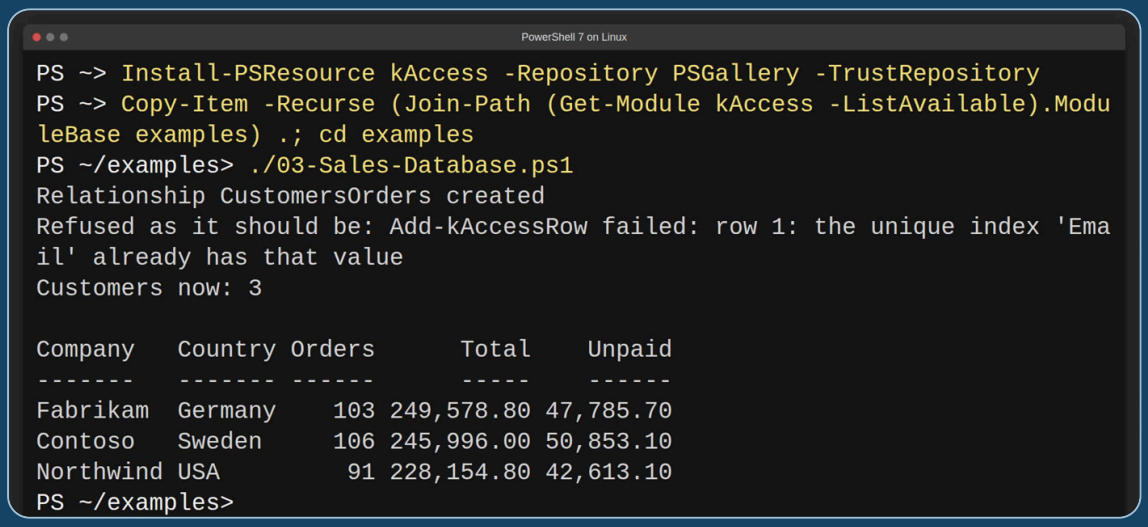
kalmstrom.com Business Solutions

kAccess

User Guide

Read and write Microsoft Access databases from PowerShell 7 on Windows, Linux and macOS – with no Access, no driver and no Java.

.accdb | .mdb | Windows | Linux | macOS | Free, MIT licence



Module version 1.0.0.10 · PowerShell 7.2 or later · Published November 2026

Contents

1 Introduction 3

2 Background: PowerShell 7 and Access files 5

3 Installation 11

4 Your first database 16

5 Tables, columns and indexes 20

6 Working with rows 25

7 Relationships and saved queries 29

8 Attachments, multi values and encrypted files 35

9 The seven examples 39

10 Command reference 53

11 Troubleshooting and limits 60

Appendix AQuick reference 64

Appendix BGlossary 66

CHAPTER 1

Introduction

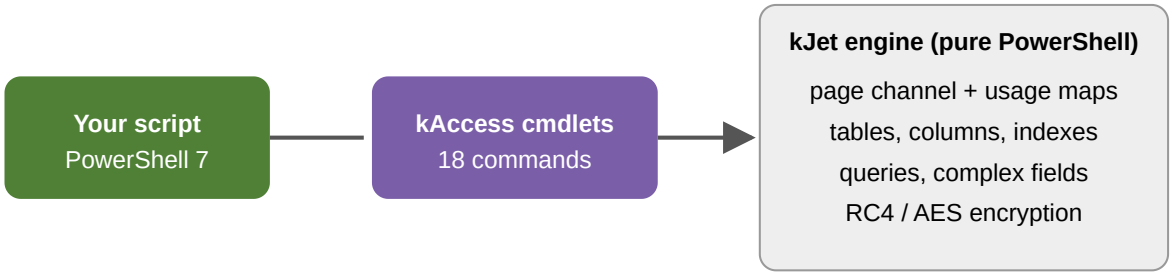
> kAccess lets a PowerShell script open, create and change Microsoft Access databases (`.mdb` and `.accdb`) on Windows, Linux and macOS - without Access, without the ACE/ODBC driver and without Java. A file that used to need a Windows desktop with Office installed can now be read on a build server, in a Linux container or on a Mac.

1.1 What kAccess does

Many companies still keep important data in Access files: order books, inventories, member lists. Getting that data into a report, a CSV export or a cloud system has always meant a Windows PC with the right driver bitness. kAccess reads and writes the file format directly, so the same script runs anywhere PowerShell 7 runs.

- **Read** tables, rows, indexes, saved queries (as SQL), attachments and multi-value fields.
- **Write** new databases, tables with AutoNumber keys and indexes, relationships, rows in bulk, saved queries.
- **Change** and delete rows straight from the pipeline; rename and drop tables.
- **Encrypted files** are read and written with `-Password` on every command that opens a file.

1.2 How it works



No Office, no ACE/ODBC driver, no Java - the .accdb / .mdb bytes are read and written directly

The engine is a PowerShell port of **Jackcess**, the well-tested open source Java library for the Access format. It understands the page layout of the Jet (Access 2000-2003) and ACE (Access 2007-2019) formats, updates indexes when rows change, and writes files that Access opens without complaint.

1.3 Who it is for

Reader	Typical use
IT administrators	Inventory all Access files on a share, export them, move data to SharePoint or SQL
Developers	Build test databases in CI, read customer files on Linux servers
Data analysts	Import CSV into Access, produce reports from Access data on any OS

1.4 Requirements

Item	Requirement
PowerShell	7.2 or later (Windows, Linux, macOS)
Microsoft Access	**Not needed**
ACE / ODBC driver	**Not needed**
File formats	Access 2000, 2002-2003, 2007, 2010, 2016, 2019 (.mdb and .accdb)

CHAPTER 2

Background: PowerShell 7 and Access files

> This chapter gives the background that makes the rest of the guide easy: what PowerShell 7 is, the five PowerShell ideas every kAccess command uses, and the Access file formats. If you already script in PowerShell 7 every day, you can go straight to Chapter 3.

2.1 PowerShell 7 in brief

PowerShell is Microsoft's command shell and scripting language. There are two editions, and kAccess needs the newer one:

	Windows PowerShell 5.1	PowerShell 7
Program	`powershell.exe`	`pwsh` (`pwsh.exe` on Windows)
Runs on	Windows only	Windows, Linux and macOS
Built on	.NET Framework 4	.NET 8 and later
Status	ships with Windows, no new features	open source, actively developed
kAccess	**Not supported**	**7.2 or later**

Both editions can live side by side on one Windows computer. Start PowerShell 7 by typing `pwsh` in any terminal; the prompt then shows `PS` and the current folder, as in the screenshots of this guide.

Install PowerShell 7

System	Command
Windows 10/11, Windows Server	`winget install Microsoft.PowerShell`
Ubuntu, Debian and other Linux	`sudo snap install powershell --classic`, or the Microsoft package repository
macOS	`brew install --cask powershell`
Docker	the image `mcr.microsoft.com/powershell`

Check the version with `$PSVersionTable.PSVersion`. PowerShell 7.4 and later already contain the `Install-PSResource` command used in Chapter 3; on 7.2 and 7.3, `Install-Module` does the same job.

2.2 Five PowerShell ideas kAccess builds on

1. Commands are named Verb-Noun

Every PowerShell command says what it does: `Get-` reads, `New-` creates, `Add-`, `Set-` and `Remove-` change. kAccess follows the same rule, and its nouns all start with `kAccess`: `Get-kAccessRow`, `New-kAccessTable` and so on. When you know one command, you can guess the others.

2. The pipeline passes objects, not text

The `|` character sends the output of one command into the next. In PowerShell the output is *objects* with named properties, so the next command can filter on a column without any text parsing:




```
Get-kAccessRow ./Sales.accdb Orders | Where-Object Amount -gt 1000 |
    Sort-Object Amount -Descending | Export-Csv ./big-orders.csv
```

The same pipeline carries rows *into* kAccess: rows piped to `Add-kAccessRow`, `Set-kAccessRow` or `Remove-kAccessRow` are collected and written together, which is what makes bulk changes fast.

3. A hashtable describes a row or a column

A hashtable is a set of `Name = Value` pairs in `@{ }`. kAccess uses one hashtable per row to add, and one per column or index to create:

```
@{ Company = 'Contoso'; Country = 'Sweden' }              # a row
@{ Name = 'Company'; Type = 'TEXT'; Length = 80 }        # a column
@{ Name = 'PrimaryKey'; Columns = @('Id'); PrimaryKey = $true }   # an index
```

4. Modules come from the PowerShell Gallery

A module is a package of commands. The PowerShell Gallery (`powershellgallery.com`) is Microsoft's public store for them, and kAccess is published there. One command downloads and installs it; `Import-Module kAccess` loads it, and PowerShell also loads it by itself the first time you use one of its commands.

5. Help, -WhatIf and -Confirm are built in

`Get-Help New-kAccessTable -Full` shows the help of any kAccess command, with examples. Commands that delete or change data – `Set-kAccessRow` , `Remove-kAccessRow` , `Remove-kAccessTable` and the `Remove-` commands for attachments and multi values – support `-WhatIf` , which shows what would happen without writing anything.

