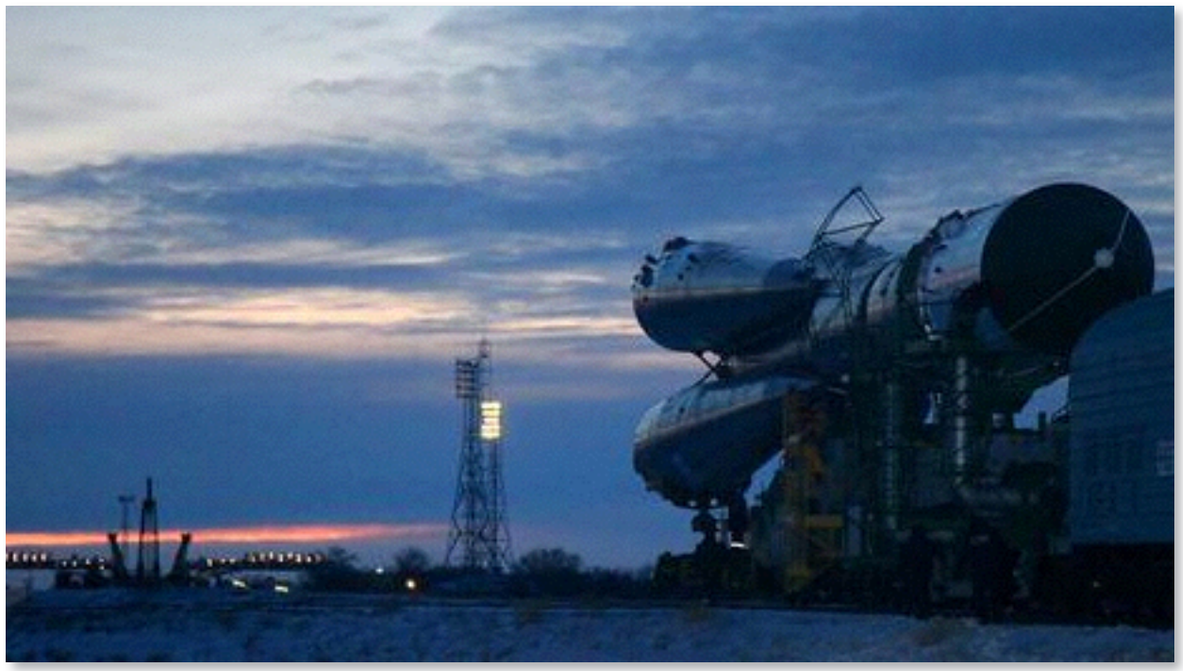
presented by 4D

データ変換

2014-06-23

目標: 4D 2003/4からv14にデータベースをアップグレード



FAQ

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: アップグレードは難しいのでしょうか?

A. このセミナーで紹介する内容をしっかり踏まえて臨めば**大丈夫**です。

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: 途中のバージョンを経由しなければならないのでしょうか?

A. いいえ。

注記: **v13**を経由すれば，より効率的な作業ができますが，必須ではありません。（後述）

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: その場合，中間バージョンのライセンスも購入する必要があるのでしょうか?

A. いいえ。

注記: **v14**は30日間無償で試用することができます。

※2003/4の試用版には，デザインモードで編集できるテーブル・メソッド・フォーム等の数に制限がありました。
v11以降，そうした制限は撤廃されましたが，試用できるのは，その時点で販売中のバージョンに限られます。
つまり，v11の試用は現在できません。

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: 変換後，もう昔のバージョンには戻れないのでしょうか?

A. v14のデータファイルは**v12**まで戻すことができます。

注記: **ストラクチャファイル**は変換を取り消しすることができません。

※v13には「v14データファイルを開く」というオプションがあります。同じように，v12には，「v13データファイルを開く」というオプションがあります（データベース設定）。ただ，初期バージョン（12.1, 13.1など）には，そのオプションがありません。ひとつ前のバージョンの戻った後，データファイルを**圧縮**すれば，そのデータは完全に前バージョンのものとなり，さらにひとつ前のバージョンまで戻すことができます。

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: 変換前にデータのバックアップを作成したほうが良いでしょうか?

A. はい。

注記: 一般的なファイルコピーで十分です。

※v2003/4までのアップグレードとは違い，v11以降の変換では，新規データファイル・新規ストラクチャファイルが作成され，そこにデータベースのコピーが書き込まれます。元のファイルは書き換えられません。したがって，何度でも同じデータベースをアップグレードすることができます。とはいえ，念のため，データファイル・ストラクチャファイルのバックアップを作成することが推奨されています。

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: 変換前にデータの修復を実行したほうが良いでしょうか?

A. はい。

注記: **タグによる修復**はしないでください。

※4D Toolsによる通常の修復は，レコードやテーブルの構造的な一貫性をチェックし，問題があればそれを修正します。修復は，圧縮も兼ねているので，続けて圧縮も実行する必要はありません。一方，タグによる修復は，構造的な問題を修復するのではなく，開けないデータベースが開けるようにするため，データベースの構造を強制的に書き換える緊急手段です。データベースが開けるのであれば，（4D Toolsが提案したとしても）実施は慎重に判断するべきです。

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: サブテーブル/サブレコードは廃止されたのでしょうか?

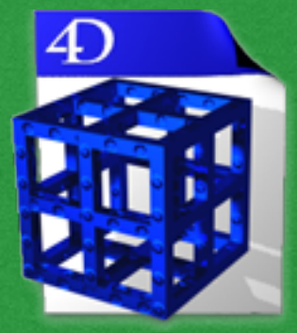
A. いいえ。

注記: 変換前から存在するサブテーブルは使い続けることができます。

※4Dのデータベースエンジンは，高速・大容量・他言語に対応するため，v11で刷新されました。このデータベースは，サブテーブル型のフィールドをサポートしていませんが，サブテーブルについては，標準的なテーブルに変換し，サブテーブルコマンドやサブフォームがこれまでどおり動作するよう，疑似的なサブテーブルとして扱います。ただし，4D v2以前で可能だった多階層のサブテーブルには対応していません。

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: データとストラクチャは別々にアップグレードできるのでしょうか?



A. はい。

注記: catalog.xmlファイルの使用が必要になるケースもあります。

※v11以降のストラクチャとデータは，固有識別子が一致しない組み合わせで開くことはできません。固有識別子は，v2004以前のデータベースを変換したときに新しく発行され，カタログファイルに記録されます。ストラクチャとデータを別々にアップグレードした場合，別々の固有識別子が発行されるため，ふたつを組み合わせて使用することができません。それを防止するには，アップグレード前のストラクチャの隣にカタログファイルを配置します。

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: 新ストラクチャで旧データを開けば自動的に変換されるのでしょうか?

A. はい。

注記: **プライマリーキーアシスタント**が起動するケースもあります。（後述）

※v14以降のジャーナル（ログ）ファイルには，レコード番号の代わりにプライマリーキーが記録されます。プライマリーキーは，ログに加え，4D Mobile，SQL REPLICATE（複製と同期），4DSYNCを有効にするためにも必要です。ログファイルが有効にされているのに，テーブルにプライマリーキーが設定されていない場合，そのテーブルをログの対象から除外するか，プライマリーキーを設定するか，決定しなければなりません。

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: 引き続きShift_JISで日本語を扱うことはできるのでしょうか?

A. はい。

注記: とはいえ**Unicodeモード**は有効にするべきです。（後述）

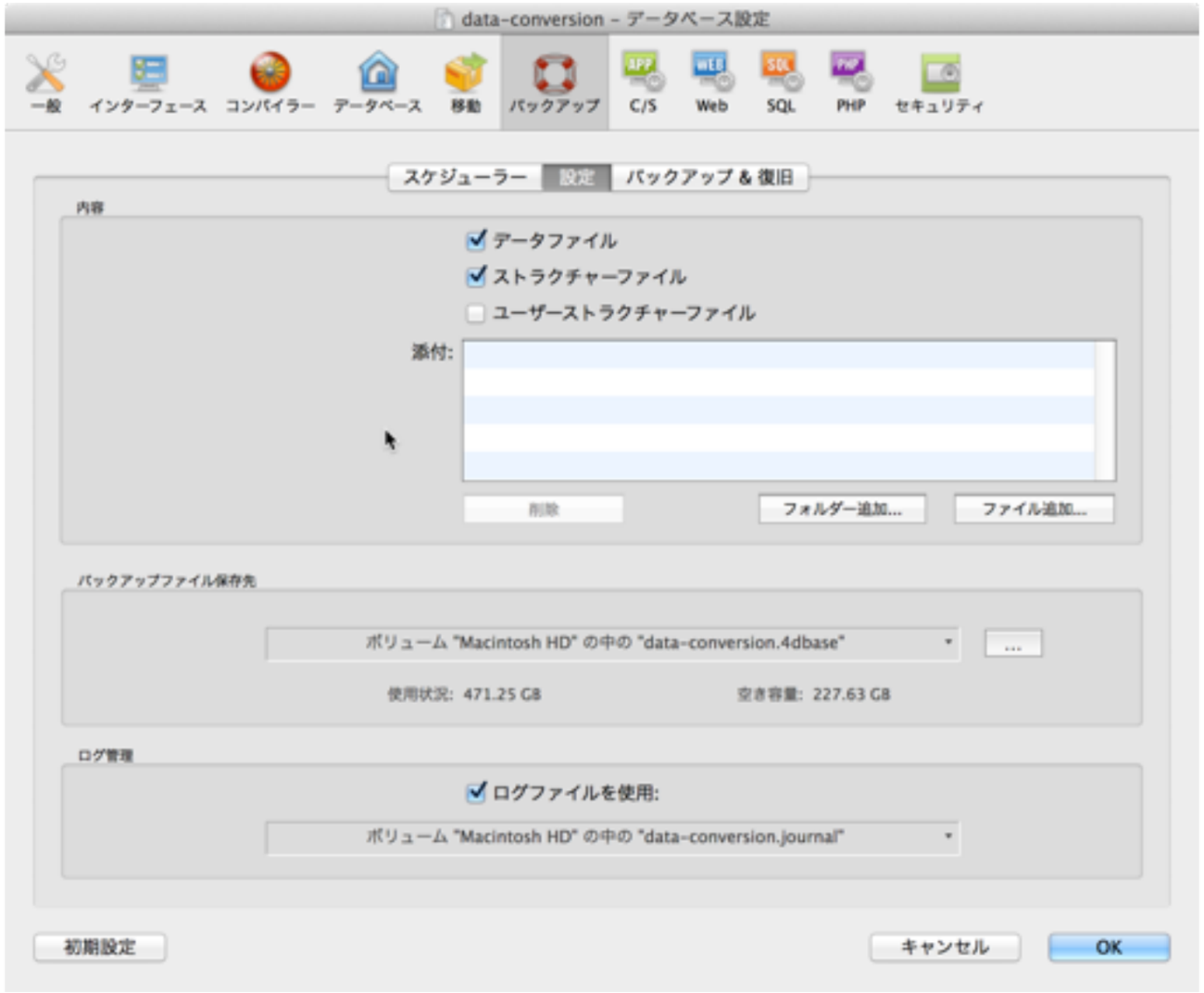
※v11以降，日本語はすべてUTF-16エンコーディングで管理されています。データファイルの内容もUTF-16です。
アップグレード直後，Unicodeモードは無効に設定されていますが，すでにデータはUTF-16に変換済みであることに留意ください。この場合，データは都度4D 2004の共通文字コード（Apple/Microsoft混合）に変換されています。
Unicodeモードを有効にすることにより，Shift_JISをはじめ，さまざまな文字コードが指定できるようになります。

