QuoteChar | = | Single character | The character used to ‘quote’ the string if the ColumnSeparator character appears within it. *Default:”* [double-quote] |
| ColumnSeparator | = | Single character | The character used to separate fields in a record from each other. *Default:,* [comma] |

# Mapping Specification

It is possible to run OFXtoDB right “out of the box” using the data model provided (see the Default OFXtoDB Data Model for details). The EXCEL data writer will even create and initialize a new spreadsheet the first time it is run with worksheets and headers. You may find, however, that your financial institution provides you with information that is not in the provided mapping or that you desire a slightly different data model or table names. In that case you will need to modify the existing mapping and you will need the information in this section to do that. Before making changes to the mapping it would be helpful to familiarize yourself with the [OFX standard](https://financialdataexchange.org/common/Uploaded%20files/OFX%20files/OFX%20Banking%20Specification%20v2.3.pdf) specification. Each financial institution includes a different set of these data elements, so it is also helpful to browse an actual file created by the financial institution that operates your accounts.

Every element within a mapping consists of four fields: the OFX list name, the physical table it maps to, a specific OFX tag containing data, and the column name within the physical table it maps to. This structure provides a great deal of flexibility. You can map the same OFXList to multiple physical tables and that data will be saved to all of them. Since OFXtoDB squeezes out duplicate primary keys, you can map account summary data in a single record to an account table while writing the transactions to a transactions table. You can map different OFX tags to the same data base column, which is useful in investment transactions where internal account transactions refer to a <TOTAL> amount, but transfers into and out of the account refer to a <TRNAMT> amount.

This mapping list can reside either in a database table (recommended for database output), or in a [Mapping] section in the OFXtoDB.ini file. The database table must have as its column names OFXList, DBTable, OFXTag, and DBColumn.

What follows is a detailed description of each of the four columns and how to specify them.

### 1. OFX List Name Specification

OFXtoDB is designed to turn list-oriented data in OFX statement files into records in a database. As such, it deals with any of the following eight OFX lists:

|  |  |
| --- | --- |
| <SECLIST> | A description of the securities held in a brokerage or mutual fund account. *Primary key: <FID>, <UNIQUEID>* |
| <INVPOSLIST> | A dated list of positions (balances) of each security (Cusip) held in each brokerage or mutual fund account. The valuation date is typically the same date across an entire OFX file to enable a complete account snapshot. *Primary key: <FID>, <ACCTID>, <UNIQUEID>, <DTPRICEASOF>* |
| <INVTRANLIST> | A dated list of securities transactions between <DTSTART> and <DTEND> performed in each brokerage or mutual fund account. *Primary key: <FID>, <ACCTID>, <FITID>* |
| <BANKTRANLIST> | A dated list of transactions performed on a group of bank or credit card accounts. *Primary key: <FID>, <ACCTID>, <FITID>* |
| <BANKTRANLISTP> | A list of pending transactions waiting to be posted on a group of bank or credit card accounts. This list should always totally replace whatever was saved previously. *Primary key: <FID>, <ACCTID>, <FITID>* |
| <LOANTRANLIST> | A dated list of activity against a group of loans. *Primary key: <FID>, <ACCTID>, <FITID>* |
| <AMRTTRANLIST> | An amortization schedule of a group of loans. *Primary key: <FID>, <ACCTID>, <FITID>* |
| <CLOSING> | Statement summary (balances, fees, interest, etc.) data for a series of time periods across a group of accounts. *Primary key: <FID>, <ACCTID>, <FITID>* |

There is also useful data in a neighborhood near each of the above lists, so an OFX List name entry is actually in two parts:

|  |  |
| --- | --- |
| OFX List | Mapping List Name |
| <SECLIST> | SECLIST |
| <INVPOSLIST> | INVSTMTRS/INVPOSLIST |
| <INVTRANLIST> | INVSTMTRS/INVTRANLIST |
| <BANKTRANLIST> | STMTRS/BANKTRANLIST for bank accounts, CCSTMTRS/BANKTRANLIST for credit cards |
| <BANKTRANLISTP> | STMTRS/BANKTRANLISTP for bank accounts, CCSTMTRS/BANKTRANLISTP for credit cards |
| <LOANTRANLIST> | LOANSTMTRS/ LOANTRANLIST |
| <AMRTTRANLIST> | AMRTSTMTRS/ AMRTTRANLIST |
| <CLOSING> | STMTENDRS/ CLOSING |

Everything to the left of the last slash (/) is an XML XPath to a beginning point for processing a sublist (actually, a list of sublists if there are multiple accounts present in the file). Every data element that occurs in the file between the left part and the right part is gathered into a list and passed in to every record in the list on the right part of the specification. For example, the STMTRS/BANKTRANLIST captures everything from STMTRS down to BANKTRANLIST which includes a Currency Definition, a BANKID (route and transit number), Account Number, Account Type (Checking/Savings), an Available Balance and a Ledger Balance. Each of those gets passed in to every record in case this is desired data. If that OFX Tag is included in the mapping it is mapped, if not it is ignored.