2.3 The Access file formats

Microsoft Access stores a whole database – tables, indexes, relationships, queries, forms and code – in one file. The database engine inside Access was called **Jet** until Access 2003 and **ACE** from Access 2007; the file format followed the engine. kAccess reads all of them and creates new files in six formats:

`-Format`	File	Engine	Opens in	Notes
`V2000`	.mdb	Jet 4	Access 2000 and later	for old applications
`V2003`	.mdb	Jet 4	Access 2002 and later	the last .mdb format
`V2007`	.accdb	ACE 12	Access 2007 and later	attachments and multi values
`V2010`	.accdb	ACE 14	Access 2010 and later	**default**
`V2016`	.accdb	ACE 16	Access 2016 and later	
`V2019`	.accdb	ACE 16	Access 2019 / Microsoft 365	Date/Time Extended columns

If you do not know which to pick, keep the default. A new `.accdb` in the `V2010` format opens in every Access version in use today. Choose `V2000` only when an old program needs an `.mdb` file.

2.4 Why scripts needed Access before

Until now a script that read an Access file went through the **Access Database Engine** (the ACE OLE DB / ODBC driver). That works, but it brings well-known problems:

- **Windows only:** there is no ACE driver for Linux or macOS, so the job cannot move to a Linux server or a container.
- **Bitness:** the 64-bit driver cannot be installed beside a 32-bit Office, and the script must match the driver.
- **Installation rights:** a locked-down server or a cloud runner often cannot install Office components at all.
- **Licensing and support:** Microsoft does not support the engine as a server component.

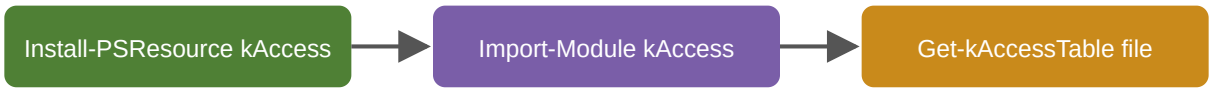
kAccess avoids all four, because it is plain PowerShell code that understands the file format itself. Other driver-free options exist on Linux, such as the mdbtools package for reading or Jackcess in Java; kAccess brings the reading *and* writing to PowerShell with no runtime besides PowerShell 7.

> **Tip:** kAccess and Access can be used on the same file, one after the other. Close the database in Access before a script writes to it, as you would with any other program that changes the file.

CHAPTER 3

Installation

> kAccess installs from the PowerShell Gallery with one command and takes a few seconds. Nothing else is installed: no driver, no runtime and no service.



3.1 What you need

- **PowerShell 7.2 or later** on Windows, Linux or macOS (Chapter 2.1 shows how to install it).
- **Internet access to the PowerShell Gallery** for the installation, or a copy of the module made on another computer (see 3.5).
- **Read access** to the Access files you want to read, and **write access** to those you want to change.

Microsoft Access, the Access Database Engine, ODBC and Java are *not* needed.

3.2 Install the module

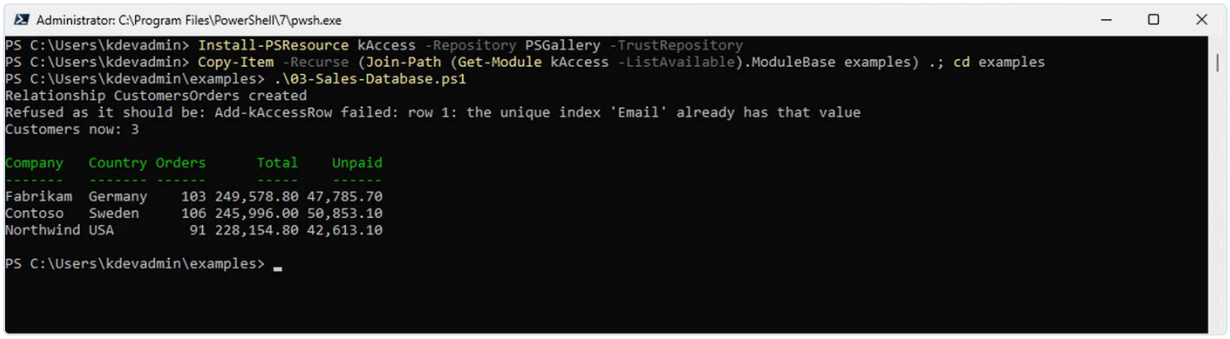
1. Open PowerShell 7 (type `pwsh` in a terminal).
2. Run the install command:

   ```
   Install-PSResource kAccess -Repository PSGallery -TrustRepository
   ```

   On PowerShell 7.2 and 7.3, use `Install-Module kAccess` instead and answer **Y** when you are asked to trust the gallery.
3. Check that PowerShell finds the module and its 18 commands:

   ```
   Get-Module kAccess -ListAvailable
   Get-Command -Module kAccess
   ```

The module is installed for the current user, so no administrator rights are needed. Add `-Scope AllUsers` in an elevated session to install it for every user of the computer, for example on a server that runs scheduled tasks under a service account.



PowerShell 7 on Windows: install, copy the examples, run one – three lines

3.3 A successful installation

After the installation, the two check commands answer like this. The version number tells you which release you have; this guide describes version 1.0.0.10.

PowerShell 7

```
PS ~/examples> ./tour-version.ps1
Name    Version
----    -------
kAccess 1.0.0.9


Add-kAccessAttachment                          Add-kAccessMultiValue
Add-kAccessRow                                 Get-kAccessAttachment
Get-kAccessMultiValue                          Get-kAccessQuery
Get-kAccessRow                                 Get-kAccessTable
New-kAccessDatabase                            New-kAccessQuery
New-kAccessRelationship                        New-kAccessTable
Remove-kAccessAttachment                       Remove-kAccessMultiValue
Remove-kAccessRow                              Remove-kAccessTable
Rename-kAccessTable                            Set-kAccessRow
```

3.4 Copy the examples

The module ships seven complete example scripts, described in Chapter 9. Copy them to a folder of your own before you run or change them, so an update of the module never overwrites your changes:

```
Copy-Item -Recurse (Join-Path (Get-Module kAccess -ListAvailable).ModuleBase
examples) .
cd examples
./01-Quick-Start.ps1
```

3.5 Install on a computer without internet

1. On a computer with internet access, save the module to a folder:

   ```
   Save-PSResource kAccess -Repository PSGallery -Path ./offline
   ```

2. Copy the `kAccess` folder from `./offline` to the other computer, into one of the folders in `$env:PSModulePath`.

3. Run `Get-Module kAccess -ListAvailable` there to check it.

3.6 Update and remove

To	Run
Update to the newest version	`Update-PSResource kAccess` (or `Update-Module kAccess`)
See the installed versions	`Get-InstalledPSResource kAccess`
Remove the module	`Uninstall-PSResource kAccess` (or `Uninstall-Module kAccess`)

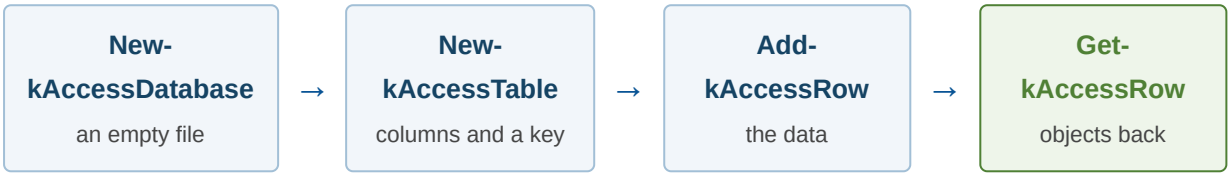
The release notes of each version are on the module's page in the PowerShell Gallery, under "Release Notes".

> **In a script or a scheduled job,** start with `Import-Module kAccess`. It makes the dependency visible at the top of the script, and a missing module is reported on the first line instead of in the middle of the job.

CHAPTER 4

Your first database

> In five minutes you create an Access database, give it a table with an AutoNumber key, add two contacts and read them back. Every step is one command, and the result is an ordinary `.accdb` file.



4.1 Create the database

```
New-kAccessDatabase ./Contacts.accdb
```

The command creates an empty Access 2010 database and answers with the new file. It never overwrites a file that exists. Add `-Format V2000` for an `.mdb` file (see Chapter 2.3).

4.2 Create a table

```
New-kAccessTable ./Contacts.accdb Contacts -Column `
    @{ Name = 'Id';    Type = 'LONG'; AutoNumber = $true },
    @{ Name = 'Name';  Type = 'TEXT'; Length = 60 },
    @{ Name = 'Email'; Type = 'TEXT'; Length = 120 },
    @{ Name = 'Added'; Type = 'SHORT_DATE_TIME' } `
    -Index @{ Name = 'PrimaryKey'; Columns = @('Id'); PrimaryKey = $true }
```

Each column is a hashtable with a name and a type. `AutoNumber` makes Access number the rows 1, 2, 3 and so on, and the index named `PrimaryKey` makes `Id` the primary key – Access shows the key symbol beside it in Design View.

4.3 Add rows

```
Add-kAccessRow ./Contacts.accdb Contacts -Row `
    @{ Name = 'Ada Lovelace'; Email = 'ada@example.com';  Added = Get-Date },
    @{ Name = 'Alan Turing';  Email = 'alan@example.com'; Added = Get-Date }
```

A row is a hashtable of column names and values. `Id` is left out, so it gets the next AutoNumber; any other column left out is empty (NULL).