目標: 4D 2003/4からv14にデータベースをアップグレード

PRIMARY KEY

目標: 4D 2003/4からv14にデータベースをアップグレード

基本①: ジャーナルにはデータ更新の履歴が記録されている



更新: 作成，変更，削除，および確定されたトランザクション。

※ログとジャーナルは同義

目標: 4D 2003/4からv14にデータベースをアップグレード

基本②: ジャーナルにはデータ更新がどこよりも先に記録される

①ジャーナルファイル（.journal）

②キャッシュ（RAM）

③データファイル（.4DD）

目標: 4D 2003/4からv14にデータベースをアップグレード

基本③: ジャーナルはデータ復元の有力な手段

データファイル（4DD）

ジャーナルファイル（.journal）

バックアップ（4BK）

目標: 4D 2003/4からv14にデータベースをアップグレード

基本③: ジャーナルはデータ復元の有力な手段

データファイル（4DD）

ジャーナルファイル（.journal）

バックアップ（4BK）

データファイル（4DD）

目標: 4D 2003/4からv14にデータベースをアップグレード

基本③: ジャーナルはデータ復元の有力な手段

データファイル（4DD）

ジャーナルファイル（.journal）

統合

バックアップ（4BK）

データファイル（4DD）

目標: 4D 2003/4からv14にデータベースをアップグレード

基本③: ジャーナルはデータ復元の有力な手段

データファイル（4DD）

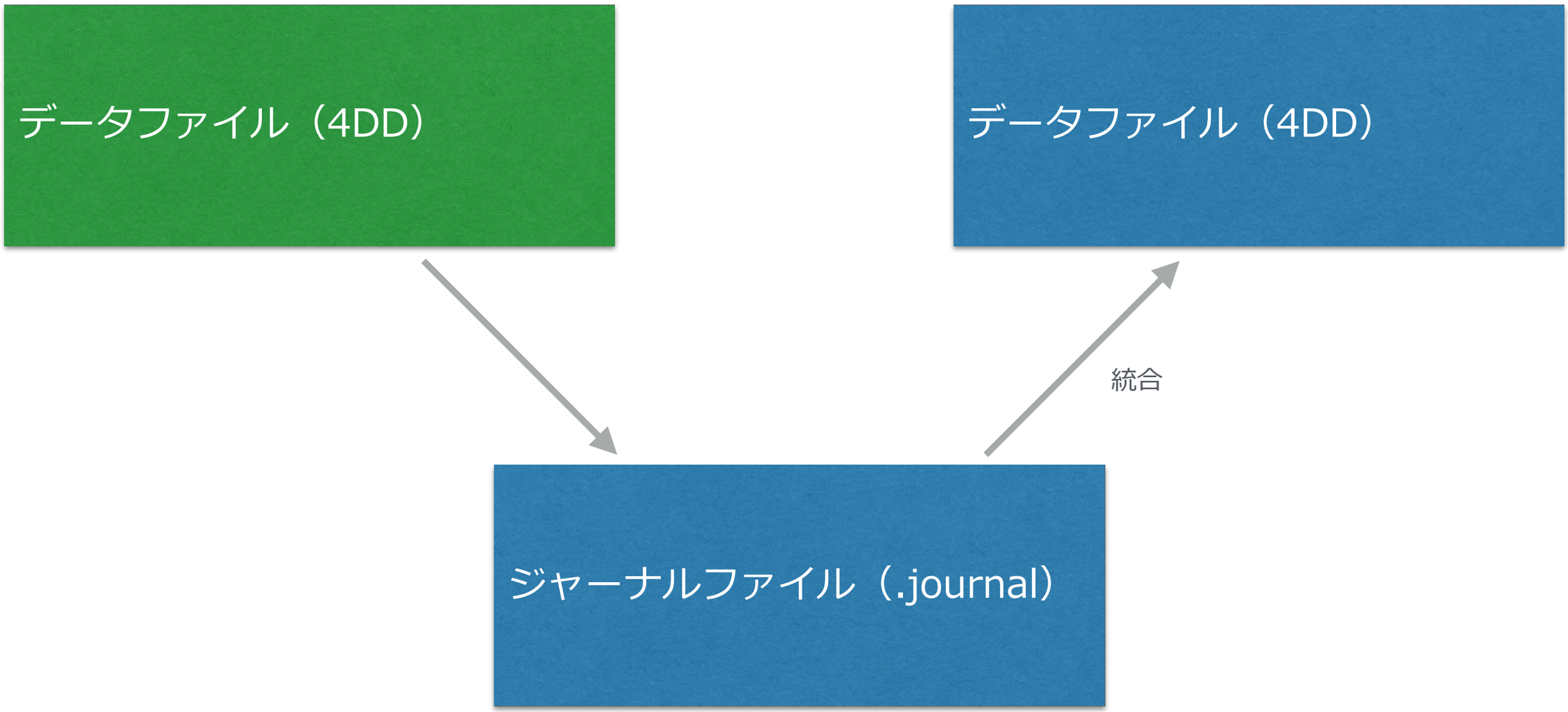
ジャーナルファイル（.journal）

バックアップ（4BK）

データファイル（4DD）

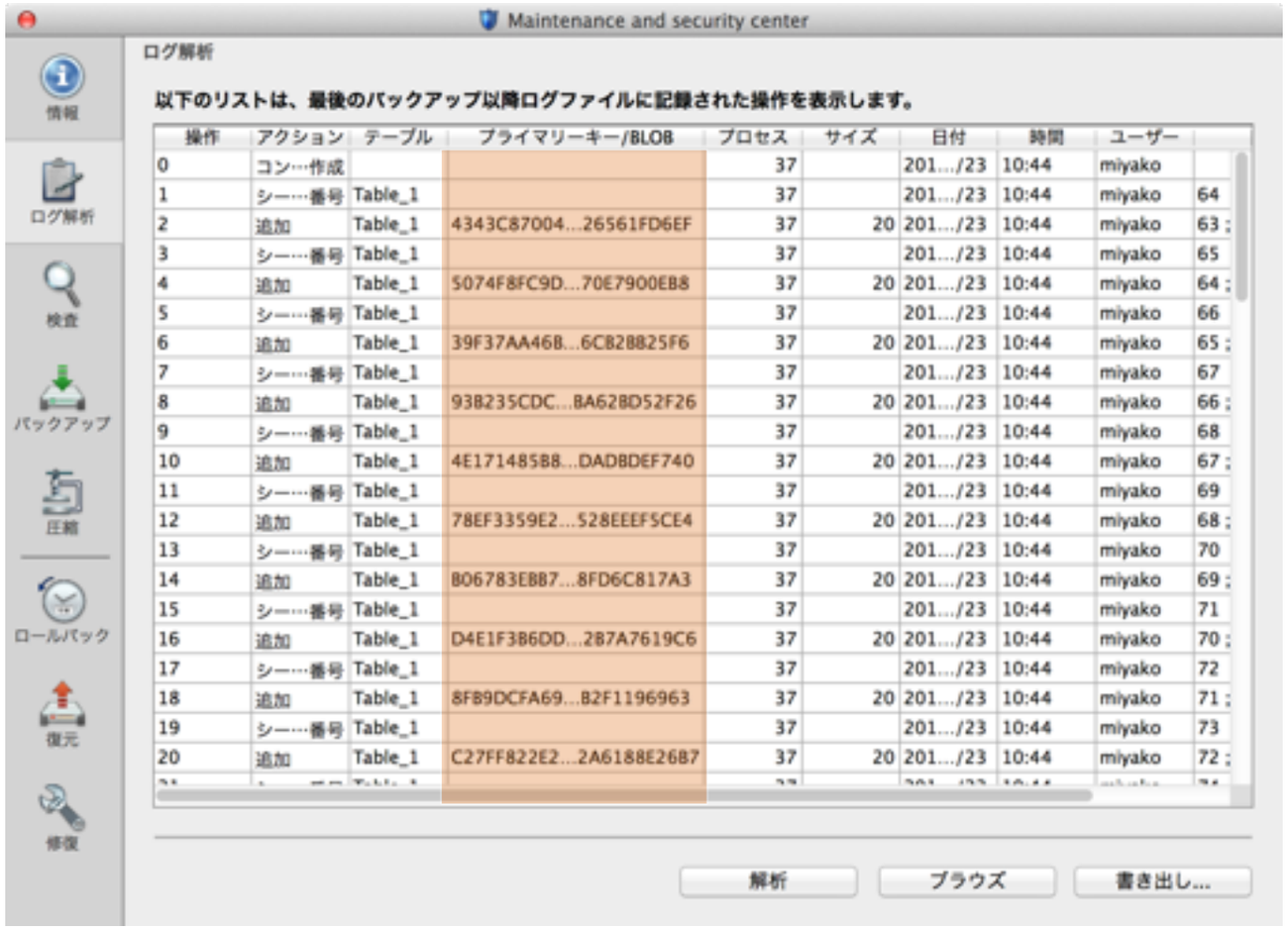
目標: 4D 2003/4からv14にデータベースをアップグレード

参考: ジャーナルはミラーリングにも使用される



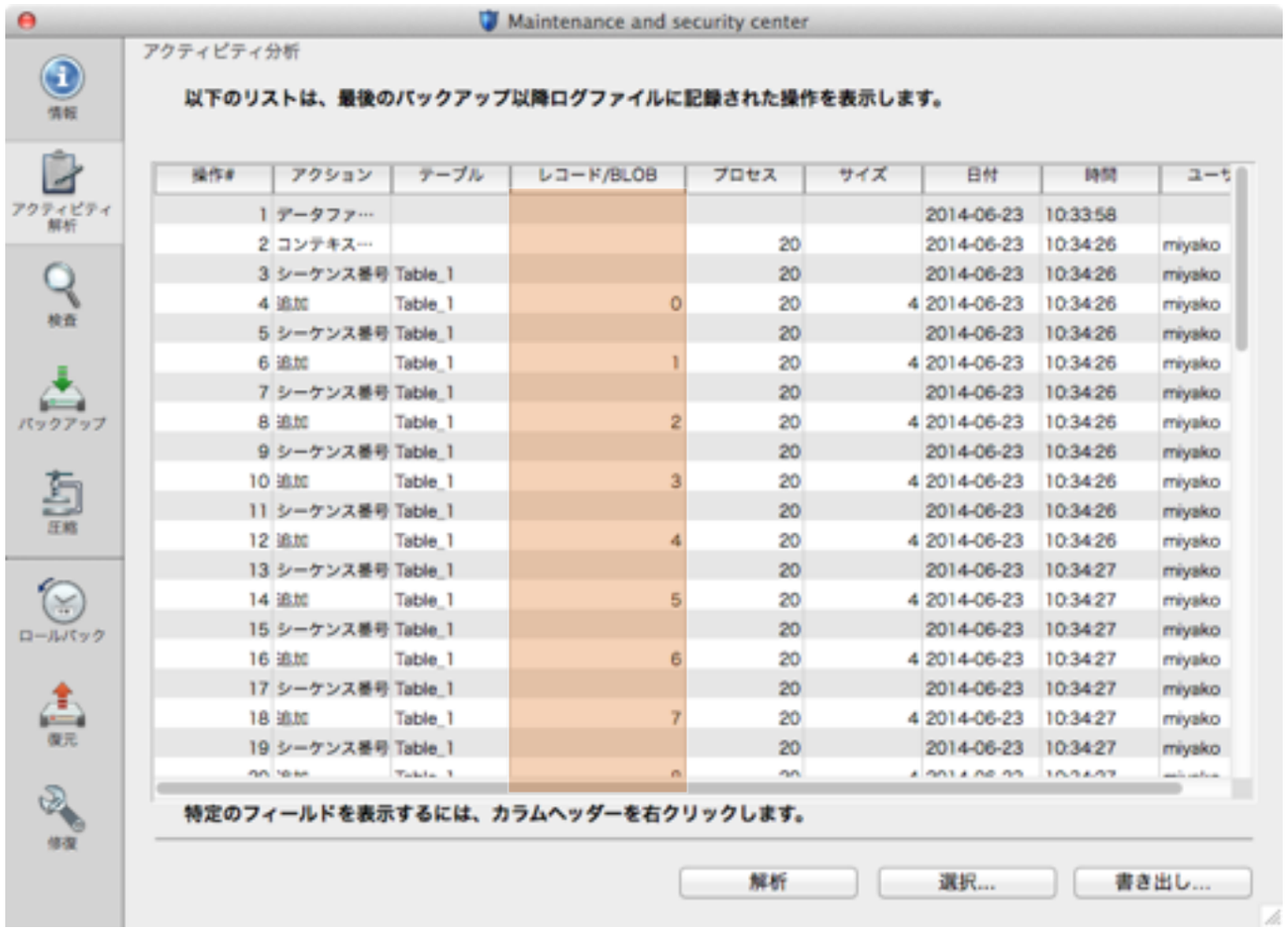
目標: 4D 2003/4からv14にデータベースをアップグレード