The OFX List Name can actually be any valid XPATH entry supported by Python’s xml.etree.ElementTree package as long as it ends in one of those eight list wrapper tags. OFXtoDB prepends a .// to each OFX List Name before searching the top of the file so as to pick up each occurrence no matter where it occurs. See the section [XPath support](https://docs.python.org/3/library/xml.etree.elementtree.html#xpath-support) in the Python documentation for details on coding XPaths.

In addition, there is a small but important amount of global data contained within the <SONRS> (sign-on response) data block that includes a numeric ID of the financial institution (<FID>) and an alpha description in <ORG>. These are made available for mapping to every record processed by OFXtoDB because it’s an important part of each record’s primary key.

### 2. The Physical Table Name

The name of the target table(s) to be mapped. There can be many tables for one OFX List and there can be many OFX Lists mapping to a single table (in the case of creating a master Accounts table containing bank accounts, credit card accounts, and brokerage accounts with balances).

A Physical Table is represented differently by each output method.

In the case of a database it is the name of the table within the physical schema designated in the configuration file. Each database table must be predefined within the database manager before running OFXtoDB for the first time.

In the case of an EXCEL file it is the worksheet name within the workbook designated in the configuration file. There is no need to pre-create these worksheets before running for the first time – the EXCEL interface takes its table definitions from the config file and will create the named worksheet if it does not already exist.

In the case of a .csv file each table name produces a different file whose name is *tablename*.csv.

### 3. The OFX Tag

This can be in two forms. If the Tag name is unique within the processing neighborhood of the OFX List then it is just the Tag name (for example, MEMO is unambiguous within each OFX List even though not across lists, so only MEMO need be specified). When the Tag is ambiguous within the processing area of the OFX List then the parent Tag is included with the OFX Tag separated by a slash. For example, inside STMTRS (BANKLIST) there are two BALAMT tags containing balances. It is necessary to specify LEDGERBAL/BALAMT to map the ledger balance and AVAILBAL/BALAMT to map the available balance.

### 4. The Physical Column Name

This designates the column within the physical table to receive the data designated by the OFX Tag. The column must exist within the table specified. For databases that means this column name must match a column in the physical table (or it’s an error). There is no requirement that all columns in a physical table be mapped, leaving freedom to maintain some through triggers or generated columns. For EXCEL this column name must match both a column header on the named worksheet and a column in the [Table:] definition in the .ini file (or it’s an error). (Note that EXCEL can have extra columns not named in any mapping, perhaps with calculations in them. It is currently beyond the scope of OFXtoDB to Fill Down formulas from above with their relative addressing.)

### Table Definitions

If you are using a database destination for your data, there is no need to separately define the tables anywhere and you can ignore the instructions in this section. OFXtoDB can query the data base directly to discover such things as the desired column data format and which columns constitute the primary key for eliminating duplicate records.

If, however, you are using EXCEL for your data base output you must also define each table (worksheet) in the OFXtoDB.ini file to define each column’s desired data type and whether it is part of the primary key for the table because EXCEL has no ability to define such things itself.

Each table definition is in its own section, with the section header [Table:*tablename*]. Below it in Name, Value pairs are each of the columns, one per line. For example, the Security\_Description table (which would store <SECLIST> data) section would be:

[Table:Security\_Description]

InstID:S,PK

Cusip:S,PK

Security:S

Ticker:S

Memo:S

SecType:E

The column name is the Name, while everything after the colon is the Value. After the colon there is a mandatory data type and an optional “,PK” to designate that this column participates in the criteria that uniquely identifies this record.