4.4 Read the rows

```
Get-kAccessRow ./Contacts.accdb Contacts | Format-Table Id, Name, Email, Added
```

The first example script does exactly these four steps. This is its real output:

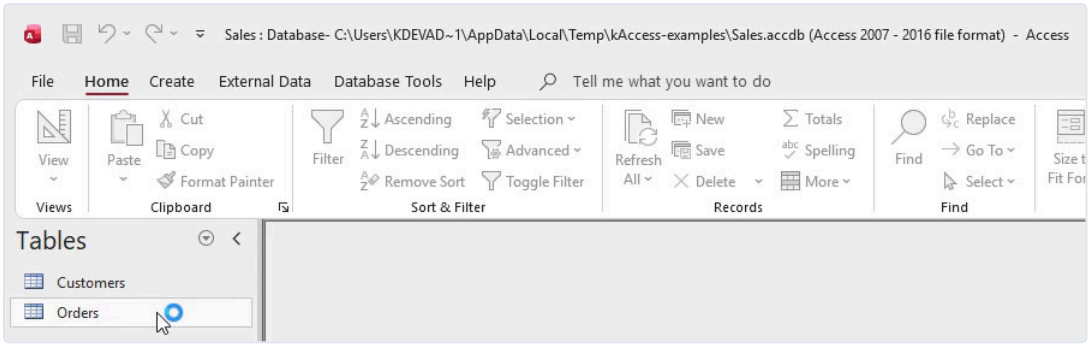
PowerShell 7 on Linux

```
PS ~/examples> ./01-Quick-Start.ps1
Id Name         Email            Added
-- ----         -----            -----
 1 Ada Lovelace ada@example.com  11/20/2026 14:24:34
 2 Alan Turing  alan@example.com 11/20/2026 14:24:34

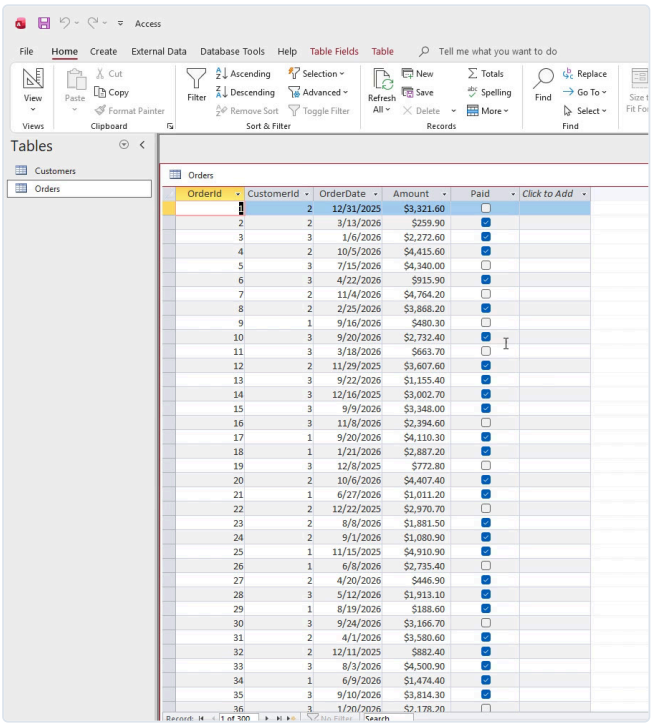
Database: /tmp/kAccess-examples/QuickStart.accdb - open it in Microsoft Access if you
have it, it is an ordinary .accdb file.
```

4.5 Open the file in Access

If Microsoft Access is installed on a Windows computer, double-click the file. The navigation pane lists the tables kAccess created, and a double-click on a table opens it in Datasheet View.



The navigation pane of a database that kAccess created



The rows in the Access datasheet - written without Access

4.6 The pattern of every command



Now that you have used four commands, you have seen the pattern of all eighteen:

- **The file comes first.** The first parameter is always `-Path`, the database file. A relative path such as `./Contacts.accdb` counts from the current folder.
- **The table comes second.** Commands that work on a table take its name as the second parameter (`-Table` or `-Name`).
- **Everything else is named.** `-Column`, `-Index`, `-Row`, `-Value` and the rest are written with their names.
- **Rows come and go through the pipeline.** Rows from `Get-kAccessRow` carry a `kAccessRowId`, so they can be piped straight on to `Set-kAccessRow` or `Remove-kAccessRow`.
- **`-Password` opens an encrypted file** in every command that reads or changes a database (Chapter 8.4).
- **An error is one clear message** that names the command and the cause, and nothing is half-written (Chapter 11).

```
Get-kAccessRow   ./Contacts.accdb  Contacts
#  command        -Path             -Table / -Name    named parameters...
```

> **Next:** Chapter 5 shows all column types and indexes, and Chapter 6 shows how to change and delete rows in bulk. If you would rather learn from complete scripts, go to Chapter 9.

CHAPTER 5

Tables, columns and indexes

> A well-designed table keeps your data correct: the right column types, a primary key and unique indexes where duplicates must never appear. kAccess creates all of them, and Access sees exactly the design you described.

5.1 Column types

The `Type` of a column uses the names of the Access file format. The table shows what each one is called in Access and what you get in PowerShell:

Type	In Access	Holds	PowerShell value
`TEXT`	Short Text	up to 255 characters; `Length` sets the maximum	string
`MEMO`	Long Text	longer text	string
`BOOLEAN`	Yes/No	true or false	bool
`BYTE`	Number (Byte)	0 to 255	byte
`INT`	Number (Integer)	−32,768 to 32,767	short
`LONG`	Number (Long Integer), AutoNumber	whole numbers, ±2.1 billion	int
`FLOAT`	Number (Single)	decimal numbers, 7 digits	single
`DOUBLE`	Number (Double)	decimal numbers, 15 digits	double
`NUMERIC`	Number (Decimal)	exact decimals; `Precision`, `Scale`	decimal
`MONEY`	Currency	amounts, exact to 4 decimals	decimal
`SHORT_DATE_TIME`	Date/Time	date and time	datetime
`EXT_DATE_TIME`	Date/Time Extended	exact to 100 ns (V2019 files)	datetime
`GUID`	Replication ID	a GUID; can be an AutoNumber	guid
`BINARY`, `OLE`	Binary, OLE Object	bytes	byte[]
`ATTACHMENT`	Attachment	files (Chapter 8.1)	–
`MULTI_VALUE`	multi-value Lookup	several values (Chapter 8.2)	–

5.2 Column options

Key	Used with	Meaning
`Name`	every column	the column name (required)
`Type`	every column	one of the types above (required)
`Length`	`TEXT`	the maximum number of characters
`AutoNumber`	`LONG`, `GUID`	`$true`: Access fills in the next number or a new GUID
`Precision`, `Scale`	`NUMERIC`	total digits and digits after the decimal point
`ValueType`	`MULTI_VALUE`	the type of each value; `TEXT` when left out

5.3 Primary keys and indexes

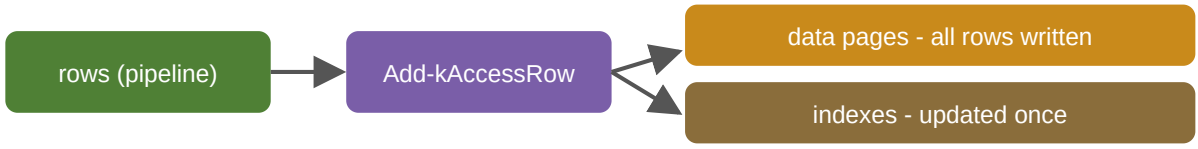
An index makes look-ups fast and can stop duplicates. Give `-Index` one hashtable per index:

```
New-kAccessTable ./Sales.accdb Customers -Column `
    @{ Name = 'CustomerId'; Type = 'LONG'; AutoNumber = $true },
    @{ Name = 'Company';    Type = 'TEXT'; Length = 80 },
    @{ Name = 'Email';      Type = 'TEXT'; Length = 120 } `
  -Index @{ Name = 'PrimaryKey'; Columns = @('CustomerId'); PrimaryKey = $true
},
         @{ Name = 'Email';      Columns = @('Email');      Unique = $true },
         @{ Name = 'ByCompany';  Columns = @('Company', '-CustomerId') }
```

Key	Meaning
`Name`	the index name; Access calls the primary key index `PrimaryKey`
`Columns`	one or more columns; a leading `-` sorts that column descending
`PrimaryKey = $true`	the primary key: unique and never empty
`Unique = $true`	no two rows may have the same value
`IgnoreNulls = $true`	rows with an empty value are left out of the index

kAccess keeps every index up to date when rows are added, changed or deleted, and it refuses a change that would break a unique index (Chapter 6.2).

Piped rows are written in one pass: the data pages first, then every index once.



5.4 Look at the design of a database

`Get-kAccessTable` lists the user tables with their row counts and indexes; with a table name it gives that table, and its `Columns` property lists the columns. Here it describes the sales database of example 03:

```
Get-kAccessTable ./Sales.accdb | Format-Table Name, RowCount, Indexes
(Get-kAccessTable ./Sales.accdb Orders).Columns | Format-Table Name, Type,
Length
```

PowerShell 7 on Linux

```
PS ~/examples> ./tour-tables.ps1
Name      RowCount Indexes
----      -------- -------
Customers        3 {PrimaryKey, Email}
Orders         298 {PrimaryKey, CustomerId}


Name       Type            Length
----       ----            ------
OrderId    LONG                 4
CustomerId LONG                 4
OrderDate  SHORT_DATE_TIME      8
Amount     MONEY                8
Paid       BOOLEAN              1
```

`Length` is the size in the file, in bytes. A `TEXT` column stores two bytes per character, so a `TEXT` column created with `Length = 80` shows 160 here.

5.5 Rename and delete tables

```
Rename-kAccessTable ./Sales.accdb Orders -NewName Orders2026
Remove-kAccessTable ./Sales.accdb OldOrders                # add -WhatIf to see
what would happen first
```

A deleted table is gone at once, but the space it used stays in the file until Access compacts it (File >Info >Compact & Repair Database).

