SQL Server and Oracle

dlib
an AXIELL product

Axiell ALM Netherlands BV

Contents

1 Database platforms compared **1**

2 SQL database analysis **3**
2.1 A full record in one table cell 4
2.2 Keep a journal of changes to records 6
2.3 Indexes 7
2.4 Pointer files 8
2.5 The 6.5.0 pointer file structure 9
2.6 Record locks 11
2.7 Specific rights per record in separate table 12
2.8 ISO date tables 13
2.9 A single table for counters 13
2.10 Examples 13

3 Remarks **19**

1 Database platforms compared

From Adlib 6.0, you can approach SQL Server and Oracle databases through Adlib, aside from Adlib's proprietary .CBF format. To this end, you can have your Adlib databases converted to the other format.

Although the Adlib CBF database format is very well suited for our applications, there are also advantages in using SQL Server or Oracle. These are for example:

- The format is supported all around the world.
- The client/server implementation allows for faster multiple-user access.
- In SQL Server and Oracle-databases, your data is very safe. For example, these types of databases cannot be opened and read from within the Windows file system.
- SQL Server and Oracle database cannot be removed if a user hasn't got the right authorization.
- SQL Server/Oracle also has replication advantages, because you can set up such a database to automatically copy every change that a user enters into a record and saves, to a backup database elsewhere.

SQL Server and Oracle are so-called relational databases. A relational database consists of a collection of tables, with in them usually a (very) limited number of field columns, in which the user can search for data, or harvest and rearrange data in many different ways in so-called views or reports, without having to reorganize those tables. So the structure of such a database is relatively simple. But that has its disadvantages too:

- The occurrences (field repetitions) for example, that occur often in an Adlib CBF file, have no similar equivalent in even a minimally normalized relational database. (Normalization is the efficient organizing of a relational database by following a number of guidelines, like removing redundant data in tables, by splitting up those tables. For a relational database this save disk space and makes sure that data is stored logically. Do note however, that Adlib CBF files have practically no redundancy, and already need only a minimum of disk space by default.)
  The normalization that usually is being applied to relational databases, can lead to hundreds or thousands of tables, for complex linked data with field repetitions like in museum or archive data-

bases. And all those tables have to be rejoined during runtime, which uses a lot of memory and processor time.

- Another disadvantage is that there exists a very explicit link between a relational database and the programs that use it. In practice this means that changes to a table structure, usually also means adjustments en recompilation of the software, while that is not necessary for Adlib-databases: for the latter you can change the structure of a database, adjust the graphical interface if necessary, and directly start the new application.

The joining of these two technologies has led to the possibility to approach (Adlib specific) relational databases, from Adlib 6.0. After all, we want to keep using largely the same software, the same user interface and the same maintenance programs (Adlib Designer), whilst we do want to offer our customers the advantages of SQL Server or Oracle but not the disadvantages.
That is why we chose to once-only convert Adlib databases (filled or empty) to a relational database, if the customer would like to use this functionality. (Note that its use and support for it, cost extra. Our sales department can tell you more about it.)
After that conversion, the Adlib databases will have been converted to as many tables in one relational database. Also, each index file, word list, pointer file, lock file and cnt file will have received its own table (the latter three from 6.1.0). For the conversion to the new format, pointer files are structured into XML, which allows for greater exchangeability, and better readability at table level. A whole record in a CBF file will have been inserted in one table cell as an XML document, accompanied by the record number in another column. This XML document was hexadecimally encoded in SQL Server 2000 (as a so-called BLOB: Binary Large Object), because only printable characters were allowed in a table cell. From SQL Server 2005 (and after a possible new conversion of your database for compatibility with Adlib 6.5.0) this is history, and each record will be saved as a readable XML document in the (Adlib SQL or Adlib Oracle) database, and will be displayed as such in database server management programs (like Microsoft SQL Server Management Studio Express), which allows database managers to more easily manage and analyze such a database. With SQL (Structured Query Language, the default interface for relational databases) and/or the SQL Server Enterprise Manager or SQL Server Management Studio Express, you will be able to search index tables and word lists (for instance for integrity checks and possibly repairs), as well as the records themselves (as long as the XML documents aren't hexadecimally encoded). However, we do strongly advise against editing your data on this level. So normally, you'll need an Adlib application and software to be able to edit, search and fill your new SQL Server or Oracle database.

2 SQL database analysis

After installation, setup and database conversion (see the *Installing Museum, Library and Archive* manual), you may open, in a/o Microsoft's SQL Server Management Studio (Express) or SQL Server Enterprise Manager, an Adlib SQL Server database, and in the Oracle Enterprise Manager Console an Adlib Oracle database, if you would like to see the structure or contents of it – in general we advise to not make changes in the database this way. (For the purpose of repairing damaged files by Axiell employees, this advice may be ignored.)
In SQL Server Management Studio Express for instance, you open a table by opening your database folder, click the *Tables* subfolder, and in the left or right window pane right-click the desired table, and in the pop-up menu choose *Open table*: all rows will be retrieved. In SQL Server Management Studio (2008) you right-click a table and click *Select top 1000 rows* to retrieve the first 1000 rows, for example.
In Microsofts SQL Server Management Studio (Express), right-click the folder of your database in the left window pane and choose *New query* in the pop-up menu to be able to enter an SQL query for that database, or click the *New query* button in the toolbar for a the selected database. (In the yellow bar at the bottom of the window you can always see for which database you are currently executing a query.) A query entered in the middle window pane can be executed by clicking the *Execute* button. SQL allows you to research the structure and contents of each table.