Obviously, the data types specified must be compatible with the data actually supplied through the OFX standard. It is an error to specify a MEMO field as numeric, but if for some [dumb] reason you wanted all the dollar amounts to be captured as strings, you could do that here. The supported data types are:

|  |  |
| --- | --- |
| S | String type. Each character in the OFX data is preserved and written as it is received. |
| E | Enumerated type. Processed exactly as string and passed unaltered to the data writer. No attempt is made to validate the enumerated values – the data base will do so upon writing. Enumerations have been successfully used – Financial institutions follow this part of the OFX spec well. |
| N | Numeric data type. Field will be converted internally to a DECIMAL before being written out. |
| B | Boolean data type. Field will be converted to a True/False before being written out. |
| D | DateTime. Any missing time/fraction/timezone information will be added per the OFX spec, the timestamp will be converted to the desired Time Zone specified in the config file, and then stripped of its time zone information (made naïve, in Python terms). Financial institutions are massively inconsistent in how they deal with time zones, so unless you are a day trader and need that portion of the field, I recommend truncating everything to Dates and forgetting the time portion. |
| DATE | Date type. Processed exactly as above, but the time portion is zeroed out as a last step, to be compatible with EXCEL’s date type, where the time fraction is always present and is zero. |

# Special OFX Exceptions

For the most part, OFXtoDB will process whatever appears within the OFX lists in the relevant sections of the files and pass them to the mapper to turn it into a data record according to the mapping specifications. There are a few exceptions worth noting.

1. The data in the <SONRS> aggregate is considered to be global information and the data that is extracted from it gets handed to the mapper for every record that is assembled during the run. Most of the data within it is control information and pertains to the circumstances around generating the file, not data about the accounts themselves. Because of that, the only elements that are extracted are the ones in the <FI> aggregate: <ORG> and <FID>. <ORG> is an alpha description of the financial institution, while <FID> is a numeric institution identifier. You should use the <FID> as part of the primary key for all your tables thereby allowing all of your accounts at all institutions to be co-resident within the same tables without fear of conflict.
2. The various xxxTRANLIST list wrappers contain, in addition to the transactions themselves, two single-element entries named <DTSTART> and <DTEND>, describing (with TimeStamps) the time period covered by that list. OFXtoDB elevates those two entries, when they appear, out of the list and instead are treated as context elements that are handed to the mapper for each record processed within that list.
3. The <INVTRANLIST> is unique among OFX lists in that it has a variety of different child elements containing transaction data. These element names identify the type of transaction, and should be thought of as a sort of transaction type (or subtype). The different names include INBANKTRAN, BUYDEBT, BUYMF, BUYOPT, BUYOTHER, BUYSTOCK, CLOSUREOPT, INCOME, INVEXPENSE, JRNLFUND, JRNLSEC, MARGININTEREST, REINVEST, RETOFCAP, SELLDEBT, SELLMF, SELLOPT, SELLOTHER, SELLSTOCK, SPLIT, TRANSFER. OFXtoDB presents these different values to the mapper as-is but under a made-up pseudo-tag called ELEMENTNAME. This allows the transaction type/subtype to be stored within each record. It can either be treated as a VarChar (string) data type or as an enumeration.  
     
   In addition, this long list of transactions is further summarized into an enumerated data type with the values BUY, SELL, TRANSFER, INCOME, FEES, and OTHER and passed to the mapper under another pseudo-tag named BUYSELL. This is intended to make portfolio analysis a little easier by storing BUYSELL into a TranType field and ELEMENTNAME into a TranSubType field.

# The Downloaded Data Files

The files that are generated in Quicken format and downloaded from financial institutions have a .qfx file type. It has a short header and the remainder is OFX data, starting with <OFX>. Here is an excerpt from one (it has been made more readable with newlines, indentation, and closing tags):