CHAPTER 6

Working with rows

> Reading, adding, changing and deleting rows is where kAccess saves the most time. Thousands of rows go in or out with one pipeline, every row is checked before anything is written, and a batch is either written completely or not at all.

6.1 Read rows

```
Get-kAccessRow ./Sales.accdb Customers | Format-Table
```

PowerShell 7 on Linux

```
PS ~/examples> ./tour-rows.ps1
CustomerId Company   Email               Country kAccessRowId
---------- -------   -----               ------- ------------
         2 Fabrikam  orders@fabrikam.com Germany 54,1
         3 Northwind sales@northwind.com USA     54,2
         1 Contoso   buy@contoso.com     Sverige 54,3
```

Each row is an object with one property per column, so all of PowerShell works on it: `Where-Object` filters, `Sort-Object` sorts, `Group-Object` groups and `Export-Csv` or `ConvertTo-Json` exports.

The extra property **`kAccessRowId`** is the row's place in the file (page, row). It is how `Set-kAccessRow` and `Remove-kAccessRow` find the row again. A row id is valid until the table is written again, so read the rows, then change them in the same pipeline.

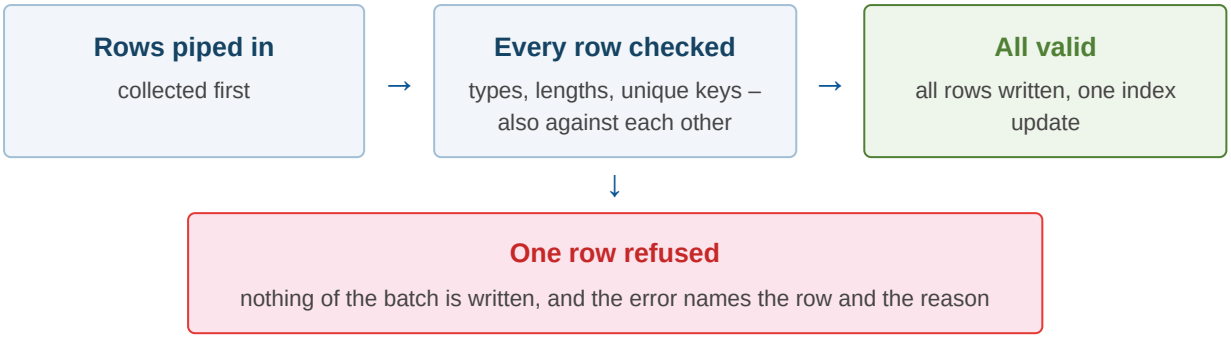
6.2 Add rows

Give one row with `-Row`, or pipe in as many as you like:

```
Add-kAccessRow ./Sales.accdb Customers -Row @{ Company = 'Contoso'; Email =
'buy@contoso.com' }

1..5000 | ForEach-Object { @{ CustomerId = 1; Amount = $_ } } | Add-kAccessRow
./Sales.accdb Orders
```

Rows piped in are collected and written **together, with one index update**. That is much faster than one command per row, so build the rows in the pipeline and send them to one `Add-kAccessRow`.


Rows piped in
collected first
Every row checked
types, lengths, unique keys – also against each other
All valid
all rows written, one index update
One row refused
nothing of the batch is written, and the error names the row and the reason

A column left out of the hashtable is empty (NULL); an AutoNumber column left out gets the next number. The all-or-nothing rule means that a failed import never leaves half a batch in the file – correct the data and run the same command again.

From example 03 – a duplicate e-mail address in a batch of two

```
Refused as it should be: Add-kAccessRow failed: row 1: the unique index 'Email' already
has that value
Customers now: 3
```

6.3 Change rows

Read the rows, keep the ones to change, and pipe them to `Set-kAccessRow` with the new values. Columns that are not in `-Value` keep their values.

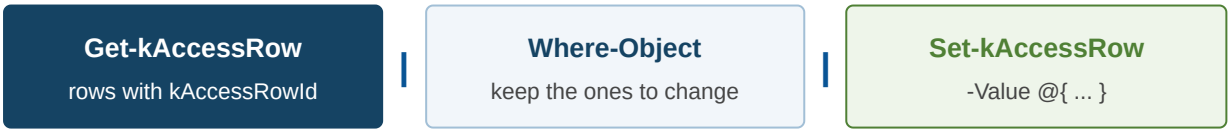
```
Get-kAccessRow ./Sales.accdb Customers | Where-Object Country -eq 'Sweden' |
    Set-kAccessRow ./Sales.accdb Customers -Value @{ Country = 'Sverige' }
```

Unique indexes are checked before anything is written, as for new rows.

6.4 Delete rows

```
Get-kAccessRow ./Sales.accdb Orders | Where-Object Amount -lt 100 |
    Remove-kAccessRow ./Sales.accdb Orders
```

All rows piped in are deleted together. If one of the ids is not a row of the table, nothing is deleted.



or `| Remove-kAccessRow` in the last step – the same read, filter, write pattern

6.5 Try it first with -WhatIf

Before a big change, add `-WhatIf`. PowerShell lists every row the command would change or delete, and the file is not touched:

```
Get-kAccessRow ./Sales.accdb Orders | Where-Object Paid -eq $false |
    Remove-kAccessRow ./Sales.accdb Orders -WhatIf
# What if: Performing the operation "Remove" on target "Orders row 61,4".
```

> **Join and group in PowerShell:** kAccess gives you the rows, and PowerShell does the rest. Example 03 (Chapter 9) joins customers and orders with a hashtable and reports sales per customer with `Group-Object` and `Measure-Object`.

CHAPTER 7

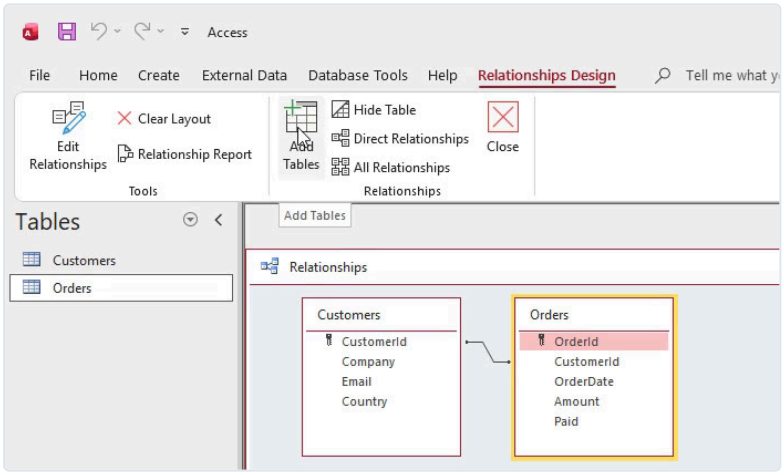
Relationships and saved queries

> A database is more than its tables. kAccess draws the relationships that Access users rely on, reads the SQL of every saved query and creates new saved queries from SQL – so a database built by a script feels like one built by hand.

7.1 Create a relationship

```
New-kAccessRelationship ./Sales.accdb -PrimaryTable Customers -PrimaryColumn
CustomerId `
    -SecondaryTable Orders -SecondaryColumn CustomerId
```

The primary table is the "one" side and the secondary table the "many" side: one customer has many orders. The command answers with the name of the relationship – the two table names, `CustomersOrders`, unless you give `-Name`. For keys of several columns, give the columns as lists in the same order.



Database Tools >Relationships in Access shows the join line kAccess created

> **Note:** the relationship is created *without referential integrity*. Access shows the join line and uses it in new queries, but it does not stop an order with an unknown customer. Check such rules in your script, or turn on "Enforce Referential Integrity" in Access afterwards.

7.2 Read saved queries

`Get-kAccessQuery` lists the saved queries with their type and SQL, the way Access shows them in SQL View. It is a fast way to document an old database or to find every query that uses a table before you change it:

```
Get-kAccessQuery ./Legacy.mdb | Where-Object Sql -match 'Invoices' | Format-
Table Name, Type
```

The queries that Access makes behind forms and reports (named `~sq_...`) are hidden; add `-IncludeHidden` to list them too.

7.3 Create saved queries from SQL

`New-kAccessQuery` stores a query under a name. Access lists it in the navigation pane and runs it like a query made in the Query Designer.

```
New-kAccessQuery ./Sales.accdb SalesPerCustomer @"
SELECT Customers.Company, Sum(Orders.Amount) AS Total
FROM Customers INNER JOIN Orders ON Customers.CustomerId = Orders.CustomerId
GROUP BY Customers.Company;
"@
```

Here three queries are added to the sales database and read back:

PowerShell 7 on Linux

```
PS ~/examples> ./tour-queries.ps1
Name             Type
----             ----
BigOrders        SELECT
SalesPerCustomer SELECT
MarkOldPaid      UPDATE

SELECT Customers.Company, Sum(Orders.Amount) AS Total
FROM Customers INNER JOIN Orders ON Customers.CustomerId = Orders.CustomerId
GROUP BY Customers.Company;
```

Supported query types

Query type	SQL
Select	`SELECT` with joins, `WHERE`, `GROUP BY`, `HAVING`, `ORDER BY`, `DISTINCT`, `TOP`
Make-table	`SELECT ... INTO NewTable FROM ...`
Append	`INSERT INTO ... SELECT` or `INSERT INTO ... VALUES`
Update	`UPDATE ... SET ... WHERE`
Delete	`DELETE FROM ... WHERE`
Crosstab	`TRANSFORM ... SELECT ... PIVOT`
Union	`SELECT ... UNION SELECT ...`
Data definition	`CREATE`, `ALTER`, `DROP`
Parameter queries	a `PARAMETERS` line before any of the above
Pass-through	any SQL, sent unchanged to an ODBC source given with `-Connect`

```
New-kAccessQuery ./Sales.accdb ServerOrders 'SELECT * FROM dbo.Orders' -Connect
'ODBC;DSN=Shop;'
```

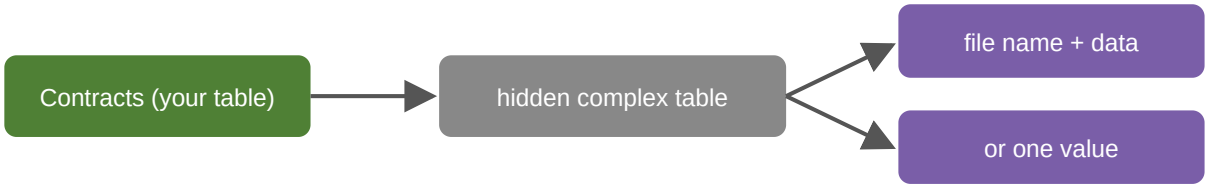
> **A saved query is stored, not run.** kAccess writes the query into the database; Access runs it when a user opens it. To work with the data in a script, read the tables with `Get-kAccessRow` and filter in PowerShell. A sub-query in `FROM` and tables in another database (`IN '...'`) are not supported.