> SQL (Structured Query Language) is somewhat comparable to the Adlib expert search language. For instance: `SELECT * FROM collect` means that you want to retrieve all records from the *collect* table. And `SELECT count(*) FROM thesau_term` for example, counts the number of terms in the *term* index of the thesaurus.

In an Adlib SQL Server database, the name of a table indicates its contents. A single name like *collect*, is a former Adlib database. A double name, in the format *databasename_indexname*, indicates an individual index. If the second half of such a double name is literally equal to or begins with "*pointerfiles*", then it concerns a table in which the pointer files for the relevant SQL main table (ex-Adlib database) are stored.

Note that SQL queries on tables that represent former Adlib databases can be complex, since each record has been put in its own, single field. It's easier to search tables of Adlib indexes, although many indexes only contain record numbers (for indexed link reference fields). So, amongst others, you have to know how Adlib stores and processes

links, and what the difference is between term indexes and the word-list for example, to be able to understand the contents of and the relation between the different SQL Server tables. See the Designer Help and/or the *Adlib software functionality profile* document for all information about these mechanisms.

2.1 A full record in one table cell

So, of an opened database the contents of a record are saved and displayed in one field. The record contents in this field are structured in XML. The priref, the creation date and modification date are in separate columns, but are also attributes of the XML *<record>* element (the latter is convenient for a possible export of XML records), namely as follows:

```
<record priref="nnnn" creation="yyyy-mm-ddThh:mm:ss" modification="yyyy-mm-ddThh:mm:ss">

   <field tag="tt" occ="nn">data for field</field>
   <field>.....</field>
    ....

</record>
```

From 6.6.0, the creation and modification date and time, and the name of the current user can automatically be stored per field as metadata in an edited record as well, if the database has been set up for this via the *Store modification history* option in Adlib Designer. You don't need to make any settings to activate this functionality. To be precise: this metadata will be stored per edited field occurrence per data language, as attributes of the field node in the XML. For example:

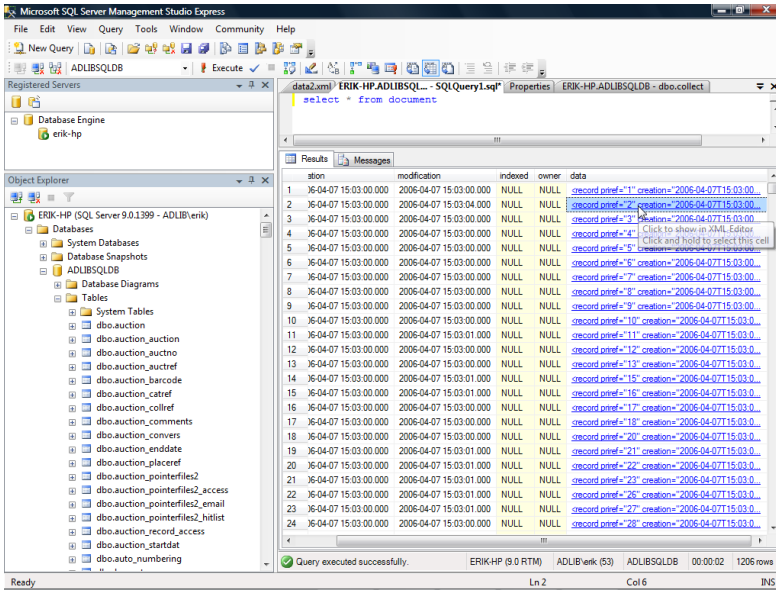
```
<field tag="T9" occ="2" cd="2011-04-11T11:26:51" cu="erik" md="2011-04-11T11:26:51" mu="erik">Bronski House journey back to Poland</field>
```

The attributes have the following meaning: `cd` stands for creation date, `cu` means creation user, `md` modification date and `mu` is the modification user.

You can't show this metadata in your adlwin.exe application, but via Designer you can create indexes on the metadata and define access points for them, so that you'll be able to search for records with fields that have been changed after a certain date and/or by a particular user.

Retrieving records

For instance, retrieve all rows from the *document* table, via: `select * from document`. In SQL Server Management Studio Express for example, the result is presented as follows:



Each row contains one record. The *priref* column contains the record number, *creation* the creation date and *modification* the last modification date of the record. The *data* column contains the records in their entirety. The data is directly readable in SQL Server 2005 or higher when you are working with Adlib 6.5.0 or higher.