OFXHEADER:100

DATA:OFXSGML

VERSION:102

SECURITY:NONE

ENCODING:USASCII

CHARSET:1252

COMPRESSION:NONE

OLDFILEUID:NONE

NEWFILEUID:NONE

<OFX>

<INVSTMTMSGSRSV1>

<INVSTMTTRNRS>

<TRNUID>0</TRNUID>

<STATUS>

<CODE>0</CODE>

<SEVERITY>INFO</SEVERITY>

</STATUS>

<INVSTMTRS>

<DTASOF>20230811160000.000[-5:EST]</DTASOF>

<CURDEF>USD</CURDEF>

<INVACCTFROM>

<BROKERID>vanguard.com</BROKERID>

<ACCTID>12345678</ACCTID>

</INVACCTFROM>

<INVTRANLIST>

<DTSTART>20230511160000.000[-5:EST]</DTSTART>

<DTEND>20230811005831.000[-5:EST]</DTEND>

<REINVEST>

<INVTRAN>

<FITID>12345678</FITID>

<DTTRADE>20230622160000.000[-5:EST]</DTTRADE>

<DTSETTLE>20230622160000.000[-5:EST]</DTSETTLE>

<MEMO>DIVIDEND REINVESTMENTDIVIDEND REINVESTMENT</MEMO>

</INVTRAN>

<SECID>

<UNIQUEID>12345A789</UNIQUEID>

<UNIQUEIDTYPE>CUSIP</UNIQUEIDTYPE>

</SECID>

<INCOMETYPE>DIV</INCOMETYPE>

<TOTAL>-882.51</TOTAL>

<SUBACCTSEC>CASH</SUBACCTSEC>

<UNITS>8.365</UNITS>

<UNITPRICE>105.5</UNITPRICE>

</REINVEST>

<REINVEST>

<INVTRAN>

<FITID>987654321</FITID>

<DTTRADE>20230531160000.000[-5:EST]</DTTRADE>

<DTSETTLE>20230531160000.000[-5:EST]</DTSETTLE>

<MEMO>DIVIDEND REINVESTMENTDIVIDEND REINVESTMENT</MEMO>

</INVTRAN>

<SECID>

<UNIQUEID>54321X987</UNIQUEID>

<UNIQUEIDTYPE>CUSIP</UNIQUEIDTYPE>

</SECID>

<INCOMETYPE>DIV</INCOMETYPE>

<TOTAL>-2.89</TOTAL>

<SUBACCTSEC>CASH</SUBACCTSEC>

<UNITS>2.89</UNITS>

<UNITPRICE>1.0</UNITPRICE>

</REINVEST>

You can discover what data your own financial institution provides by examining the file itself and reading about what each of the tags mean in the [OFX standard](https://financialdataexchange.org/common/Uploaded%20files/OFX%20files/OFX%20Banking%20Specification%20v2.3.pdf). If you decide to add a field to the data you want to capture, you can do that by:

|  |  |
| --- | --- |
| PostgreSQL | Excel |
| 1. Add a column to the data base table in Postgresql | 1. In the Excel file select the worksheet with the correct table name. Insert a new column or add to the end. In the first row, give a column name label for the column (no spaces). |
| 2. In the OFX mapping table, add a row with the correct OFX List name and tablename plus the OFX Tag where the data resides and the data base column name you added in step 1. | 2. In the OFXtoDB.ini file in the [Mapping] section scroll down to the correct OFX List and table name and add a line there with the same OFX List and tablename plus the OFX Tag with the data and the column name you added above. |
|  | 3. Also in the OFXtoDB.ini file find the Table: section with the correct table name and add a line with the column name from the previous steps and its desired data type, separated by colon (:). It is a very good idea to keep the column names here in the same order they appear in the Excel file columns. |

HINT: OFX files become much easier to read after converting them from SGML to XML. This can be done in the NotePad++ editor by:

1. Adding a newline just before the <OFX> tag to separate the Quicken header from the OFX data.

2. Adding close tags to all the elements with missing close tags. Under Search..Replace select a Regular Expression replace – In the ‘Find what’ box enter <(?!/)([^>]+>)([^<]+)(?!</\1) and in the ‘Replace with’ box enter <$1$2</$1. Press the Replace All button to execute.

3. On the Plugins menu make sure you have installed the XML Tools plugin, then from that same menu select “XML Tools>” and then select Pretty Print – Indent Attributes

# Data Model Provided In The Distribution



BEWARE: If using any database writer, do not be tempted to use a sequence as your tables’ primary keys. You will defeat the duplicate record checking and risk adding the same record multiple times into your tables.

# How To Create a Schema Sandbox in Postgresql

When adding data to a PostgreSQL data base under program control the first step is to establish a data base session by logging in under a particular user and password. This causes a security vulnerability because at some point in its execution the program must know the password to set up the session and a malicious actor could obtain it no matter how well the program tries to obfuscate it. Rather than try to hide that password, the OFXtoDB program relies on limiting the damage that a hacker can do by confining all table writes to a single schema and then confining the program's database userid to only those activities within that schema necessary to write its data there (UPDATE, INSERT and SELECT). With these controls in place a malicious user can still use that ID to steal your financial data or fill your disk up with garbage (denial of service), but they cannot affect the rest of the data base. Those possibilities can be further mitigated by ensuring your PostgreSQL instance only accepts sessions from localhost if all activity is confined to the one PC, or making a router rule to only make db connections if the IP address starts with 192.168 or 10 if you need to allow data base action across your home network.

This section explains how to set up just such a "sandboxed" ID in PostgreSQL. The GRANTs needed to make this work are a little tricky to get right, so this document will explain it in a step-by-step fashion.

The most important part of this solution is to make all the tables be owned by the limited user, because without it the OFXtoDB program will not be able to determine the target tables' primary keys which is necessary to prevent duplicate financial records from being stored in the tables. To do this, we will refer to 2 user IDs, 'postgres' and 'limited', where postgres is the standard superuser ID and limited will be the limited-access ID to be used by the OFXtoDB program in storing its records. In addition, we will assume the target schema to store this data will be named 'sandbox'.