基本④: v14のジャーナルファイルには**プライマリーキー**が記録されている



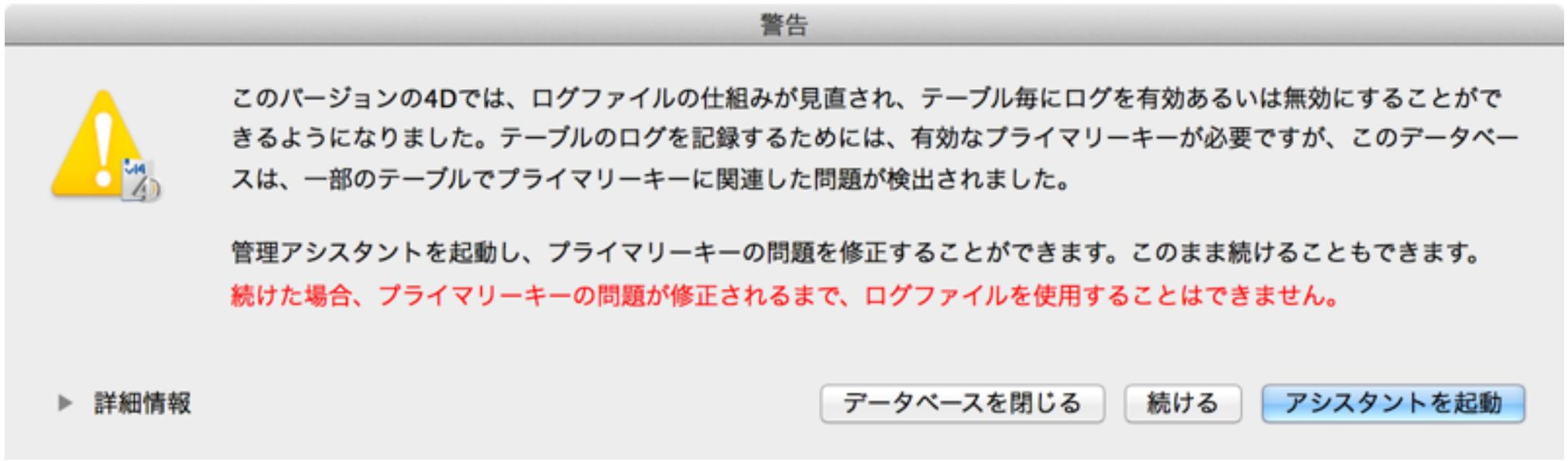
目標: 4D 2003/4からv14にデータベースをアップグレード

参考: v13以前のジャーナルファイルには**レコード番号**が記録されていた



目標: 4D 2003/4からv14にデータベースをアップグレード

基本⑤: v14でジャーナルファイルを使用するためには**プライマリーキー**が必須



目標: 4D 2003/4からv14にデータベースをアップグレード

Q: 既存のIDフィールド（例: 顧客番号）は有効なプライマリーキーですか?

A. いいえ。

注記: プライマリーキーは理論上，重複・NULL・**変更不可**でなければなりません。

※プライマリーキーは，ビジネスロジックやフィールドデータ（入力値）からは独立したものであるべきです。ビジネスロジック上，レコードを一意に特定するIDフィールド（自然キー，ナチューラルキー）が存在するとしても，ジャーナリングの性格上，それとは別にデータベース設計上のプライマリーキー（代理キー，**サロゲートキー**）を設定することが勧められています。

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: 既存の内部IDフィールド（例: GUID）は有効なプライマリーキーですか?

A. いいえ。

注記: フィールドに**プライマリーキー属性**が設定されていなければなりません。

※プライマリーキー属性は，v11以降，存在するものですが，これまで4Dでは特に必要ではなかったため，大多数のデータベースでは使用されていないと思われます。v14で必須になるプライマリーキーは，単にレコードを特定する自然キー/代理キーであるというだけでなく，実際にこの属性が設定されたフィールドでなければなりません。そのようなフィールドには，ストラクチャエディター上で下線が表示されます。

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: どのようにプライマリーキーを作成すれば良いのでしょうか?