> In SQL Server 2000, records are encoded hexadecimally (indicated by 0x) here. (Note that in the SQL Server Enterprise Manager the *data* column only indicates that its content is binary, and doesn't display the actual contents like here in SQL Server Management Studio Express.) In principle you can (programmatically or via a handy conversion website like Translator, Binary) convert the complete hexadecimal string to ASCII. Every two alphanumerical characters in here represent one ASCII character. *3C* for example, is equal to decimal 60, and the 60th ASCII character is "<", the start of an XML tag. And hex *72* for instance, translates to "r". So every XML record starts with *<record* (hexadecimally: 3c 72 65 63 6f 72 64)

Thus, should you desire to build your own application as the interface for this data, instead of just using your Adlib applications, then you

can process each retrieved record as XML. To be able to actually find certain records, you will have to use the index tables which exist for many fields.

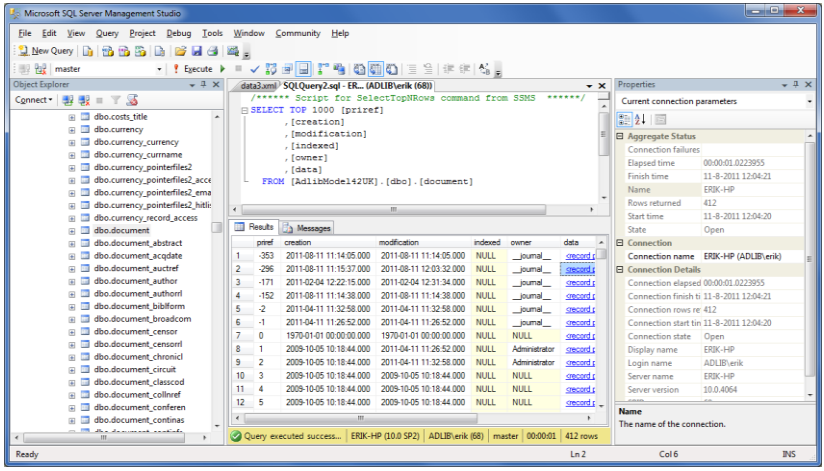
2.2 Keep a journal of changes to records

From 6.6.0 you can let Adlib log all saved changes in records, if you are using an Adlib SQL Server or Adlib Oracle database. This allows for easy backtracking to see who, when and which changes have been entered, or you can use the change log to find and restore the original data when incorrect data has been filled in.

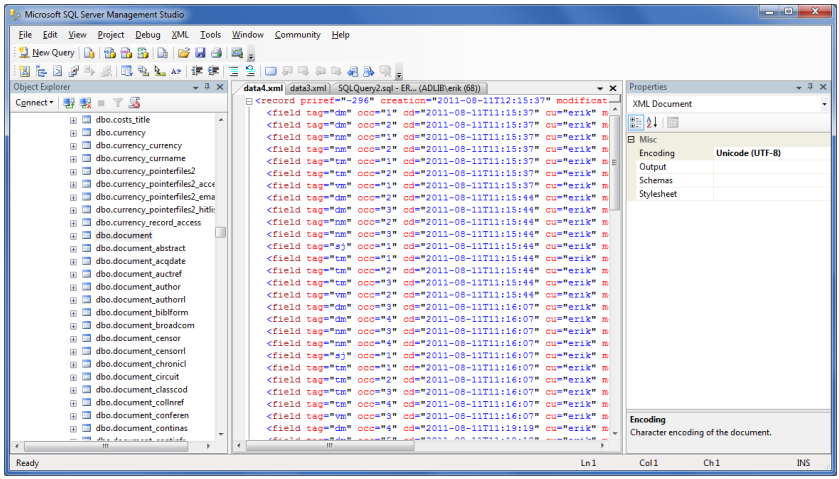
After this functionality has been set for an Adlib database (see the Designer help for more information), for every record which has been edited by a user an extra record will be created in the relevant Adlib database table in the Adlib SQL Server or Adlib Oracle database, and this extra record will have the negated number of the original record number: for example, the first change in a record with the number 171 causes the creation of a record with the number -171. In this "negative" record, all* changes in the record will be logged from now on: each data change will be saved in a new field occurrence in this record. Note that the size of your database will increase substantially.
* Only changes in linked fields and any merged-in fields won't be logged.

Viewing the journal record in XML

You cannot view the journal record itself in your adlwin.exe application, but you can in your SQL Server management software if necessary.



In this example you can see several negative records with the owner "_journal_". Click the underlined data to open it in a separate tab. You will see the stored XML in this journal record.



2.3 Indexes

In Adlib SQL (and Adlib Oracle), indexes are stored in separate tables. From Adlib 6.5.0, an index table has eight columns. Per index key there is one row in such a table. The columns are:

- *Term* contains the key on which you can search (although the *term* field may contain linked-record numbers). The indexed values in this column are stored without any accents and entirely in lower case.
- *Display term* contains the indexed term as it has been entered by the user, including any accents and capitals.
- *Priref* contains the record number of the record from which the indexed term originates.
- *Domain* contains the name of the domain to which a term may belong. (From Adlib 6.5.0, the double colon and the domain no longer appear in the *Term* column.)
- *Strippedterm* contains the indexed term but without any punctuation marks like hyphens, quotes and comma's.
- *Dmp_primary (primary key)* may contain the primary phonetic code, as determined by the DoubleMetaphone algorithm.