CHAPTER 8

Attachments, multi values and encrypted files

> Modern Access databases store files in attachment fields, several choices in multi-value fields and precise timestamps in Date/Time Extended columns, and many are protected by a password. kAccess handles all four, so even these databases can be read, migrated and kept up to date from a script.

Access keeps these values in a hidden table; kAccess finds it for you.



8.1 Attachment fields

An attachment field holds files – pictures, PDFs, documents – inside the database. Create one with the type `ATTACHMENT` in an Access 2007 or later file:

```
New-kAccessTable ./Products.accdb Products -Column `
    @{ Name = 'Id'; Type = 'LONG'; AutoNumber = $true }, @{ Name = 'Name'; Type
= 'TEXT'; Length = 60 },
    @{ Name = 'Pictures'; Type = 'ATTACHMENT' },
    @{ Name = 'Colours';  Type = 'MULTI_VALUE'; ValueType = 'TEXT' }
```

Files are added to the rows you pipe in, and listed, saved or deleted in the same way:

```
# add a file to the chair
Get-kAccessRow ./Products.accdb Products | Where-Object Name -eq 'Chair' |
    Add-kAccessAttachment ./Products.accdb Products -Column Pictures -File
./chair.jpg
# save every attachment in the table to a folder
Get-kAccessAttachment ./Products.accdb Products -Column Pictures -Destination
./pictures
# delete all BMP pictures
Get-kAccessAttachment ./Products.accdb Products -Column Pictures | Where-Object
FileType -eq 'bmp' |
    Remove-kAccessAttachment ./Products.accdb Products -Column Pictures
```

Each attachment comes back with its `FileName`, `FileType`, `Size`, `TimeStamp` and its bytes in `Data`, so a script can also hand the file to another system without saving it first.

8.2 Multi-value fields

A multi-value field holds several values in one cell, such as the colours a product comes in. Add, list and delete the values with the `*-kAccessMultiValue` commands:

```
Get-kAccessRow ./Products.accdb Products | Where-Object Name -eq 'Chair' |
    Add-kAccessMultiValue ./Products.accdb Products -Column Colours -Value
'Oak', 'Walnut'
Get-kAccessMultiValue ./Products.accdb Products -Column Colours
```

The real output of the commands in 8.1 and 8.2, run against a new database:

PowerShell 7 on Linux

```
PS ~/examples> ./tour-complex.ps1
FileName  FileType Size
--------  -------- ----
chair.txt txt        20


Id Value
-- -----
 1 Oak
 2 Walnut
```

> **How Access stores them:** attachment and multi-value columns keep their contents in hidden tables that Access makes for them. kAccess creates and maintains those hidden tables exactly as Access does, which is why the fields work when the file is opened in Access. Such columns cannot be indexed, as in Access.

8.3 Date/Time Extended

Access 2019 added the **Date/Time Extended** type: dates from year 1 to 9999, exact to 100 nanoseconds, which matches a .NET `[datetime]` exactly. kAccess reads and writes these columns, and creates them in files of the `V2019` format:

```
New-kAccessDatabase ./Log.accdb -Format V2019
New-kAccessTable ./Log.accdb Events -Column @{ Name = 'At'; Type =
'EXT_DATE_TIME' }, @{ Name = 'Text'; Type = 'MEMO' }
Add-kAccessRow ./Log.accdb Events -Row @{ At = Get-Date; Text = 'Backup
finished' }
```

8.4 Password-protected databases

Every kAccess command that opens an existing database takes `-Password`. With it, kAccess decrypts the file as it reads and encrypts what it writes, so an encrypted database stays encrypted:

```
$Password = Read-Host 'Database password' -MaskInput
Get-kAccessRow ./Payroll.accdb Salaries -Password $Password | Export-Csv
./salaries.csv
```

File	Encryption	Read	Write
.mdb (Access 2000–2003)	Jet RC4 database password	Yes	Yes
.accdb (Access 2007)	Office RC4, CryptoAPI RC4	Yes	Yes
.accdb (Access 2010 and later)	AES and Agile AES	Yes	Yes
A new database	–	Created without a password	

To protect a database that kAccess created, open it in Access with File >Open >Open Exclusive, then choose File >Info >Encrypt with Password.

> **Keep passwords out of scripts.** Read the password with `Read-Host -MaskInput`, or take it from a secret store such as the SecretManagement module or an Azure Key Vault, instead of writing it in the script file.

CHAPTER 9

The seven examples

> The module ships seven complete scripts that solve real tasks: importing a CSV file, building a sales database, maintaining data, taking an inventory before a migration, exporting everything and converting between Access versions. Run them as they are, or copy one as the start of your own script.

#	Script	What it shows	Parameters
01	`01-Quick-Start.ps1`	a database, a table with an AutoNumber key, rows in and out	–
02	`02-Csv-To-Access.ps1`	any CSV into a new table, column types from the data	`-Csv -Database -Table`
03	`03-Sales-Database.ps1`	keys, a unique index, a relationship and a sales report	–
04	`04-Maintain-Data.ps1`	bulk changes and deletes, rename and drop tables	`-Database`
05	`05-Database-Inventory.ps1`	every table of every Access file in a folder tree	`-Path`
06	`06-Export-Everything.ps1`	every table as CSV and JSON	`-Database -Out`
07	`07-Convert-Format.ps1`	a whole database copied into another Access format	`-Source -Target -Format`

9.1 Run the examples

Copy the scripts to your own folder (Chapter 3.4) and run them in order. Without parameters they work in a folder named `kAccess-examples` in your temp folder, and example 03 creates the `Sales.accdb` that examples 04, 06 and 07 use:



05 Inventory reads every database the others created

Each of the following sections describes one script: what it is good for, how to run it, what it does step by step, and its real output. The temp folder is shown as `/tmp/kAccess-examples`, as on Linux; on Windows it is `%TEMP%\kAccess-examples`.

9.2 Example 01 – Quick start

01

A new Access database in 30 seconds

`./01-Quick-Start.ps1` · no parameters · creates `QuickStart.accdb`

The shortest complete kAccess script, and the best way to check an installation. Chapter 4 explains every line.

1. Creates `QuickStart.accdb` with `New-kAccessDatabase`.
2. Creates the table `Contacts`: an AutoNumber `Id` as primary key, `Name`, `Email` and the date `Added`.
3. Adds two contacts in one `Add-kAccessRow` call.
4. Reads them back and shows them as a table.

PowerShell 7 on Linux

```
PS ~/examples> ./01-Quick-Start.ps1
Id Name         Email            Added
-- ----         -----            -----
 1 Ada Lovelace ada@example.com  11/20/2026 14:24:34
 2 Alan Turing  alan@example.com 11/20/2026 14:24:34

Database: /tmp/kAccess-examples/QuickStart.accdb - open it in Microsoft Access if you
have it, it is an ordinary .accdb file.
```

9.3 Example 02 – Import a CSV file

02

Any CSV file into a new Access table

`./02-Csv-To-Access.ps1 -Csv ./sales.csv -Database ./Sales.accdb -Table Sales`

Turns a CSV export from any system into a properly typed Access table, with thousands of rows in seconds. Without `-Csv` it imports a sample: the processes running on your computer.