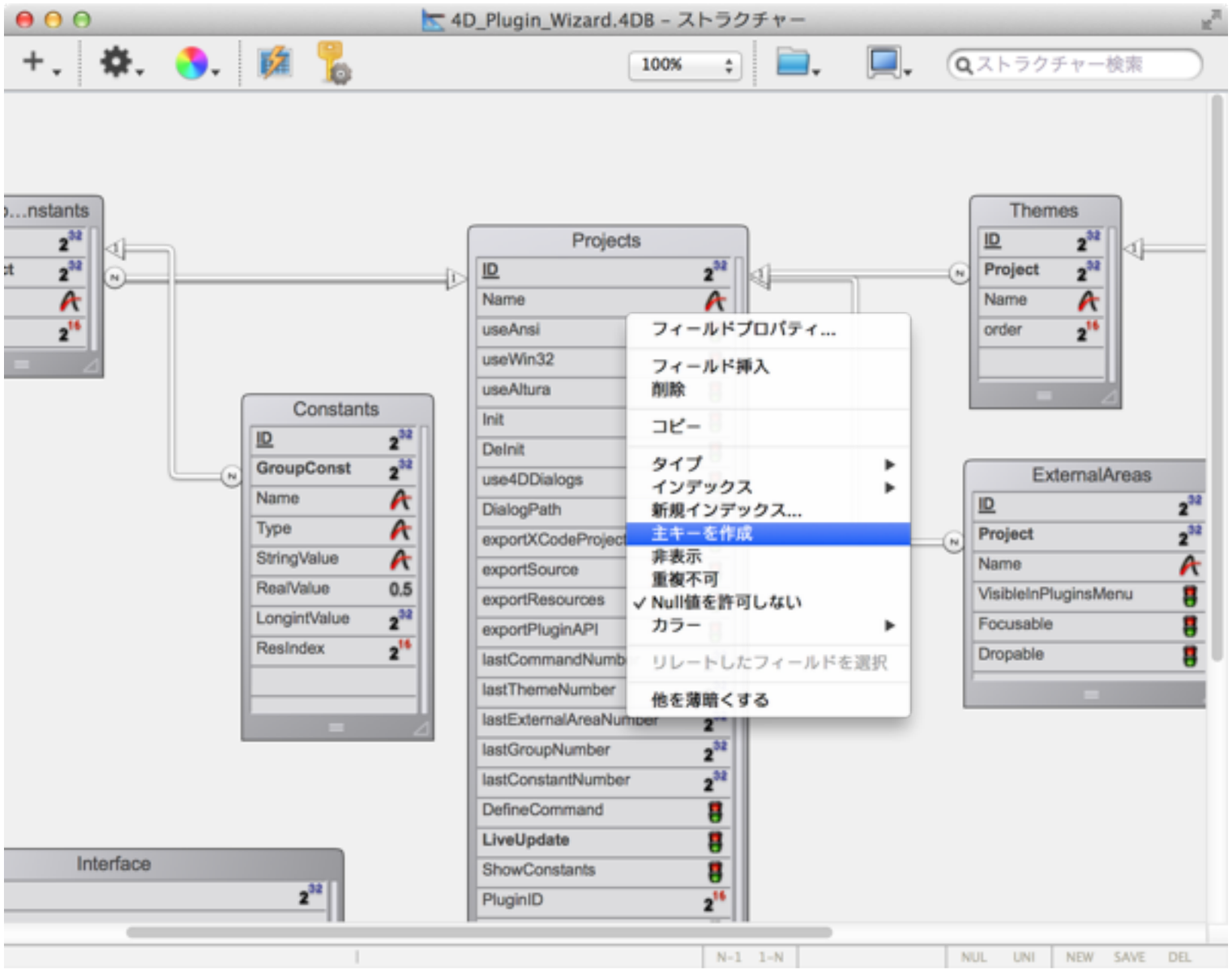
A. データベースの規模に合わせて選ぶことができます。

注記: 小~中規模のデータベースであれば，アシスタントが便利です。

※4Dのプライマリーキーは，倍長整数またはUUID型でなければなりません。自動インクリメント/自動UUIDに設定するかどうかは任意です。いずれにしても，重複不可，NULL不可で，インデックスが必要とされています。新規テーブルに備わっているキー，あるいはプライマリーキーマネージャーで作成したキーは，自動インデックスが設定されています。一方，13.5以降，重複不可属性が設定されたフィールドには，B-TREEインデックスが自動的に作成されます。SQLで作成されたプライマリーキーは後者に該当します。もっとも，結果的に作成されるインデックスは同一です。

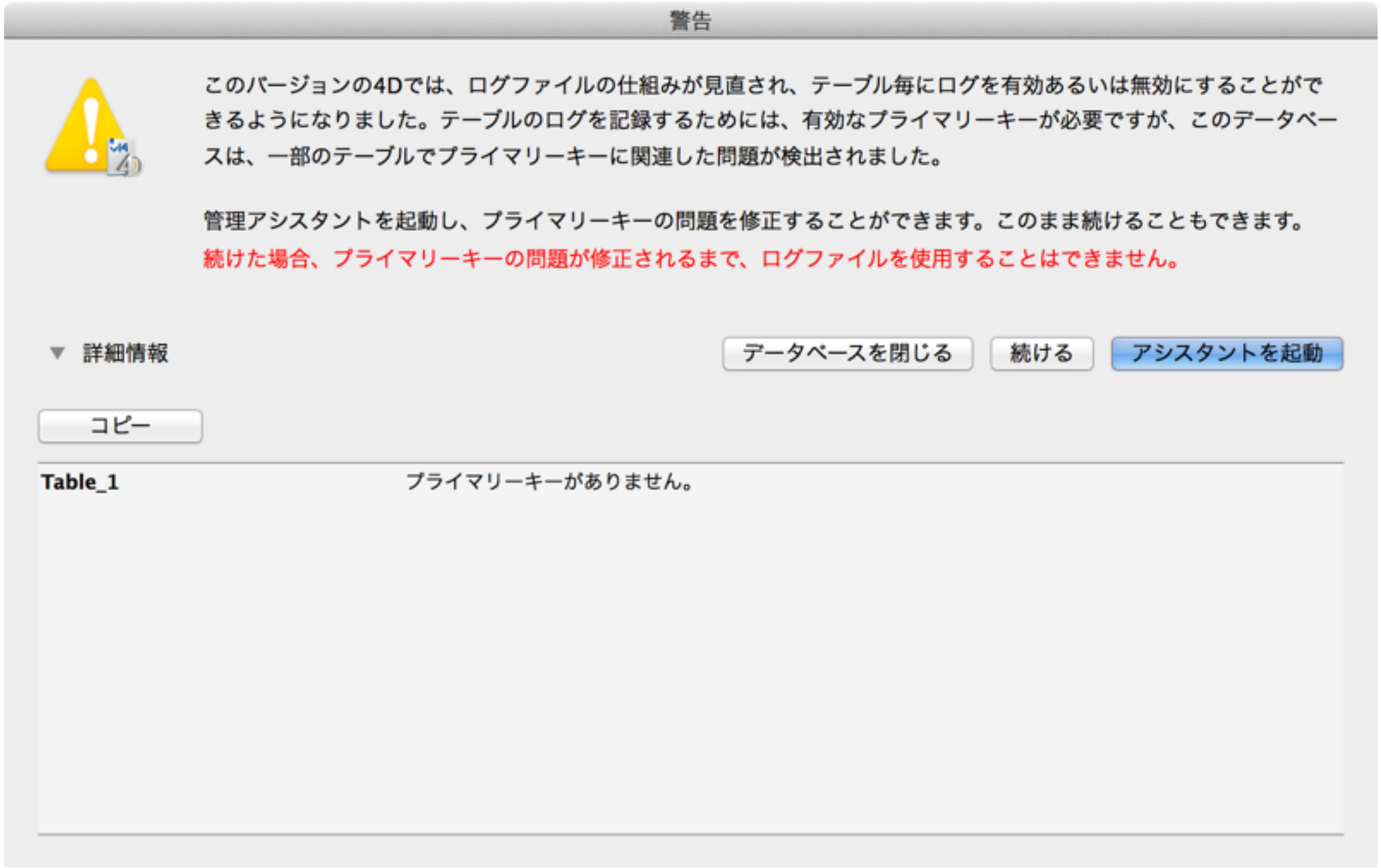
目標: 4D 2003/4からv14にデータベースをアップグレード

① ストラクチャーエディター



目標: 4D 2003/4からv14にデータベースをアップグレード

② プライマリーキーマネージャー



目標: 4D 2003/4からv14にデータベースをアップグレード

③ SQL

```
ALTER TABLE TABLE_1
ADD PRIMARY KEY (MYUUID);
```

```
ALTER TABLE TABLE_1
ADD MYUUID UUID AUTO_GENERATE PRIMARY KEY;
```

※SQLでプライマリーキーを作成するメリットは，コードが再利用できることにあります。たとえば，プライマリーキーとなるフィールド名を何度も入力しなくて済みます。多数のテーブル，あるいは複数のストラクチャを変更しなければならないとしても，SQLであれば，汎用的なコードで対応することができます。

目標: 4D 2003/4からv14にデータベースをアップグレード

```
$numTables_l:=Get last table number

For ($curTable_l;1;$numTables_l)
   $statement_t:=""
   If (Is table number valid($curTable_l))
      $tableName_t:=Table name($curTable_l)
      $statement_t:=$statement_t+"ALTER TABLE ["+$tableName_t+"] "
      $statement_t:=$statement_t+"ADD myUUID UUID AUTO_GENERATE
PRIMARY KEY;"
      Begin SQL
         EXECUTE IMMEDIATE :$statement_t;
      End SQL
   End if
End for
```

http://kb.4d.com/assetid=76991

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: プライマリーキー付きのv14ストラクチャでv13データを開いても大丈夫ですか?

A. はい。

注記: その場合，最初に**プライマリーキーチェック**が実行されることになります。

※プライマリーキーチェックは，いくつかのステップから成っています。はじめに，ジャーナルファイルが有効であるかチェックし，もし有効であれば，すべてのテーブルにプライマリーキーが設定されていることをチェックします。プライマリーキーのないテーブルがあれば，ジャーナルファイルの使用を停止（コンパイルモード），またはアシスタントを起動します。プライマリーキーが設定されている場合，すべての値が有効（重複なし・NULLなし）であることをチェックします。問題がある場合，ジャーナルファイルの使用を停止（コンパイルモード），またはアシスタントを起動します。

目標: 4D 2003/4からv14にデータベースをアップグレード

プライマリーキーチェック

①: プライマリーキーフィールドにNULLがないかチェック。

注記: NULLチェックは，インデックスの有無に関係なく，**シーケンシャル検索**です。

②: プライマリーキーフィールドに重複がないかチェック。

注記: v13.5以降，重複不可フィールドには強制的にB-TREEインデックスが設定されています。

③: プライマリーキーフィールドのNULLを是正。

注記: 自動インクリメントまたは自動UUIDプライマリーキーフィールドに限られます。

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: プライマリーキーチェックには時間がかかりますか?