- *Dmp_secondary (secondary key)* may contain the secondary phonetic code, as determined by the DoubleMetaphone algorithm.
- *Language* contains the language in which an index value has been entered, if the field contents are multi-lingual.
  In multi-lingual Adlib SQL and Oracle databases, until 6.5.0 all translations of multi-lingual field contents were indexed without language indication, which made it impossible to make a language-specific search for some term. From 6.5.0, term and word indexes contain this extra column which makes searching in a specific data language possible.

With these additions in Adlib 6.5.0, the size of SQL indexes may have increased sixfold. Still, it allows for faster searching because Adlib can directly use the proper index column without having to apply any intermediate processing.
In the future, you'll be able to indicate explicitly how Adlib should search a term index on a submitted term, for instance in the *Stripped* form, or phonetically or in a specific language. It is possible in 6.5.0 to search in a specific language via the *Expert search language*, and search phonetically via the *Search wizard*, but the other search possibilities have not been implemented yet.

2.4 Pointer files

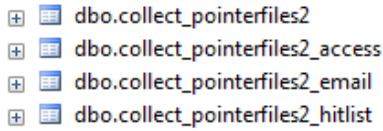
For Adlib 6.4.0 or older, there is one <*database table*>*.pointerfiles* table per database table. Each pointer file in it is described in its own row of the table. The available columns are: *owner* (maker of the pointer file), *title*, *number*, *selectionstatement* (the search statement or selection), *hitcount* (number of results), *modification* (date on which the pointer file result was updated last), and *data* (contains this pointer file as an XML document).

As a result of testing with large pointer files (with more than 100.000 hits) it was shown that the 6.3.0 implementation of pointerfiles performed insufficiently in the SQL based versions of adlwin (this applies both to the Microsoft SQL Server and the Oracle implementations). Retrieving a pointerfile of over 100.000 hits could easily take over an hour of processing time. The Adlib proprietary database implementation performed much better (30 seconds for the same pointer file).

To solve this problem in 6.5.0, the pointerfile format in the SQL implementation of Adlib has radically changed, and the pointer file reading and writing algorithms have been rewritten. After the required conversion procedure with the upgrade to Adlib 6.5.0 or higher, the modifications have been processed in your database; for users, the change makes no difference.

2.5 The 6.5.0 pointer file structure

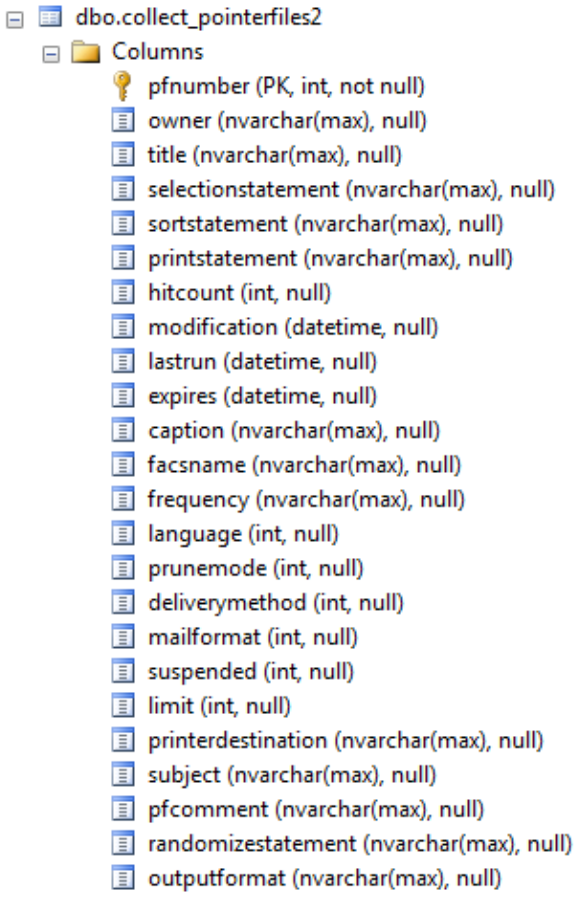
In the 6.5.0 pointer file structure, each pointer file is stored in four separate tables: a table for the pointer file header, a table for the pointer file hitlist, a table for the e-mail addresses for any SDI component of the pointer file and a table for the access rights of the pointer file. All four tables have the same "base name" that consists of the name of the Adlib database with the extension "pointerfiles2":



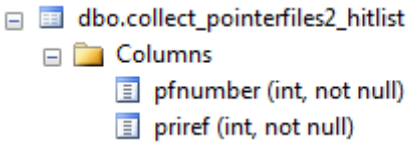
The extension *pointerfiles2* was chosen deliberately to make a distinction between "old" style pointer files and "new" style pointer files. The result of this is that "old" pointerfiles will be visible in pre-6.5.0 release versions of adlwin and the "new" style pointer files will only be visible post 6.5.0 release versions of adlwin.

The pointer file header table structure is as shown in the figure below.