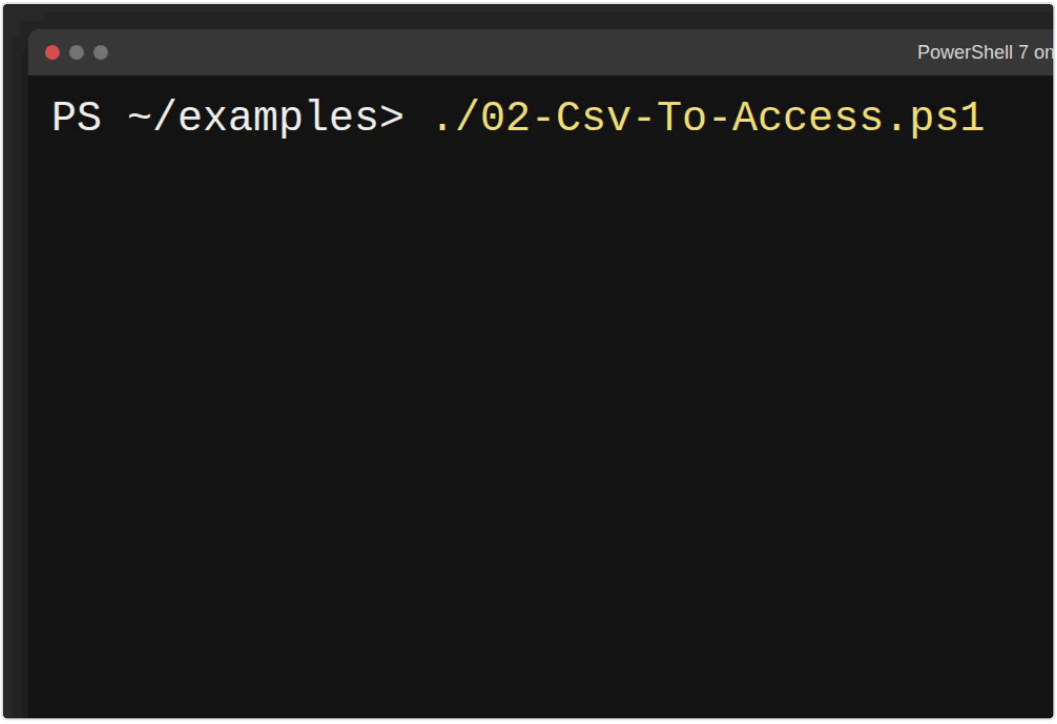
1. Reads the CSV file with `Import-Csv`.
2. Works out the narrowest Access type for each column from all its values, in this order: `LONG`, `DOUBLE`, `BOOLEAN`, `SHORT_DATE_TIME`, then `TEXT` – or `MEMO` when a value is longer than 255 characters.
3. Creates the database if needed, and a table with an AutoNumber `RowId` key plus one column per CSV column.
4. Converts every value to its type and sends all rows to **one** `Add-kAccessRow` – the bulk write that makes it fast.
5. Reports the number of rows, the time it took and the column types it chose.

PowerShell 7 on Linux

```
PS ~/examples> ./02-Csv-To-Access.ps1
165 rows imported into Processes in 0.5
s

Name     Type
----     ----
RowId    LONG
Name     MEMO
Id       LONG
MemoryMB DOUBLE
Started  SHORT_DATE_TIME
```

> **Adapt it:** the function `Get-ColumnType` is the place to add your own rules, for example to keep postal codes and phone numbers as `TEXT` even when they look like numbers. Empty CSV cells become empty (NULL) values.



Example 02 on Linux: a CSV file becomes an Access table

9.4 Example 03 – A sales database

03

Customers and orders with keys, a relationship and a report

`./03-Sales-Database.ps1` · no parameters · creates `Sales.accdb`

A small but real database design, and proof that kAccess keeps the data correct. This is the example in both demo videos.

Customers

◆ CustomerId – AutoNumber, primary key
Company
Email – unique index
Country

1 → ∞

Orders

◆ OrderId – AutoNumber, primary key
CustomerId – index
OrderDate, Amount (Currency)
Paid (Yes/No)

The design that example 03 creates, with the relationship CustomersOrders

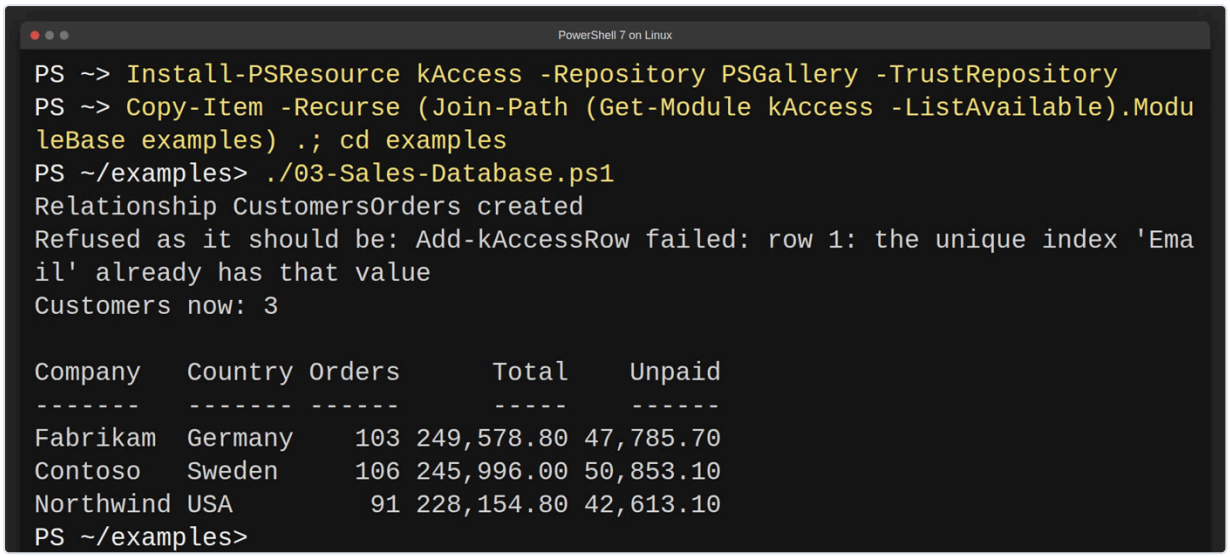
1. Creates both tables with their primary keys, a unique index on `Email` and an index on `Orders.CustomerId`.
2. Creates the relationship between `Customers.CustomerId` and `Orders.CustomerId`.
3. Adds three customers and 300 random orders from the last year, each in one bulk write.
4. Tries to add a batch of two new customers, one of them with an e-mail address that already exists. The unique index refuses it, and **neither** row is written.
5. Joins customers and orders in PowerShell and reports orders, total and unpaid amount per customer.

PowerShell 7 on Linux

```
PS ~/examples> ./03-Sales-Database.ps1
Relationship CustomersOrders created
Refused as it should be: Add-kAccessRow failed: row 1: the unique index 'Email' already
has that value
Customers now: 3

Company   Country Orders      Total    Unpaid
-------   ------- ------      -----    ------
Fabrikam  Germany    103 249,578.80 47,785.70
Contoso   Sweden     106 245,996.00 50,853.10
Northwind USA         91 228,154.80 42,613.10
```

The report is built with plain PowerShell: the customers go into a hashtable by id, and the orders are grouped with `Group-Object CustomerId` and summed with `Measure-Object Amount -Sum`. Open `Sales.accdb` in Access to see the tables and the relationship, as in Chapter 7.1.



Example 03 on Linux: the sales report

9.5 Example 04 – Maintain data

04

Day-to-day changes to an existing database

`./04-Maintain-Data.ps1 -Database ./Sales.accdb` · default: the `Sales.accdb` of example 03

The typical clean-up jobs that otherwise mean an evening in Access, as one script that can run on a schedule.

1. Marks every unpaid order older than 180 days as paid – one pipeline, one index update.
2. Changes the country of every Swedish customer from "Sweden" to "Sverige".
3. Deletes the small test orders under 100.
4. Creates a table, renames it and deletes it again, to show `Rename-kAccessTable` and `Remove-kAccessTable`.
5. Lists the tables with their row counts and indexes.

PowerShell 7 on Linux

```
PS ~/examples> ./04-Maintain-Data.ps1
27 old orders marked as paid
2 orders under 100 deleted

Name      RowCount Indexes
----      -------- -------
Customers        3 {PrimaryKey, Email}
Orders         298 {PrimaryKey, CustomerId}
```

The numbers differ from run to run, because example 03 creates random order dates and amounts.

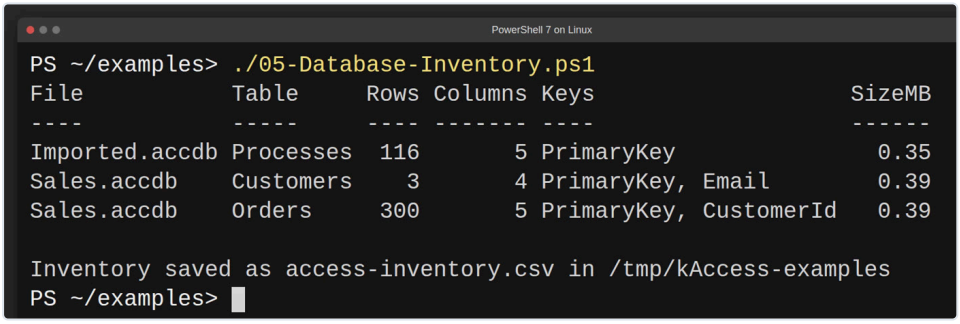
9.6 Example 05 – Inventory of a folder

05 **Every Access database in a folder tree**

`./05-Database-Inventory.ps1 -Path \\server\share\departments`

The first step of every Access migration: find out what there is. The script lists each `.accdb` and `.mdb` file with its tables, row counts, column counts, keys and file size, and saves the list as `access-inventory.csv` in the folder – ready for Excel.

1. Finds all `.accdb` and `.mdb` files below `-Path`.
2. Reads the tables of each file with `Get-kAccessTable`.
3. Lists a file it cannot read with the reason, for example an encrypted file, and goes on with the next one.
4. Shows the inventory and saves it as CSV.