A. はい。

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: プライマリーキーチェックには時間がかかりますか?

Windows 7
i7-4770 (3.4GHz, 4 Core)
32GB RAM
Samsung 840 Pro 500GB SSD

データファイル: 8GB
テーブル数: 255
フィールド数: 12,444
レコード数: 5,114,973

チェック所要時間: 14:43

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: プライマリーキーチェックには時間がかかりますか?

Windows 7
i7-4770 (3.4GHz, 4 Core)
32GB RAM
Samsung 840 Pro 500GB SSD

データファイル: 8GB
テーブル数: 255
フィールド数: 12,444
レコード数: 5,114,973

チェック所要時間: 14:43

人為的なデータベース
フィールドタイプは文字列のみ
ほぼ断片化なし

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: プライマリーキーチェックには時間がかかりますか?

Windows 7
i7-4770 (3.4GHz, 4 Core)
32GB RAM
Samsung 840 Pro 500GB SSD

データファイル: 8GB
テーブル数: 255
フィールド数: 12,444
レコード数: 5,114,973

チェック所要時間: 14:43

人為的なデータベース
フィールドタイプは文字列のみ
ほぼ断片化なし

データファイル: 30GB
テーブル数: 120
フィールド数: 1,440
レコード数: 13,386,172

チェック所要時間: ?

GET TABLE FRAGMENTATION
断片化: 5%~65%
（3テーブルは100万件以上で30%以下）

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: プライマリーキーチェックには時間がかかりますか?

Windows 7
i7-4770 (3.4GHz, 4 Core)
32GB RAM
Samsung 840 Pro 500GB SSD

データファイル: 8GB
テーブル数: 255
フィールド数: 12,444
レコード数: 5,114,973

チェック所要時間: 14:43

人為的なデータベース
フィールドタイプは文字列のみ
ほぼ断片化なし

データファイル: 30GB
テーブル数: 120
フィールド数: 1,440
レコード数: 13,386,172

チェック所要時間: 1:19:28

GET TABLE FRAGMENTATION
断片化: 5%~65%
（3テーブルは100万件以上で30%以下）

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: どうすればダウンタイムを最少限に抑えられますか?

A. 変換時に既存のフィールドをプライマリーキーに設定することができます。

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: プライマリーキーチェックには時間がかかりますか?

Windows 7
i7-4770 (3.4GHz, 4 Core)
32GB RAM
Samsung 840 Pro 500GB SSD

データファイル: 8GB
テーブル数: 255
フィールド数: 12,444
レコード数: 5,114,973

チェック所要時間: 14:43→2:17

プライマリーキーアシスタントを使用
既存のフィールドをプライマリーキーに

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: プライマリーキーチェックには時間がかかりますか?

Windows 7
i7-4770 (3.4GHz, 4 Core)
32GB RAM
Samsung 840 Pro 500GB SSD

データファイル: 8GB
テーブル数: 255
フィールド数: 12,444
レコード数: 5,114,973

チェック所要時間: 14:43→2:17

データファイル: 30GB
テーブル数: 120
フィールド数: 1,440
レコード数: 13,386,172

チェック所要時間: 1:19:28→14:15

プライマリーキーアシスタントを使用
既存のフィールドをプライマリーキーに

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: 変換時に既存フィールドをプライマリーキーにすることにはリスクが伴いますか?

A. はい。

※プライマリーキーチェックで例外が検出された場合，ジャーナルファイルの使用が停止されます。ですから，データ変換時に既存のフィールドをプライマリーキーに設定するのであれば，そのフィールドに重複あるいはNULL（自動インクリメントあるいは自動UUIDが設定されていない場合）がないことを事前にチェックしなければなりません。また，ジャーナルを使用するためには，再度バックアップが必要になる，したがって余計なダウンタイムが発生することにも留意してください。

目標: 4D 2003/4からv14にデータベースをアップグレード

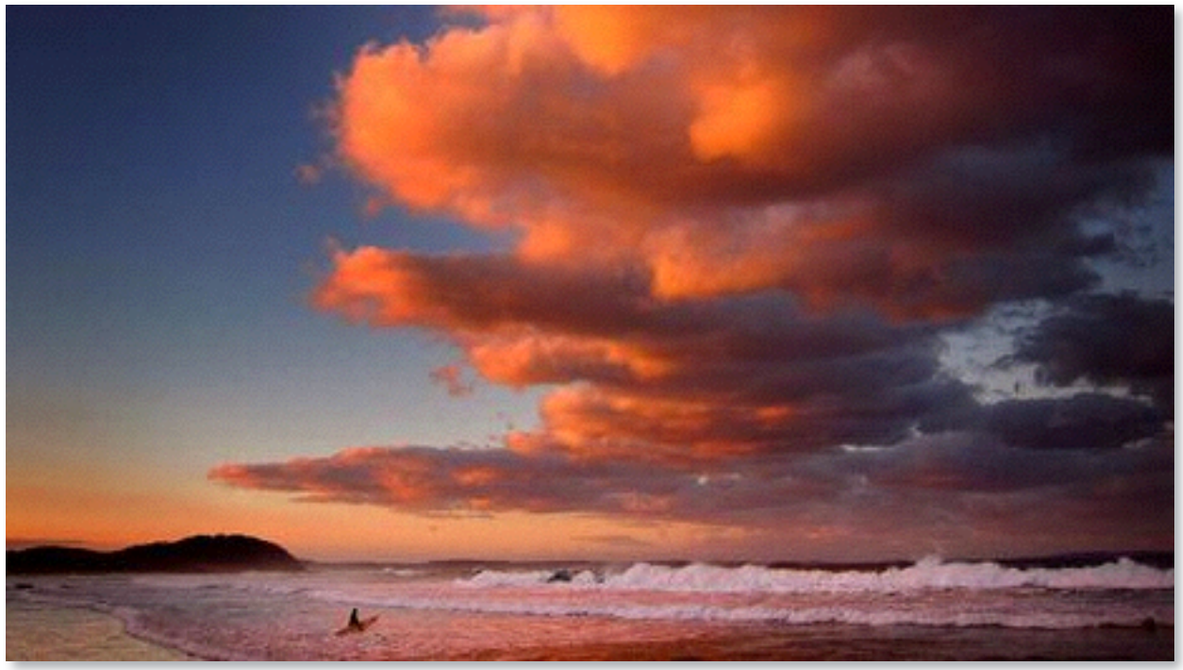
Q: では，どうすれば良いのですか?

A. 変換前，v13でプライマリーキーを作成し，有効な値を保存しておきます。

※既存のビジネスロジックやリレーション設定に干渉しないよう，新たなプライマリーキーフィールドを作成すると良いでしょう（サロゲートキー）。その場合，既存のレコードは，NULLになるので，有効な値を代入して更新します。
v13のデータをv14で開いた場合，すでにプライマリーキーが設定されていたとしても，チェック自体は実行されます。しかし，新しい値を発行しなくても良いのでチェック時間は最少限，また無効な値によりログファイルの設定が不意に外されることを回避することができます。アシスタントはv13コンポーネントとしても公開されています。

ftp://ftp-public.4d.fr/Components/4Dv13/4Dv13_PK_Manager_Component

目標: 4D 2003/4からv14にデータベースをアップグレード



IMAGES

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: PICT形式の画像フォーマットはv14でもサポートされていますか?

A. いいえ。

注記: 画像形式は**v13**でまとめて変換することができます。

※v11以降，PICTを含む画像のフォーマットは，CONVERT PICTUREで変換することができます。しかし，v14以降，PICT形式はサポートされていません。v13.2以降，4D PackのAP Is picture deprecatedコマンドにより，変換の必要なPICTフォーマットのピクチャを事前に検出することができます。変換せずにv14にアップグレードした場合，ピクチャの代わりに「サポートされない画像フォーマット」という文字列が表示されます。**14R3**以降，その拡張子またはタイプ（4文字以内）が表示されます。

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: PICT形式の透過属性はv14に持ち越すことができますか?

A. はい。

注記: **14R2**で透過PNGに変換することができます。

※v11以降，PICTでフォーマットは，特定のカラー（通常はホワイト）を透明の代替色として指定することができました。そのようなピクチャをv13でPNGに変換した場合，透過部分はホワイト等になります。**14R2**のCONVERT PICTUREは，透過ピクセルとして処理するべきRGBカラーを指定することができます。これにより，よりクオリティの高い非PICT画像フォーマットにアイコンなどを変換することができます。コマンドは，似たような仕様のBMPを始め，どの画像フォーマットに対して有効です。

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: QuickTimeのインストールは必要ですか?

A. いいえ。

注記: インストールすれば，QuickTime形式の画像も扱えるようになります。

※v2004までは，Windows版でJPEGなどの画像を扱うためにはQuickTimeのインストールが必要でした。v11以降，JPEGやPNGなどの代表的な画像フォーマットは直接4Dで処理することができます。しかし，以前にQuickTime経由で取り込まれたJPEGピクチャの表示には，引き続きQuickTimeが必要です。そのため，QuickTimeがインストールされているWindowsマシンまたはMacで画像をPNGに変換することが勧められています。そうしておけば，以後，QuickTimeがインストールされていないWindowsでも画像が表示できるようになるからです。

目標: 4D 2003/4からv14にデータベースをアップグレード

Q: ピクチャライブラリはv14でもサポートされていますか?