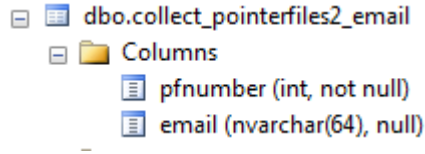
Note that all pointer file header data elements (including SDI data) are now expressed as separate columns. Also note that text columns of indefinite length use the `nvarchar(max)` feature of SQL Server 2005, which implies that Adlib 6.5.0 can no longer be used on SQL Server 2000.



The pointer file hitlist table structure is as follows:

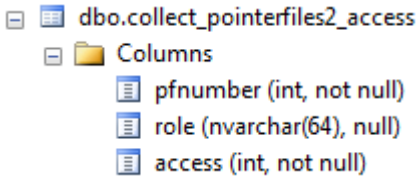


The hitlist table structure is very simple, it consists of rows of just two numbers: the pointer file number and the priref of the record in the pointer file.
The other two tables are just as simple. The pointer file e-mail list table structure is as follows:



(The e-mail table is a separate table because each pointer file can have multiple SDI e-mail destinations.)

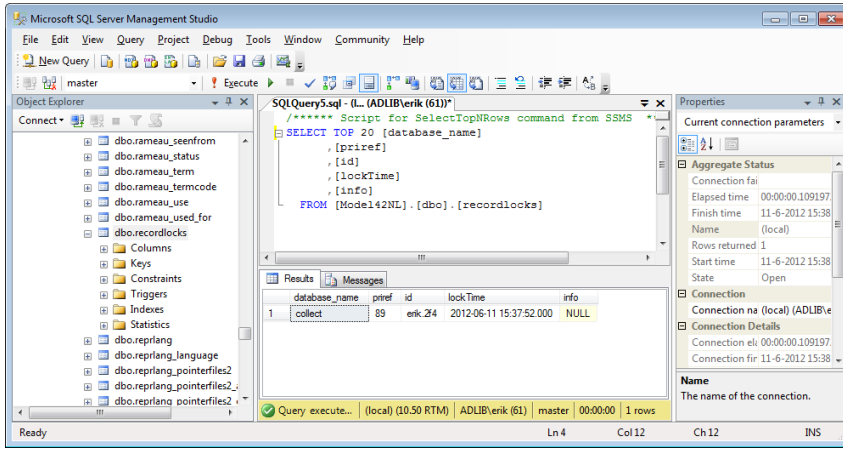
The pointer file access rights table structure is as follows:



The access table *access* column of a pointer file may contain the following values: 0 (undefined access rights), 1 (none access rights), 2 (read access), 3 (write access), 4 (full access).

2.6 Record locks

There is one *recordlocks* table for the entire SQL database. Each row in this table describes one record lock: the name of the database table to which the lock applies, the record number of the locked record, the lock id and the time of locking. Retrieve all rows from this table to see which locks are present at the moment.

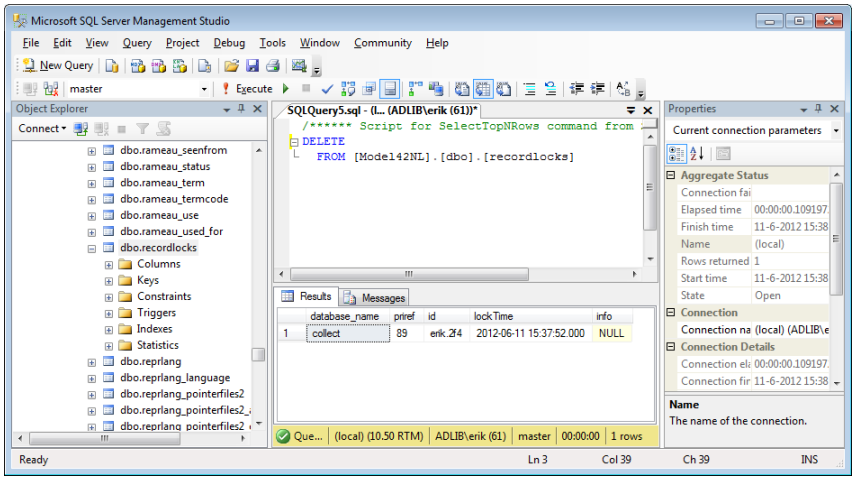


Normally, record locks exist only while a record is being edited, to prevent someone else from editing the record at the same time.

However, if the computer ever crashes, one or more record locks may be left behind in the database, after which those records can not be edited by anyone anymore. Then preferably use Adlib Designer to remove the remaining record locks in a user friendly way. In MS SQL Server Management Studio you can do this as well though, in the *record locks* table. In the opened *record locks* table you can remove all record locks at once via the following SQL query:

```
DELETE FROM [<my-database>].[dbo].[recordlocks]
```

Replace `<my-database>` by the actual name of your database, without sharp brackets. Make sure that nobody is working in Adlib when you execute this statement.



If currently people are at work in the database, and therefore you only want to remove one specific remaining lock, then do this with:

```
DELETE FROM [<my-database>].[dbo].[recordlocks] WHERE
priref = <record-number>
```

Replace `<record-number>` by the actual record number, without the sharp brackets.