Example 05 in the Linux demo

PowerShell 7 on Linux – after all seven examples

```
PS ~/examples> ./05-Database-Inventory.ps1
File             Table     Rows Columns Keys                   SizeMB
----             -----     ---- ------- ----                   ------
Imported.accdb   Processes  165       5 PrimaryKey               0.22
QuickStart.accdb Contacts     2       4 PrimaryKey               0.19
Sales.accdb      Customers    3       4 PrimaryKey, Email        0.24
Sales.accdb      Orders     298       5 PrimaryKey, CustomerId   0.24

Inventory saved as access-inventory.csv in /tmp/kAccess-examples
```

9.7 Example 06 – Export everything

06

Every table as CSV and JSON

```
./06-Export-Everything.ps1 -Database ./Legacy.mdb -Out ./export
```

Gets your data *out* of Access, for Excel, SharePoint, SQL Server, Power BI or a web page. Scheduled on a Linux server, the export runs every night without Access anywhere.

1. Creates the output folder.
2. For each table, reads all rows and leaves out the `kAccessRowId` property.
3. Writes `<table>.csv` with `Export-Csv` and `<table>.json` with `ConvertTo-Json`.

PowerShell 7 on Linux

```
PS ~/examples> ./06-Export-Everything.ps1
Customers                3 rows -> Customers.csv, Customers.json
Orders                 298 rows -> Orders.csv, Orders.json
```

9.8 Example 07 – Convert to another Access format

07

A whole database copied into a new format

```
./07-Convert-Format.ps1 -Source ./Sales.accdb -Target ./Sales2000.mdb -Format V2000
```

Makes an Access 2000 `.mdb` that an old application can open from a modern `.accdb` – or the other way round, to bring an old file up to date.

1. Creates the target database in the chosen `-Format` (Chapter 2.3).
2. Creates each table with the same columns and types. A `TEXT` length is halved, because `Get-kAccessTable` gives bytes and `New-kAccessTable` takes characters.
3. Creates the primary key on the first column when the source table has one.
4. Copies all rows in one bulk write per table and lists the result.

PowerShell 7 on Linux

```
PS ~/examples> ./07-Convert-Format.ps1
Customers        3 rows copied
Orders         298 rows copied

Name      RowCount
----      --------
Customers        3
Orders         298
```

> **Note:** the script copies tables, column types, the primary key and the rows. Other indexes, relationships and saved queries are not copied; add them with `-Index`, `New-kAccessRelationship` and `New-kAccessQuery` if the target needs them. Attachment and multi-value columns exist only in Access 2007 and later files.

CHAPTER 10

Command reference

> All 18 commands with their parameters and an example. Every command except `New-kAccessDatabase` also takes `-Password` for an encrypted file, and `Get-Help <command> -Full` shows the same help in PowerShell.

10.1 Databases and tables

`New-kAccessDatabase`

Creates a new, empty database file and answers with the file. An existing file is never overwritten.

Parameter	Description
`-Path` (1st)	the new file, `.accdb` or `.mdb`
`-Format`	`V2000`, `V2003` (.mdb), `V2007`, `V2010` (default), `V2016`, `V2019` (.accdb)

```
New-kAccessDatabase ./Old.mdb -Format V2000
```

`Get-kAccessTable`

Lists the user tables with `Name`, `RowCount`, `Columns` (Name, Type, Length) and `Indexes`.

Parameter	Description
`-Path` (1st)	the database
`-Name` (2nd)	only this table

```
Get-kAccessTable ./Shop.accdb Orders | Select-Object -ExpandProperty Columns
```

`New-kAccessTable`

Creates a table with columns and, optionally, indexes and a primary key. Chapter 5 lists the types and options.

Parameter	Description
`-Path` (1st), `-Name` (2nd)	the database and the new table
`-Column`	one hashtable per column: `Name`, `Type`, `Length`, `AutoNumber`, `Precision`, `Scale`, `ValueType`
`-Index`	one hashtable per index: `Name`, `Columns`, `PrimaryKey`, `Unique`, `IgnoreNulls`

```
New-kAccessTable ./Shop.accdb Items -Column @{ Name = 'Id'; Type = 'LONG';
AutoNumber = $true }, @{ Name = 'Title'; Type = 'TEXT' } `
    -Index @{ Name = 'PrimaryKey'; Columns = @('Id'); PrimaryKey = $true }
```

`Rename-kAccessTable`

Renames a table. Parameters: `-Path` (1st), `-Name` (2nd), `-NewName` (3rd).

```
Rename-kAccessTable ./Shop.accdb Orders -NewName Orders2026
```

`Remove-kAccessTable`

Deletes a table. Parameters: `-Path` (1st), `-Name` (2nd); supports `-WhatIf` and `-Confirm`. The space is freed when Access compacts the file.

```
Remove-kAccessTable ./Shop.accdb OldOrders
```

10.2 Rows

`Get-kAccessRow`

Reads the rows of a table: one object per row, with one property per column plus `kAccessRowId`.

Parameter	Description
`-Path` (1st), `-Table` (2nd)	the database and the table

```
Get-kAccessRow ./Shop.accdb Orders | Where-Object Total -gt 100
```

`Add-kAccessRow`

Adds rows. Rows piped in are written together with one index update; when one row is refused, nothing is written.

Parameter	Description
`-Path` (1st), `-Table` (2nd)	the database and the table
`-Row` (pipeline)	one hashtable per row: column name = value

```
1..500 | ForEach-Object { @{ Title = "Item $_" } } | Add-kAccessRow ./Shop.accdb Items
```

`Set-kAccessRow`

Changes columns of rows read with `Get-kAccessRow`; other columns keep their values. Supports `-WhatIf`.

Parameter	Description
`-Path` (1st), `-Table` (2nd)	the database and the table
`-kAccessRowId` (pipeline)	the rows, from `Get-kAccessRow`
`-Value`	a hashtable of column name = new value

```
Get-kAccessRow ./Shop.accdb Orders | Where-Object Customer -eq 'Contoso' | Set-kAccessRow ./Shop.accdb Orders -Value @{ Total = 0 }
```

`Remove-kAccessRow`

Deletes rows read with `Get-kAccessRow`, all together; when one id is not a row of the table, nothing is deleted. Parameters: `-Path`, `-Table`, `-kAccessRowId` (pipeline). Supports `-WhatIf`.

```
Get-kAccessRow ./Shop.accdb Orders | Where-Object Total -eq 0 | Remove-kAccessRow ./Shop.accdb Orders
```

10.3 Relationships and queries

`New-kAccessRelationship`

Creates a relationship (without referential integrity) and answers with its name.

Parameter	Description
`-Path` (1st)	the database
`-PrimaryTable`, `-PrimaryColumn`	the "one" side and its key column(s)
`-SecondaryTable`, `-SecondaryColumn`	the "many" side and its column(s)
`-Name`	the relationship name; default the two table names

```
New-kAccessRelationship ./Shop.accdb -PrimaryTable Customers -PrimaryColumn Id -SecondaryTable Orders -SecondaryColumn CustomerId
```

`Get-kAccessQuery`

Lists the saved queries with `Name`, `Type`, `Sql`, `Hidden` and `Connect`.

Parameter	Description
`-Path` (1st), `-Name` (2nd)	the database; only this query
`-IncludeHidden`	also the hidden `~sq_` queries behind forms and reports

```
(Get-kAccessQuery ./Shop.accdb -Name TopCustomers).Sql
```

`New-kAccessQuery`

Creates a saved query from its SQL text (Chapter 7.3 lists the query types).

Parameter	Description
`-Path` (1st), `-Name` (2nd), `-Sql` (3rd)	the database, the query name and its SQL
`-Connect`	an ODBC connect string: makes a pass-through query

```
New-kAccessQuery ./Shop.accdb BigOrders 'SELECT Orders.* FROM Orders WHERE
Orders.Total > 1000;'
```

10.4 Attachments and multi values

All six commands take `-Path` (1st), `-Table` (2nd) and `-Column`. The `Get-` and `Add-` commands work on the rows piped in from `Get-kAccessRow`; `Get-` without input lists every row.

`Get-kAccessAttachment`

Lists the files: `kAccessRowId`, `Id`, `FileName`, `FileType`, `Size`, `TimeStamp`, `Data` (the bytes). `-Destination` also saves the files to that folder.

```
Get-kAccessAttachment ./Shop.accdb Products -Column Pictures -Destination
./pictures
```

`Add-kAccessAttachment`

Adds the files in `-File` (one or more paths) to each row piped in.

```
Get-kAccessRow ./Shop.accdb Products | Where-Object Name -eq 'Chair' | Add-
kAccessAttachment ./Shop.accdb Products -Column Pictures -File ./chair.jpg
```

`Remove-kAccessAttachment`

Deletes the attachments piped in from `Get-kAccessAttachment` (by `-Id`). Supports `-WhatIf`.

`Get-kAccessMultiValue`

Lists the values: `kAccessRowId`, `Id`, `Value`.

`Add-kAccessMultiValue`

Adds the values in `-Value` to each row piped in.

```
Get-kAccessRow ./Shop.accdb Products | Where-Object Name -eq 'Chair' | Add-
kAccessMultiValue ./Shop.accdb Products -Column Colours -Value 'Red', 'Blue'
```

`Remove-kAccessMultiValue`

Deletes the values piped in from `Get-kAccessMultiValue` (by `-Id`). Supports `-WhatIf`.