A. はい。

注記: Resourcesフォルダーに画像を移動すれば，パフォーマンスが向上します。

※Resourcesは，v11以降，画像，テキスト，その他のストラクチャー共有リソースを管理するためのフォルダーです。ここに置かれた画像ファイルは，参照カウントという仕組みにより，メモリの使用が最適化されます。また，GIFアニメは，フォーム上で実際に動きます。余裕があれば，ピクチャライブラリの中身をPNG形式でエクスポートし，Resourcesフォルダーに移動することを検討しても良いでしょう。（GitHubリンク参照）

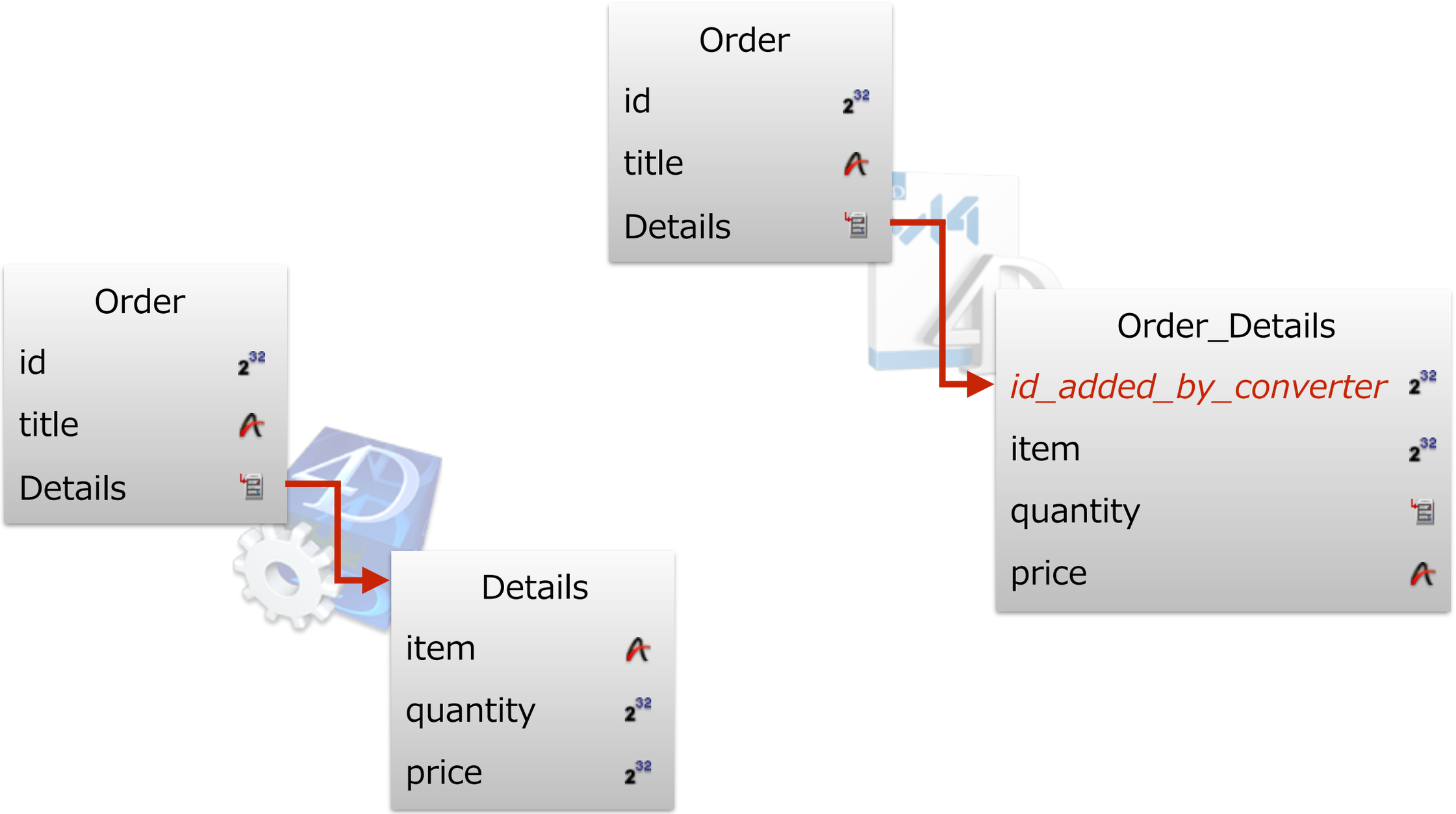
https://github.com/miyako/4d-utility-resources-kit