2.7 Specific rights per record in separate table

Specific access rights per record, which have been made possible via the *Rights*, *Authorisation type* option (see the Designer Help for more information about this type of data protection), will be stored in a separate table from 6.5.0. The new table has three columns: *priref* (repeatable), *role* and *rights*.

The advantage is that Adlib no longer has to read a record to find out whether the current user has the proper access rights for the intended action, which improves performance of Adlib.

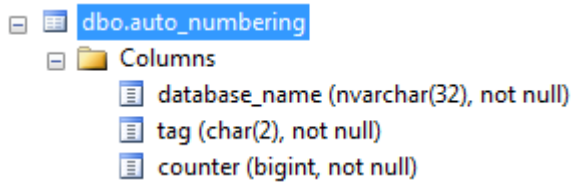
2.8 ISO date tables

The 6.5.0 structure of ISO date indexes contains the dates entered by the user (complete or incomplete) in a column named *displayterm*, and a *term* column which will always automatically contain complete ISO dates.

With 6.5.0, the data type of the term column has been changed from a term index (SQL) or nvarchar2 index (Oracle) to a float resp. number index to allow for especially large dates (beyond - or + 9999-01-01) in your data.

2.9 A single table for counters

In your Adlib SQL or Adlib Oracle database, there's a separate table called *dbo.auto_numbering* which contains all counters for all tables, with automatically numbered fields as well as the named counters introduced in 6.6.0 (see the ADAPL `GetCounter` function).



There are three columns in this table:

- The *database_name* column holds the names of any named counters and Adlib database names for automatically numbered fields. Because the named counters are stored in an existing database table structure, the name of a named counter is saved in the not quite appropriately named *database_name* column. The advantage of using the existing structure is that you don't need a database conversion to apply named counters.
- The *tag* column contains field tags of any automatically numbered fields.
- The *counter* column holds the most recently assigned counter values.

2.10 Examples

Suppose you are looking for all records in the *document* table, in which the author field holds the name `Fruithof, Th.` In Adlib, records of names of persons and institutions are stored in the *people* file.

The names themselves are in the *name* field. For this field an index is available so that Adlib will be able to quickly search for names, in this case the index is the *people_name* table. From a record in *document* there is a link to this field, via a link reference: only the record number of the linked name record will be saved in a documentation record. So you'll first have to search *people_name* for the record number of `Fruithof, Th.` Because domains occur in this index – domains usually appear in *thesau* and *people* and their indexes – and you are looking for the author `Fruithof, Th.`, you have to execute an SQL statement similar to the following: `select * from people_name where term like 'fruit%' and domain='author'`. In our example database one record is found, for the relevant author, with record number 20. So it is this record number that has to occur in the index of the field in *document* in which authors are stored: *document_author*, since in there only the record number of the name record occurs (20, in this case), not the searched name itself. With the following query you can find out in which documentation record the name record 20 (author Fruithof) appears: `select * from document_author where term = '20'`. In our example this turns out to be record 1 (in *document*, as stated). You can retrieve this record via e.g.: `select * from document where priref = '1'`.

With somewhat more advanced SQL statements it's also possible to execute these steps all at once (or to enter more complex queries), for example:

```
SELECT document.priref, document.data
FROM document
INNER JOIN document_author ON document.priref = docu-
ment_author.priref
INNER JOIN people_name ON document_author.term = peo-
ple_name.priref
WHERE people_name.term like 'fruit%'
and domain='author'
and (document.priref >= 0 and document.priref <= 500)
order by document.priref
```

Via the inner joins the keyfields are indicated with which the tables, in which you want to search, are linked together in Adlib. In this query are also specified the priref limits of the dataset you want to search, and a sort criterium.
A similar example for searching on a certain title in the same dataset (here *Books*), for which you only want to retrieve record numbers, can look as follows:

```
SELECT distinct document.priref
FROM document
INNER JOIN document_title ON document.priref = docu-
```

```
ment_title.priref
INNER JOIN wordlist ON document_title.term = word-
list.wordnumber
WHERE wordlist.term like 'w%'
and (document.priref >= 0 and document.priref <= 500)
order by document.priref
```

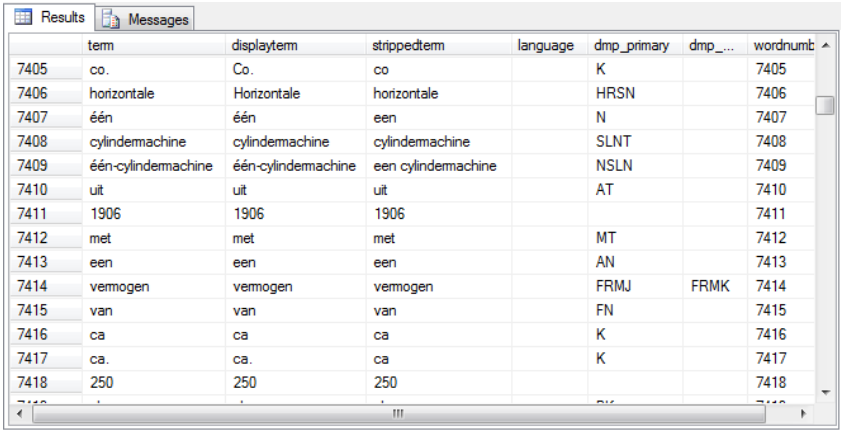
Some other examples of handy queries:

- select all document records in which a field with occurrence 5 appears: `select * from document where data.exist ('/record/field[@occ="5"]') = 1`
- select all document records in which a field with tag 'ti' appears: `select * from document where data.exist ('/record/field[@tag = "ti"]') = 1`
- select all document records in which a field with the 'creation' attribute appears: `select * from document where data.exist ('/record/field[@creation]') = 1`

■ Using the wordlist

Noteworthy here is the special *wordlist* index. The *wordlist* contains all unique words that occur in all word-indexed (free text) fields. These are fields with long texts, such as titles or descriptions.

