PXPLUS EXTERNAL DATABASE SUPPORT

PXPLUS EXTERNAL DATABASE SUPPORT

- While most aspects of a business application can be served within the PxPlus family of products, today's end users are often required to work with data that resides in completely different software worlds
  - Businesses may need to integrate popular "off-the-shelf" software with their legacy systems, applications and databases
- PxPlus can work directly with data in a number of **External Databases,** over networks and on completely different operating systems
  - Any external database with an ODBC driver via [ODB]
  - MySQL/MariaDB via [MYSQL]
    - Requires MySQL/MariaDB C Connector: libmysql.dll
  - Microsoft SQL Server via [ADO]
  - Oracle Server via [OCI]
    - Requires Oracle instant client: oci.dll
  - IBM DB2 via [DB2]
    - Requires DB2 CLI client: db2cli.dll

PXPLUS EXTERNAL DATABASE SUPPORT

- An External Database table is accessed via an **OPEN** directive just like a PxPlus native file
  - Use the **[xxx]** prefix depending on the type of database
  - The database connection information and table name are given either after the **[xxx]** prefix or via **OPT=**
    - Check the documentation of the specific database prefix you want to use for the exact syntax

```
open(hfn,iol=*,OPT="SERVER=192.168.1.114;PORT=3306;USER=xxxx;PSWD=xxxxxxx")"[MYSQL]test_db;invoice_header"
```

- For Read-only data that is not too large, dramatically improve performance with **OPEN LOAD** directive
  - Locally caches the whole table in memory
  - Controlled system wide via the **'CL'=value** system parameter
    - All tables with a record count below value will be cached, 0 will disable **OPEN LOAD** caching, default is 1000
  - Individually controlled via **OPEN LOAD** directive when **OPT=** clause of **CACHE=value** is included, overriding **'CL'**
    - A value of *yes* means to always cache, *no* means to never cache, *nnn* means to cache if record count is less than *nnn*

```
open load(hfn,iol=*,OPT="SERVER=192.168.1.114;PORT=3306;USER=xxxx;PSWD=xxxx;cache=yes")"[MYSQL]test_db;invoice_header"
```

PREFIX FILE

PREFIX FILE

- Can define a new prefix file entry with the database connection information
  - Code can be kept cleaner or be switched from native file to a database without changing code
  - The prefix file is a PxPlus native variable length keyed file
  - Prefix file entry/record
    - The key is the name you want to access your database table by
    - The first field will contain the database prefix and the DSN/Table declaration
    - The second field will contain the database connection options usually in the **OPT=**
    - The third field may be specified that contains the IOList to use when opening the file with an **IOL=*** option

```
KEYED "PFXFILE",127
OPEN (1) "PFXFILE"
WRITE(1,KEY="my_table")"[MYSQL]test_db;my_table;SERVER=192.168.1.114;PORT=3306;USER=xxxx;PSWD=xxxxxxx",
"KEY=field1;REC=field1:10,field2:8.2,field3:40,field4:8,field5:2,field6:20",
"field1$,field2,field3$,field4$,field5,field6$"
CLOSE (1)
```

PREFIX FILE

- Use the **PREFIX FILE** directive to start using a prefix file
  - Once setup any open using the name key will actually open the database connection defined in the prefix entry

```
PREFIX FILE "PFXFILE"
OPEN (1,iol=*) "my_table"
```