目標: 4D 2003/4からv14にデータベースをアップグレード

SUBTABLES

目標: 4D 2003/4からv14にデータベースをアップグレード

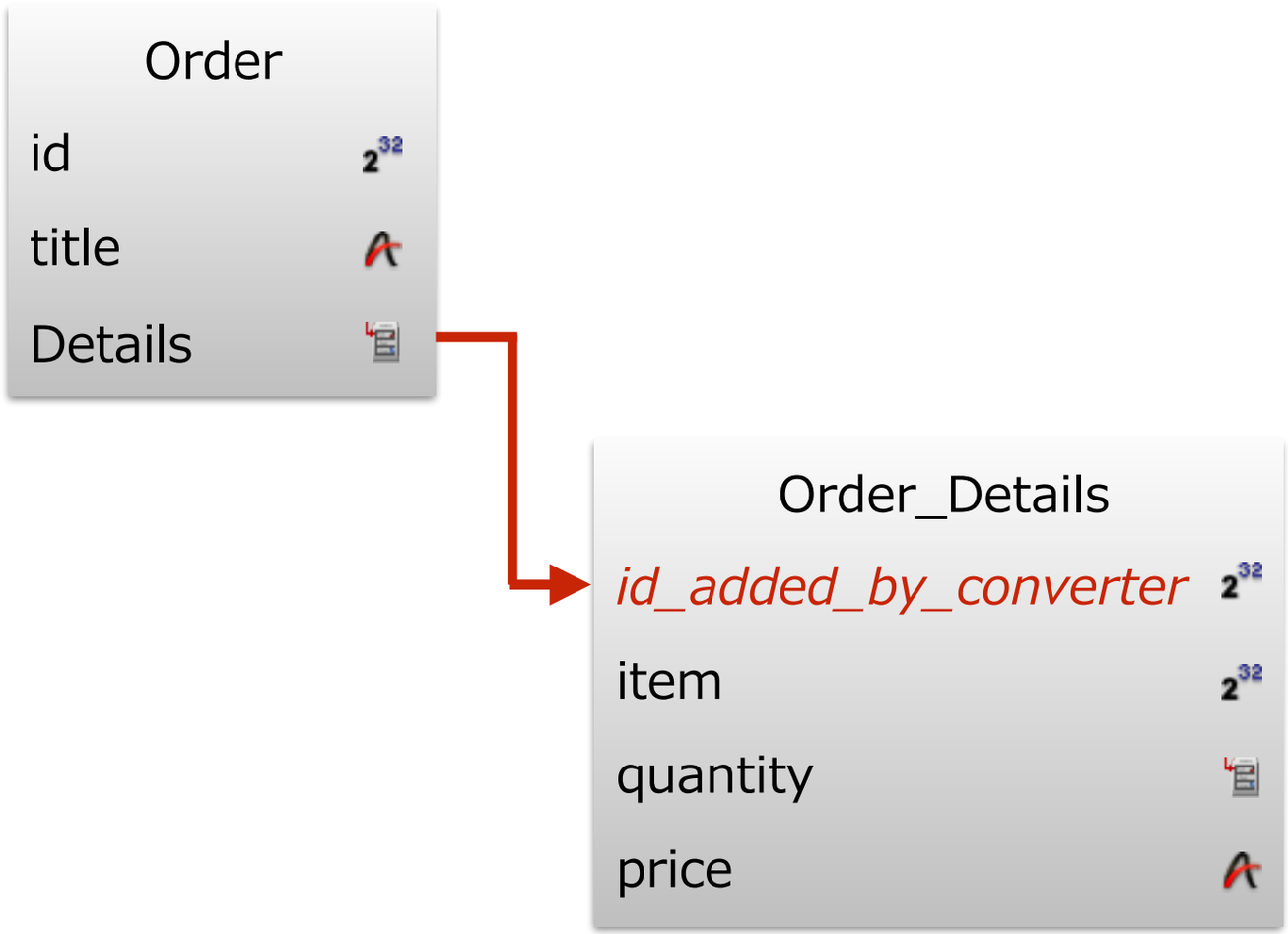
ポイント①: サブテーブルは標準テーブルに変換されます。



目標: 4D 2003/4からv14にデータベースをアップグレード

ポイント②: サブテーブルは変換された標準テーブルに対する疑似APIです。

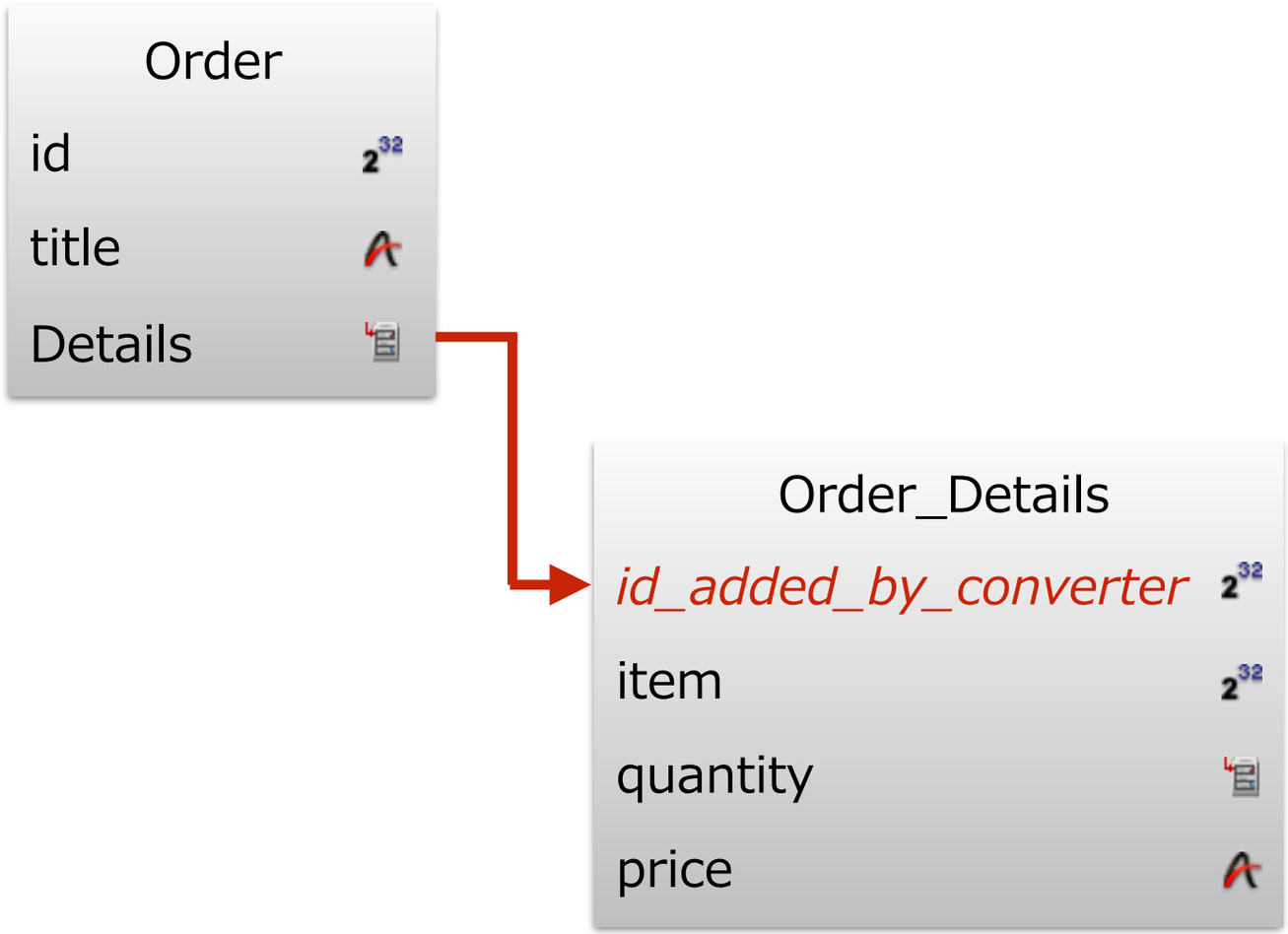
```
[Order]Details //サブテーブル
[Order_Details] //テーブル
```



目標: 4D 2003/4からv14にデータベースをアップグレード

ポイント③: サブレコードコマンドを使用すればサブテーブルのように扱えます。

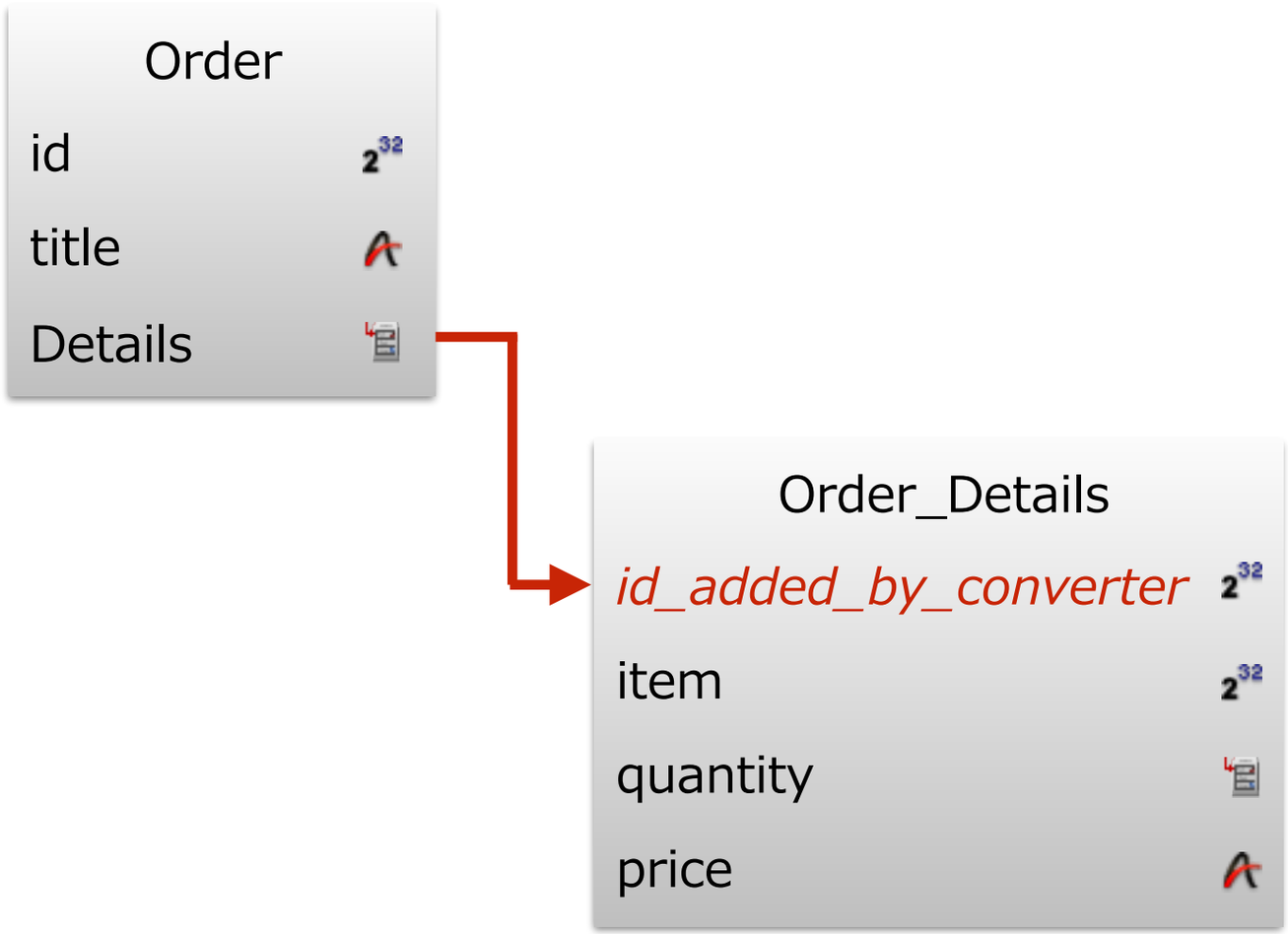
```
ALL SUBRECORDS([Order]Details)
```



目標: 4D 2003/4からv14にデータベースをアップグレード

ポイント④: 標準コマンドを使用すれば標準テーブルのように扱えます。

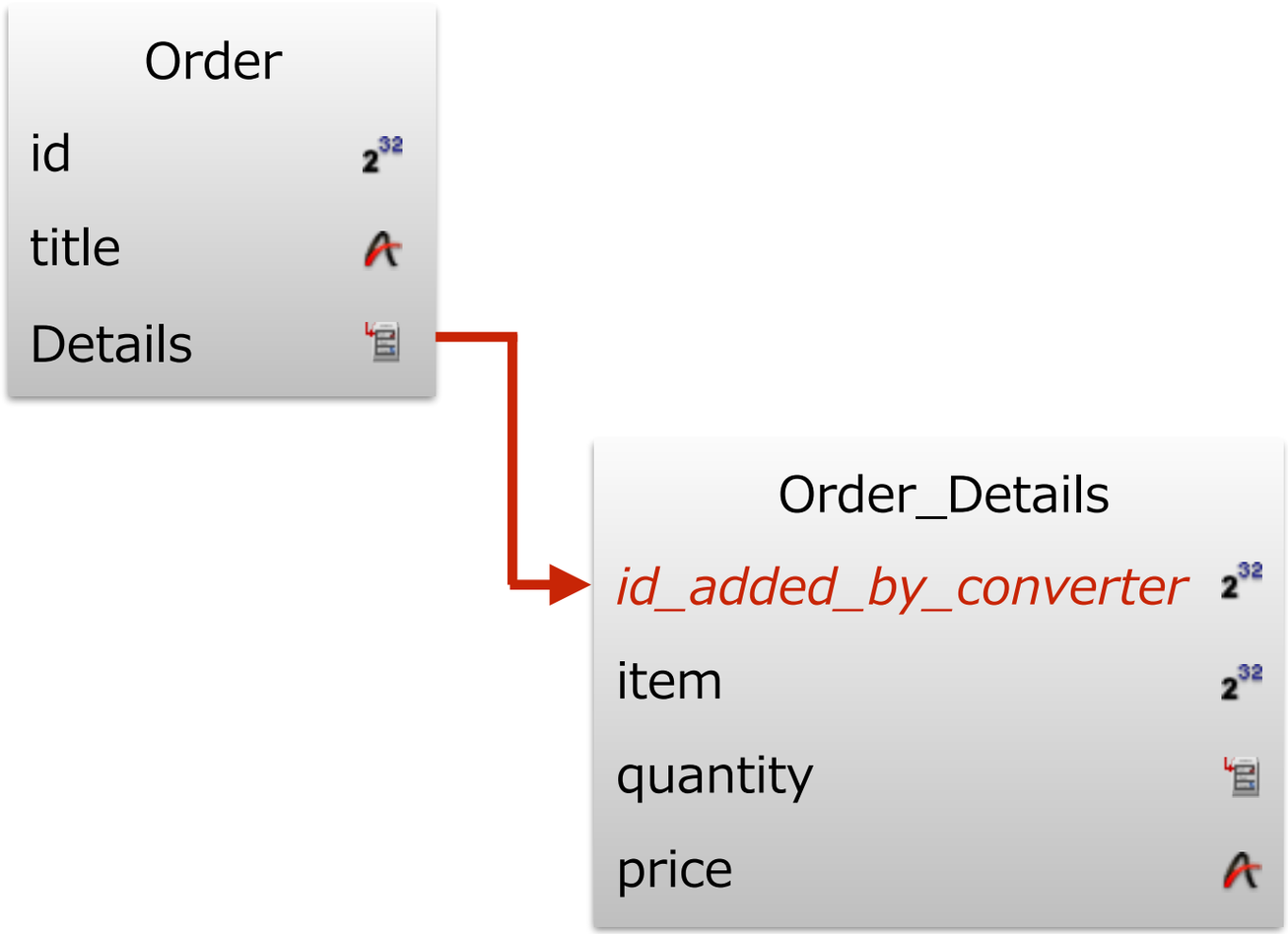
```
QUERY([Order_Details];[Order_Details]id_added_by_converter=Get
subrecord key([Order]Details))
```