CHAPTER 11

Troubleshooting and limits

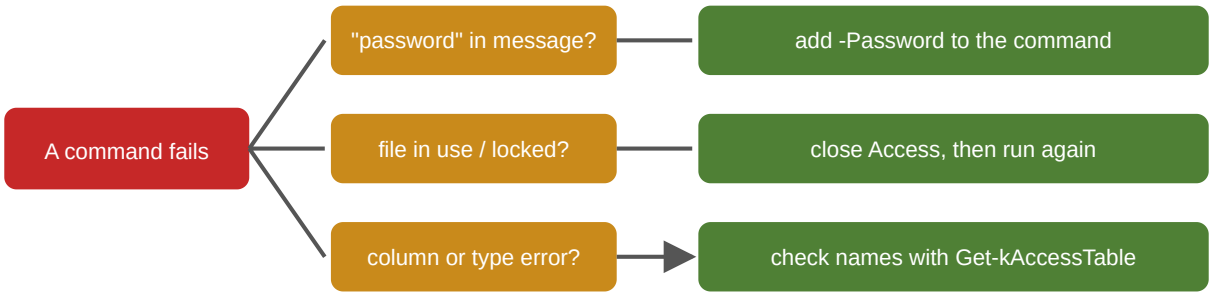
> When kAccess cannot do what you asked, it stops with one clear message that names the command and the cause – and it has written nothing. This chapter explains the messages you are most likely to meet, and lists the limits plainly, so you know them before you plan a job.

11.1 What an error looks like

Every error is a normal PowerShell terminating error, so `try` / `catch` handles it and a scheduled job stops instead of carrying on with bad data. Three typical refusals, as kAccess reports them:

PowerShell 7 on Linux

```
PS ~/examples> ./tour-errors.ps1
Get-kAccessRow failed: there is no table 'Invoices'
Get-kAccessRow failed: the file '/tmp/kAccess-examples/Missing.accdb' does not exist
Add-kAccessRow failed: row 0: the unique index 'Email' already has that value
```



11.2 Common problems

You see	Cause	What to do
The term 'Get-kAccessRow' is not recognized	the module is not installed for this user, or you run Windows PowerShell 5.1	start `pwsh`, run `Get-Module kAccess -ListAvailable`, install (Chapter 3)
the file '...' does not exist	a wrong path; a relative path counts from the current folder	check with `Test-Path`; use a full path in scheduled jobs
there is no table '...'	a misspelt table name, or a linked table, query or system table	list the names with `Get-kAccessTable`
the unique index '...' already has that value	a duplicate key in the rows or in the batch itself	remove the duplicate; nothing of the batch was written
cannot be opened ... password	the file is encrypted	add `-Password` (Chapter 8.4)
is not a row id	a row from another table, or a hand-made object, was piped to `Set-` or `Remove-kAccessRow`	pipe rows straight from `Get-kAccessRow` of the same table
Access says the database is in use	the file is open in Access while a script writes to it	close it in Access first; run writing jobs when nobody works in the file
The file did not get smaller after deletes	freed pages stay in the file	compact it in Access (File >Info >Compact & Repair Database)

> **Catch and log in scheduled jobs:** wrap the work in `try { ... } catch { $_.Exception.Message | Add-Content ./job.log; exit 1 }`, so the task scheduler or cron sees the failure and the log tells you why.

11.3 Limits

Knowing the limits up front saves a surprise later. kAccess deliberately does what it does completely, and leaves out what it cannot do correctly:

Area	Limit
Forms, reports, macros and VBA	not read and not written; they are kept unchanged in the file
Running queries	saved queries are read and created as SQL, not run; filter the rows in PowerShell
Query SQL	no sub-query in `FROM` and no tables in another database (`IN '...'`)
Relationships	created without referential integrity
Encryption	encrypted files are read and written, but a new database is created without a password
Attachment and multi-value columns	only in Access 2007 and later files; cannot be indexed, as in Access
Date/Time Extended columns	created only in `V2019` files, as in Access
Deleted tables and rows	the space is freed when Access compacts the file
Linked tables	the data of a table linked to another file or server is not in this file; open the source instead
Windows PowerShell 5.1	not supported; kAccess needs PowerShell 7.2 or later

11.4 Questions and answers

Can several scripts use the same file at once?

Reading is safe. Let only one program write at a time: kAccess opens the file for the length of one command, and two writers at the same moment can overwrite each other's work.

How fast is it?

Fast enough for everyday data: in the Linux demo, example 02 imported more than a hundred rows in less than a second. Pipe rows into one `Add-` , `Set-` or `Remove-kAccessRow` rather than calling the command once per row – the bulk write is the single biggest speed-up.

Is my data safe?

Every row is checked before anything is written, a refused batch writes nothing, and the files are verified against the reference library Jackcess and against Microsoft Access itself. Still, as with any tool that writes to a file: keep a backup copy before a large change.

Where do I report a problem?

Use the contact page at `www.kalmstrom.com`. Include the kAccess version (`Get-Module kAccess -ListAvailable`), the PowerShell version, the command and the full error message.

APPENDIX A

Quick reference

Install and start

Task	Command
Install	`Install-PSResource kAccess`
Update	`Update-PSResource kAccess`
Load	`Import-Module kAccess`
Help	`Get-Help New-kAccessTable -Full`

Database and tables

Task	Command
New database	`New-kAccessDatabase f.accdb`
List tables	`Get-kAccessTable f.accdb`
Columns	`(Get-kAccessTable f.accdb T).Columns`
New table	`New-kAccessTable f.accdb T -Column ...`
Rename	`Rename-kAccessTable f.accdb T -NewName U`
Delete	`Remove-kAccessTable f.accdb T`

Column and index hashtables

```
@{ Name='Id'; Type='LONG';
AutoNumber=$true }
@{ Name='Title'; Type='TEXT';
Length=100 }
@{ Name='Price'; Type='MONEY' }
@{ Name='PrimaryKey';
Columns=@('Id'); PrimaryKey=$true }
@{ Name='Email'; Columns=@('Email');
Unique=$true }
```

Rows

Task	Command
Read	`Get-kAccessRow f.accdb T`
Add one	`Add-kAccessRow f.accdb T -Row @{...}`
Add many	`$rows | Add-kAccessRow f.accdb T`
Change	`... | Set-kAccessRow f.accdb T -Value @{...}`
Delete	`... | Remove-kAccessRow f.accdb T`
Dry run	add `-WhatIf`

Relationships and queries

Task	Command
Relationship	`New-kAccessRelationship f.accdb -PrimaryTable ...`
List queries	`Get-kAccessQuery f.accdb`
New query	`New-kAccessQuery f.accdb Q 'SELECT ...'`

Special columns

Task	Command
Save attachments	`Get-kAccessAttachment f.accdb T -Column C -Destination dir`
Attach a file	`... | Add-kAccessAttachment f.accdb T -Column C -File x`
Multi values	`Get-kAccessMultiValue f.accdb T -Column C`
Encrypted file	add `-Password $pw`

Export one table

```
Get-kAccessRow f.accdb T |
  Select-Object -ExcludeProperty
kAccessRowId |
  Export-Csv T.csv
```

APPENDIX B

Glossary

Term	Meaning
.accdb	the Access database file of Access 2007 and later (the ACE engine)
.mdb	the Access database file of Access 2000–2003 (the Jet engine)
ACE, Jet	the database engines inside Access; also the name of the driver that scripts used before kAccess
AutoNumber	a column that Access fills with the next number (or a new GUID) for each new row
Bulk write	rows piped into one kAccess command and written together, with one index update
Hashtable	a PowerShell set of `Name = Value` pairs in `@{ }`; describes a row, a column or an index
Index	a sorted list of the values of one or more columns: fast look-ups, and no duplicates when unique
Jackcess	the open-source Java library for Access files that kAccess is ported from
kAccessRowId	the property on every row from `Get-kAccessRow` that tells where the row is in the file (page, row)
Module	a package of PowerShell commands; kAccess is one
Multi-value field	an Access column that holds several values per row
Pass-through query	a saved query whose SQL Access sends unchanged to an ODBC server
Pipeline	the `|` that sends objects from one command to the next
PowerShell Gallery	Microsoft's public store of PowerShell modules, where kAccess is published
Primary key	the unique index that identifies each row; Access calls it `PrimaryKey`
Referential integrity	an Access rule that a related row must exist; not set by kAccess

Index

-Password – 8.4

-WhatIf – 2.2, 6.5

Add-kAccessAttachment – 8.1, 10.4

Add-kAccessMultiValue – 8.2, 10.4

Add-kAccessRow – 4.3, 6.2, 10.2

All-or-nothing writes – 6.2, 11.1

Attachments – 8.1

Column types – 5.1

Compact and repair – 5.5, 11.2

Converting formats – 9.8

CSV import – 9.3

Date/Time Extended – 8.3

Encrypted files – 8.4

Errors – 11.1, 11.2

Examples – 9

Export to CSV and JSON – 9.7

File formats – 2.3

Get-kAccessAttachment – 8.1, 10.4

Get-kAccessMultiValue – 8.2, 10.4

Get-kAccessQuery – 7.2, 10.3

Get-kAccessRow – 4.4, 6.1, 10.2

Get-kAccessTable – 5.4, 10.1

Indexes – 5.3

Installation – 3

Inventory of a folder – 9.6

Limits – 11.3

Multi-value fields – 8.2

New-kAccessDatabase – 4.1, 10.1

New-kAccessQuery – 7.3, 10.3

New-kAccessRelationship – 7.1, 10.3

New-kAccessTable – 4.2, 5, 10.1

Offline installation – 3.5

PowerShell 7 – 2.1

Primary key – 5.3

Relationships – 7.1

Remove-kAccessRow – 6.4, 10.2

Remove-kAccessTable – 5.5, 10.1

Rename-kAccessTable – 5.5, 10.1

Saved queries – 7.2, 7.3

Set-kAccessRow – 6.3, 10.2

Unique index – 5.3, 6.2

Update and remove the module – 3.6