LINK FILE

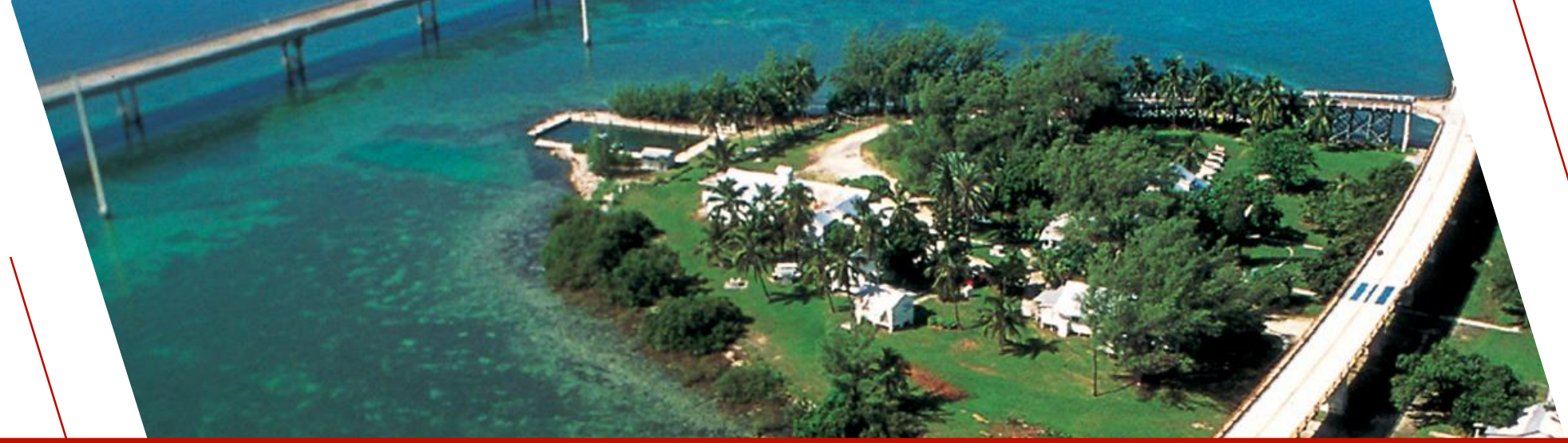
LINK FILE

- Can define a link file with the database connection information
  - Code can be kept cleaner or be switched from native file to a database without changing code
  - An external database link file is a [Pvxdev] **Link File** that points to a device driver, ***dev/extdb**
  - Link File Format:
    - Line 1: Typical line for a [Pvxdev] link file, pointing to the device driver, *dev/extdb (256 characters)
    - Line 2: Database Type
    - Line 3: Database Name
    - Line 4: Table Name
    - Line 5 (and higher): Options
  - Any line in the link file (after the first line) that starts with an **=** (equals sign) will be evaluated
    - This can be used to avoid plain text password in the link file
      - ="USER="+%adoUser$
      - ="PSWD="+%adoPswd$
  - The **IOL=** and **OPT=** on the **OPEN** of the link file will be used
    - Any OPT= on the OPEN of the link file overrides the OPT= defined in the link file

LINK FILE

```
[Pvxdev].                                    extdb
ADO
ServerName
TableName
DB=databaseName
NONULLS=YES
Connect='Provider=SQLOLEDB;'
EXTROPT=(UPDLOCK)
DATEFMT=YYYYMMDD
KEY=fieldOne,*NAME:KeyOne
KEY=fieldTwo,fieldThree,*NAME:KeyTwo
REC=fieldOne:12,fieldTwo:40,fieldThree:6.2,fieldFour:6.0
```

- Suppose that you created a link file called "ADOProduct" that defined an ADO connection to the Product table
  - OPEN (chan,IOL=*)"ADOProduct"
- This will then make a connection to the external database as defined in the link file
- It would be as if "ADOProduct" was a PxPlus data file to your program even though it is an external database table



USING EXTERNAL DATABASES

USING EXTERNAL DATABASES

- Existing code using native PxPlus files will work without changes
- Queries, Reports, File Maintenance, and Webster+ equivalents will work without changes
- **READ**, **WRITE**, **INSERT**, **UPDATE** and **REMOVE** directives will generate the SQL to work with the database automatically
  - Can generate better optimized SQL if the keys are defined to match the tables index fields

PxPlus Code		SQL Code
open (1)"[ODB]MYDB;CUSTOMER" read record (1)R$	→	SELECT * FROM CUSTOMER
open (1)"[ODB]MYDB;CUSTOMER;KEY=CST_ID" read record (1,key= "00420000")R$	→	SELECT * FROM CUSTOMER WHERE CST_ID = "00420000"

USING EXTERNAL DATABASES

- **SELECT**, which allows you to use SQL-like syntax in your programs to access native files, also will convert into real SQL to work with the database automatically
  - A **WHERE** clause in the **SELECT** directive can optimize the SQL generated
    - Only if comparing field variable against literal

```
SELECT * FROM "[ODB]MYDB;CUSTOMER"
WHERE  State$ = "ON"
```

```
SELECT * FROM CUSTOMER WHERE State = 'ON'
```

    - Use **STATIC** clause to optimize if comparing field variable to a variable

```
SELECT * FROM "[ODB]MYDB;CUSTOMER"
STATIC WHERE  State$ = curState$
```

USING EXTERNAL DATABASES

- You can enable a debug mode where the SQL commands generated will be displayed via a msgbox
  - SET_PARAM '!Q'
- Directly execute SQL on the database
  - Use a **key="!<SQL>"** on a **READ** directive
  - Read Record also supported
  - Table name from connection ignored

```
open (1)"[ODB]MYDB;some_table"
sqlcmd$="SELECT first_name FROM students WHERE student_id IN (SELECT student_id FROM grades WHERE grade = 'A')"
read record (1, key="!"+sqlcmd$) result$
```

USING EXTERNAL DATABASES

- Open a direct connection to the database by **not** specifying a table name
  - If connection kept open can be used to avoid needing login credentials on each database connection
  - Can query the following with **key=**

KEY=	Action	SQL Function
"?"	Returns the list of table, catalog, or schema names, and table types	SQLTables()
"*xxxx"	Returns the list of column names in table *xxxx*	SQLColumns()
"**xxxx"	Returns a list of statistics about table *xxxx* and the indexes associated with the table	SQLStatistics()

```
open (chan)"[ODB]MYDB;"
read (chan, key="?“) tableCatalog$,tableSchema$,TableName$,tableType$,Remarks$
while 1
read (chan, err=*break) tableCatalog$,tableSchema$,TableName$,tableType$,Remarks$
wend
```

USING EXTERNAL DATABASES

- Open a direct connection to the database by **not** specifying a table name
  - **WRITE RECORD** directive allows you to execute SQL directly while the **READ RECORD** directive returns the results

```
open (chan)"[ODB]MYDB"
sqlcmd$="SELECT first_name FROM students WHERE student_id IN (SELECT student_id FROM grades WHERE grade = 'A')"
write record (chan) sqlcmd$
read record (chan) result$
```

  - Same **key="!<SQL>"** on a **READ** directive works here too