目標: 4D 2003/4からv14にデータベースをアップグレード

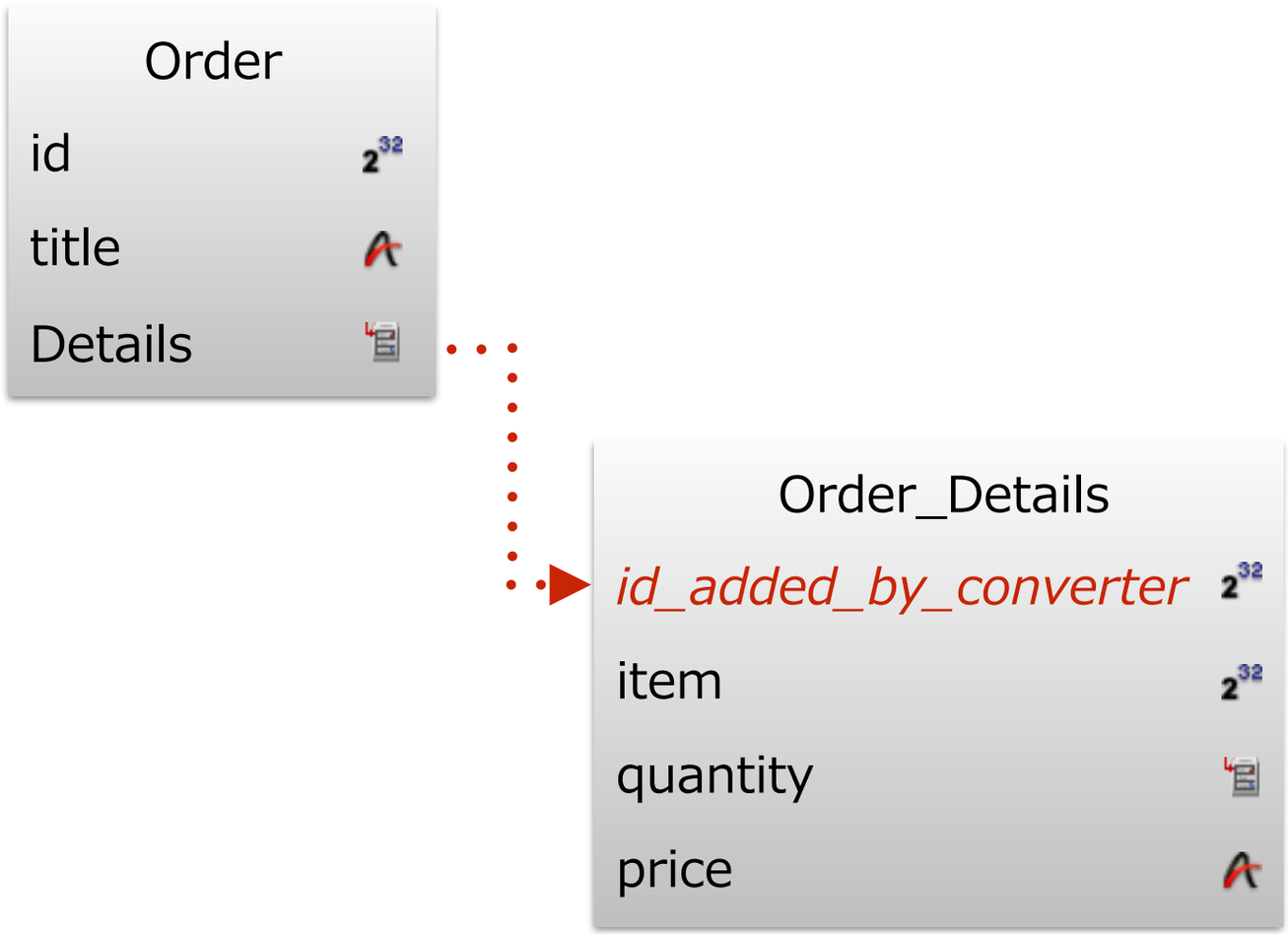
ポイント⑤: サブセレクションとセレクションは連動していません。

```
Records in subselection([Order]Details)
Records in selection([Order_Details])
```



目標: 4D 2003/4からv14にデータベースをアップグレード

ポイント⑥: サブテーブルのリンクを切断した後は元に戻せません。



目標: 4D 2003/4からv14にデータベースをアップグレード

ポイント⑦: サブテーブルから標準テーブルの移行は計画的に進める

1. サブテーブルのリンクを切断するメリットを評価する

   書き出し/読み込み
   SQLアクセス
   ユーザーインタフェース（リストボックス等）

2. サブテーブルコマンドを使用している箇所は一連の処理を書き換える

   サブセレクション→セレクション
   サブレコード→レコード
   サブフォームのデータソース（サブテーブルフィールド→テーブル）

3. サブテーブルコマンド/サブフォームが完全に無くなったところでリンクを切断

目標: 4D 2003/4からv14にデータベースをアップグレード

参考: サブテーブルを（敢えて）作成する方法

```
PA_CreateElementsFromXMLDefinition((PA_Unistring*)(pParams[0]));
```

https://github.com/miyako/4d-plugin-structure-access/

目標: 4D 2003/4からv14にデータベースをアップグレード

TEXT

目標: 4D 2003/4からv14にデータベースをアップグレード

ポイント①: データファイルの文字コードはアップグレード時に変換されます。

Shift_JIS

UTF-16

目標: 4D 2003/4からv14にデータベースをアップグレード

ポイント②: 非Unicodeモードは変換されたデータに対する疑似APIです。

目標: 4D 2003/4からv14にデータベースをアップグレード

ポイント③: UnicodeモードはON/OFFを切り替えることができます。※再起動が必要。

5=Length(“あいうえお”)

目標: 4D 2003/4からv14にデータベースをアップグレード

ポイント④: Shift_JISからUnicodeの移行は計画的に進める

```
USE CHARACTER SET("Windows-31J";0)
USE CHARACTER SET("Windows-31J";1)

SEND PACKET(…)
EXPORT TEXT(…)

USE CHARACTER SET(*;0)
USE CHARACTER SET(*;1)
```

目標: 4D 2003/4からv14にデータベースをアップグレード

ポイント④: Shift_JISからUnicodeの移行は計画的に進める

```
If (Get database parameter(Unicode mode)=0)
    $len:=Length($value)
Else
    CONVERT FROM TEXT($value;”Windows-31J”;$valueData)
    $len:=BLOB Size($valueData)
End if
```

目標: 4D 2003/4からv14にデータベースをアップグレード

EXTRA

参考: 消費税の計算，実数，イプシロン値

Q: 消費税改定後の実数計算で注意することがありますか?

A. はい。

注記: 実数は**近似値**であることを忘れないでください。

※Alturaツールボックスが実数を表示するときには，実数から文字列に変換するSANE (Standard Apple Numeric Environment) 変換ルーチンと同様の丸め処理が適用されていました。68KとPPCのCPUでは，実数の内部データ長が違うためです。この設定は，データベースパラメーターのReal Display Precisionで変更することができました。v14は，システムライブラリで実数を処理しているため，このセレクターは不要となっています。一方，イプシロン値 (SET REAL COMPARISON LEVEL) は引き続き有効であり，デフォルトは10^{-6}です。

参考: 消費税の計算，実数，イプシロン値

ポイント: 差がイプシロン値 (10^{-6}) 以内の実数値は等価とみなされます。

$0.00002 - 0.00001 = 0.00001 = 10^{-5} > 10^{-6}$ //0.00002 # 0.00001

$0.000002 - 0.000001 = 0.000001 = 10^{-6} = 10^{-6}$ //0.000002 = 0.000001

参考: 消費税の計算，実数，イプシロン値

ポイント: イプシロン値はSET REAL COMPARISON LEVELで変更できます。

```
SET REAL COMPARISON LEVEL(10^-7)
0.000002 - 0.000001 = 0.000001 = 10^-6 > 10^-7 //0.000002 # 0.000001
```

参考: 消費税の計算，実数，イプシロン値

ポイント: コンピューター演算における実数値は近似値です。

```
$r1:=-1700*1.08
$r2:=-1700*108/100

SET REAL COMPARISON LEVEL(10^-12)
$comp1:=$r1=$r2//True

SET REAL COMPARISON LEVEL(10^-13)
$comp2:=$r1=$r2//False
```

解説: 1.08と108/100は同じ値ですが，上記の例で分かるように，小数点以下第13位で誤差が発生しています。メソッドエディター等に1.08と入力した場合，その内部的な値は，1.0800000000001かもしれないということです。

参考: 消費税の計算，実数，イプシロン値

ポイント: Intは切り捨て関数ではなく，丸め関数です。

```
$negative:=Int(-1.08)//-2
$positive:=Int(1.08)//1
```

注記: 負の値は**ゼロから遠い数値**に実数を丸めます。

参考: 消費税の計算，実数，イプシロン値

ポイント: 実数，負の値，Intの組み合わせには注意が必要です。

```
$bad:=Int(-1700*1.08)//-1837
```

参考: 消費税の計算，実数，イプシロン値

ポイント: 実数を整数に変換することにより，誤差を切り捨てることができます。

```
$good:=Floor(-1700*1.08)//-1836
```

```
C_REAL($1)
C_LONGINT($0)

$0:=$1
```

参考: 消費税の計算，実数，イプシロン値

ポイント: 整数値で計算した後，実数に変換する方法も有効です。

```
$good:=Int(-1700*108/100)//-1836
```

参考: 消費税の計算，実数，イプシロン値

ポイント: 文字列に変換することでも誤差を切り捨てることができます。

```
$good:=Int(Num(String(-1700*1.08)))
```