Step one is to create the limited user and the sandbox schema. Log into a session with the postgres (superuser) ID. The user ID creation is very straightforward, and you can either issue the SQL to do it or do it from the menus in PGADMIN4. To create the sandbox schema, issue the following SQL:

CREATE SCHEMA sandbox;

GRANT USAGE, CREATE On SCHEMA sandbox to limited

GRANT ALL On ALL TABLES IN SCHEMA sandbox to limited

The program OFXtoDB also reads descriptive information about the tables that is stored in information\_schema and pg\_catalog, so you may need to GRANT SELECT On ALL TABLES in those schemas as well, although I did not. It appears those schemas come pre-made with SELECT grants to public so it might be unnecessary.

The trick here is to understand the two-tier permission system in PostgreSQL. Coming from Oracle, the second of the two Grants above is all you ever need to give access and new users to PostgreSQL don't understand how you can GRANT SELECT on a table successfully and then be prevented by permissions from SELECTing on it. The answer is that first PostgreSQL checks your access to the schema and THEN checks for appropriate access to the table (it then goes on to check access to the columns, if any). Without access to the schema, no access to the tables. The GRANT USAGE, CREATE On SCHEMA gives both a USAGE permission which allows the user to look up tables in the schema, and a CREATE permission which allows the creation of new objects (tables, views, TYPEs, functions, etc.) in the schema. You need both to set up the tables, but we will later REVOKE the CREATE permission when we finish locking down the userid. We are temporarily GRANTing ALL permissions to limited, because a user under that ID/role must be the one to create the data base tables which will house the data.

After designing the data base, physical tables, and the primary keys, you will create the actual tables. Log out of the postgres session and log into a limited session. You can create the tables with SQL or with the PGADMIN4 interface. The first important point is to create the primary keys while you are defining the data columns - the Postgres data writer in OFXtoDB reads the primary keys to understand how to de-dup records already in the data base as well as repeated records within each data feed (Separate SECLISTs for multiple accounts is a prime creator of duplicate securities). Do not be tempted to use a sequence for the primary key! You will defeat that important de-duping capability. The second point here is if you are going to use enums as column types, you must create the enum as a Type before CREATing the table - each enum is stored at the schema level, not at the table level.

The next step is to create the OFX mapping table in this same schema and populate it with the List⇨table, tag⇨column mappings (also under the limited userid). The OFX mapping table can be called whatever you wish (its name is designated in the OFXtoDB.ini file), but its four column names must be OFXList, DBTable, OFXTag, and DBColumn, and the mixed case is important (use double quotes around the column names). Here you can either leave out the primary key entirely or make all 4 columns the primary key. See the topic [Mapping Specification](#_Mapping_Specification) for details on how to code a mapping.

Now you're ready to take your new data base for a test drive. Create your OFXtoDB.ini file in the same directory as the program with the following entries (change these values to suit your customizations):

[common]

TimeZone = America/New\_York

# OFXFile = <.qfx file specification>

Writer = Postgres

[Postgres]

# host=localhost

# port=5432

# dbname=postgres

# user=postgres

user=limited

password=plugh

schema=sandbox

mapping = UseDataBase

OFXmapping = OFX\_to\_Tables

See the topic OFXtoDB.ini parameters for more details on any of these. Run the program from the command line as "OFXtoDB [.qfx file spec]". Either include the .qfx file on the command line or embed it in the .ini under the OFXFile parameter. If there's something wrong, such as trying to save an alpha data field into a numeric format, the program will be pretty specific about its error. Otherwise, the program finishes normally with some data table statistics like:

|  |  |  |
| --- | --- | --- |
|  | Added | Updated |
|  | New | Existing |
| Accounts | 0 | 1 |
| Account\_Bal\_History | 0 | 1 |
| Bank\_Transactions | 0 | 144 |

The most likely problems are misspellings of tag or column names in the mapping, and these will not produce an error. They simply won't store the data as expected. So, check your .qfx file carefully. If you see data that you think you mapped but it's not getting stored in the table, check the mapping table entry for typos.

Once you are satisfied that the mappings are working, it is time to lock down the permissions for limited. Log into the postgres superuser id and issue the following DDL commands:

Revoke CREATE On SCHEMA "sandbox" From limited;

Revoke ALL On ALL TABLES IN SCHEMA "sandbox" From limited;

Grant SELECT, INSERT, UPDATE On ALL TABLES IN SCHEMA "sandbox" to limited;

Grant Delete on table "sandbox"."OFX\_to\_Tables" to limited;

This last Grant is a convenience to allow full data maintenance on the mapping table itself, but you can omit it and maintain it from a superuser session.