There is only one *wordlist* for all main tables together. Which words occur in this file can easily be discovered via the SQL statement: `select * from wordlist`.

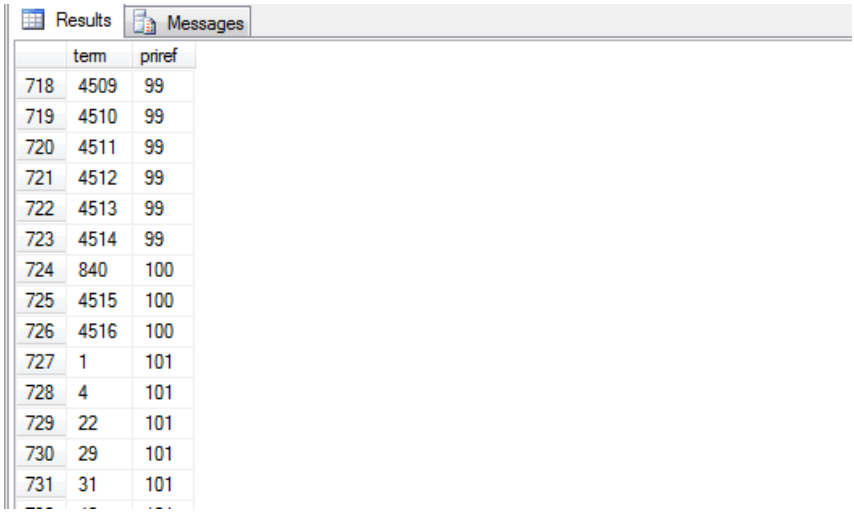


Results | Messages

	term	displayterm	strippedterm	language	dmp_primary	dmp_...	wordnumb
7405	co.	Co.	co		K		7405
7406	horizontale	Horizontale	horizontale		HRSN		7406
7407	één	één	een		N		7407
7408	cylindermachine	cylindermachine	cylindermachine		SLNT		7408
7409	één-cylindermachine	één-cylindermachine	een cylindermachine		NSLN		7409
7410	uit	uit	uit		AT		7410
7411	1906	1906	1906				7411
7412	met	met	met		MT		7412
7413	een	een	een		AN		7413
7414	vermogen	vermogen	vermogen		FRMJ	FRMK	7414
7415	van	van	van		FN		7415
7416	ca	ca	ca		K		7416
7417	ca.	ca.	ca		K		7417
7418	250	250	250				7418

In this index you can see that each term has a word number. And just like with linked fields, in long-text fields only the word numbers are stored, not the literal text, again to save disk space. You can observe this in the following query:

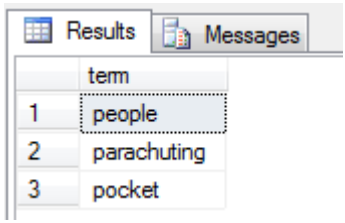
```
select top 1000 term, priref from document_title
```



	term	priref
718	4509	99
719	4510	99
720	4511	99
721	4512	99
722	4513	99
723	4514	99
724	840	100
725	4515	100
726	4516	100
727	1	101
728	4	101
729	22	101
730	29	101
731	31	101

From record 100 for example, three words have been indexed, with the word numbers 840, 4515 and 4516. With the following query, for instance, you can retrieve these words from the *wordlist* (the words from record 100 which have been indexed in the title index in the document database – in SQL these are both tables of course):

```
select wordlist.term from wordlist inner join docu-
ment_title on wordlist.wordnumber = document_title.term
where document_title.priref=100
```



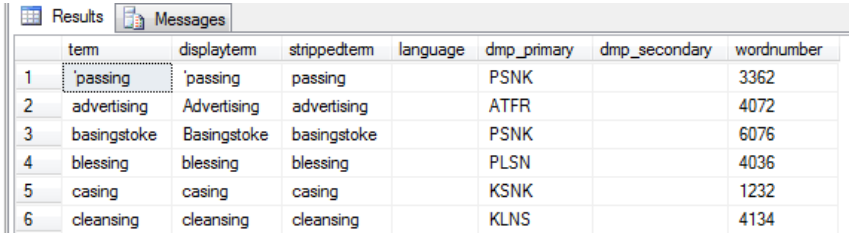
	term
1	people
2	parachuting
3	pocket

To retrieve all words from the title index for a set of records (99-101), and each word accompanied by the record number of the source document record while sorted on term, you would use the following query:

```
select wordlist.term,document_title.priref from wordlist
inner join document_title on wordlist.wordnumber = docu-
ment_title.term where document_title.priref in (99,100,101)
order by term
```

Another approach is the following: if you are looking for documentation records of which the title contains the word "`jungle`" for example, then you'll first have to retrieve its word number, for instance via: `select * from wordlist where wordlist.term = 'jungle'`. In our database, this word has number *49054*. Now you have to look up this number in the title index of *document*: `select * from document_title where document_title.term = '49054'`. In our example only one record number is found: *655*. And finally we can retrieve this record from *document*, via: `select * from document where document.priref = '655'`.

You can search the *wordlist* truncated as well, via the % character. For example: `select * from wordlist where wordlist.term like '%sing%'`



Results | Messages

	term	displayterm	strippedterm	language	dmp_primary	dmp_secondary	wordnumber
1	'passing	'passing	passing		PSNK		3362
2	advertising	Advertising	advertising		ATFR		4072
3	basingstoke	Basingstoke	basingstoke		PSNK		6076
4	blessing	blessing	blessing		PLSN		4036
5	casing	casing	casing		KSNK		1232
6	cleansing	cleansing	cleansing		KLNS		4134

In a normal *wordlist*, each word appears only once per data language (or without data language). If the *wordlist* has become corrupted for whatever reason, then words may appear more than once per data language. Such corruption must be repaired. One of the ways to do this is to remove all duplicates from the *wordlist* manually, after which you could use the Indexcheck tool to repair all word indexes (by means of a specific list of all word-indexed tags per Adlib database). This way, the *wordlist* will only be repaired where it is corrupted, and ik won't need to be rebuilt. With the following query you can find all duplicates in the *wordlist*:

```
select wordlist.term, count(*) as WordCount from wordlist
group by wordlist.term having COUNT(*) > 1 order by term
```

As long as a term doesn't appear more than once for the same data language, or as long as a term doesn't appear more than once without data language, everything's fine. In the other case you must delete the redundant duplicates. For example, if you want to remove the word with word number 186, then execute the following query:

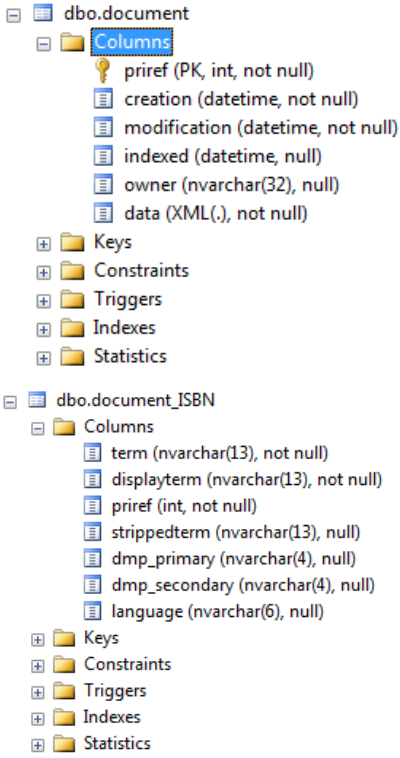
```
delete from wordlist where wordnumber = 186
```

As mentioned, you must now repair all word indexes in which that word (really the word number) appears, using Indexcheck. Do make sure first that no-one is working in Adlib at the moment.

If possible, see the *Adcopy, Indexcheck and Linkrefcheck* manual (not available for download) for more information about finding and repairing data corruption.

■ The columns of a table

Further note that the left window pane of SQL Express Manager 2005 (the tables list) also displays information about the fields (columns) which are present in the relevant relational table, about the PK (primary key) on which the table can be linked, and the type of the variable of said fields, for instance:



3 Remarks

- SQL Server Management Studio Express may also be run from the network, by connecting to the SQL server through a remote desktop connection.

- SQL Server keeps information for internal purposes in *sys…* tables. These files may never be altered manually.

- Through the pop-up menu (right-click your SQL Server database, you may create or schedule backups, or execute a recovery through *Restore*.

- Search results in SQL Server Management Studio can be stored in .csv or .txt format, if desired, by right-clicking the result table and choosing *Save result as* in the pop-up menu.

- If you wish you change the structure of a database through Designer, do first make a backup of your *\data* folder (the *.inf* files) and of your SQL Server database (or Oracle database), and then take e.g. the Adlib SQL Server database offline so that no-one can change or enter records. When you're done changing the database structure, you'll have to bring the database online again.
  In SQL Server Enterprise Manager you can do this via the *Take offline* and *Bring online* options in the pop-up menu of the database. In SQL Server Management Studio Express these "database state" options are hidden, and there is only the read-only *Database state* option available on the *Options* tab of the *Database properties*.
  You can take a database offline via a Transact-SQL statement using e.g. `sp_dboption` procedure. For instance, to take the *AdlibSQL* database offline, you execute the following statement: `sp_dboption 'AdlibSQL','offline','true'`. Before you do this, make sure there are no active processes using the database. You can make the database online again by executing the following statement: `sp_dboption 'AdlibSQL','offline','false'`.