RagTime Connect

Einführung in die Datenbank-Anbindung für RagTime

Hilden März 2002, zweite Fassung

Inhalt

Vorbemerkung

Dieser Text ist ursprünglich für ein „Trainer-Training“ entstanden. Die erste Fassung entstand im September 2001 — und damit vor erscheinen von RagTime Connect. Diese Fassung vom März 2002 wurde durchgesehen, weil sich kleine Details in RagTime Connect geändert haben. Zusätzlich wurden ein paar kleine Ergänzungen eingefügt.

Das Trainer-Training fand mit Teilnehmern statt, die RagTime sehr gut kennen aber die kaum SQL-Kenntnisse hatten. Entsprechend dieser Zielgruppe sind die Informationen zu RagTime-typischen Dingen eher knapp gehalten während der Anhang ausführlich auf SQL-Select-Anweisungen eingeht.

Dieser Text war zugleich eine Vorarbeit für die Beispieldateien, die mit RagTime Connect geliefert werden. Die in diesem Zusammenhang entstandene „Geographie“-Datenbank liegt auch diesem Text für Illustrationen zugrunde.

Hilden, März 2002

Einführung RagTime Connect

RagTime Connect ist eine Datenbank-Anbindung für RagTime, die in Dokumenten die Übernahme von Daten aus ODBC-Datenbanken ermöglicht. Dabei kann außer auf Texte und Zahlen auch auf Bildinformationen oder andere Dateitypen, die RagTime importieren kann, zugegriffen werden.

Die RagTime Connect Erweiterung installiert in RagTime einen neuen Komponententyp, RagTime Connect. Zusätzlich werden einige Rechenblattfunktionen installiert, die speziell für den Datenbank-Zugriff nützlich sind. Die Formel-Syntax von RagTime wurde so erweitert, dass viele Techniken ohne zusätzliche Funktionen ermöglicht werden. Seriendrucke beispielsweise werden nicht über eine spezielle RagTime Connect Seriendruckfunktion erzeugt sondern mit der eingebauten Standardfunktion "Serienbrief". RagTime Connect installiert eine zusätzliche Steuer-Palette, mit der zwischen Datensätzen geblättert werden kann.

Was ist eine RagTime Connect Komponente

Eine RagTime Connect Komponente stellt die Verbindung zu einer Datenbank her. Will man auf einer Datenbank mehrere Suchen gleichzeitig ausführen, legt man dafür nicht mehrere Komponenten an sondern mehrere Abfragen in einer. Mehrere Komponenten braucht man, wenn ein Dokument mehrere Datenbanken gleichzeitig benutzt, einen Oracle-Server z.B. und eine Access-Datenbank.

Die Verbindung zur Datenbank wird dabei über ODBC (Open Database Connectivity) hergestellt. RagTime Connect hat deshalb nichts mit den spezifischen Netzwerkeigenschaften einer Datenbank und deren Login-Prozedur zu tun. Diese Schicht erledigt ODBC. RagTime Connect speichert in der Komponente den ODBC Connection String und kann damit die Verbindung erneut aufbauen. Auch gegenüber dem Connection String ist RagTime Connect agnostisch: Jeder ODBC-Treiber legt dort die Informationen ab, die er braucht. (Ausnahme:

Bei Access analysiert RagTime den Connection String und merkt sich außer dem absoluten Pfad der Datenbank auch den relativen vom Dokument aus gesehen. Dies erlaubt namentlich, dass die mitgelieferten Beispieldokumente mit Access funktionieren, unabhängig davon, wo der Beispielordner installiert wurde.)

Die Daten, die eine RagTime Connect Abfrage liefert, können mit zwei Techniken im Dokument benutzt werden: Die Felder können — z.B. durch Ziehen aus dem Inventar — in Containern installiert werden. An allen Stellen, an denen RagTime Formeln erlaubt, kann auch mit Formeln auf die Ergebnisse einer Abfrage zugegriffen werden.

Die RagTime Connect Komponente kann auf die Datenbank ausschließlich über SELECT-Anweisungen zugreifen. Änderungen in der Datenbank sind nicht möglich.

Der Abfrage-Editor von RagTime Connect gibt zwei Teile eines SQL-Statements fest vor: SELECT und FROM. Es ist nicht möglich, dass jemand eine Schreiboperation auf der Datenbank ausführt. Die notwendigen Anweisungen müssten mit INSERT, UPDATE oder DELETE beginnen.

Schreiboperationen von RagTime aus wären nicht grundsätzlich sinnlos, aber problematisch im Rahmen der RagTime Connect Konzeption.

Gründe für die Beschränkung auf Lesezugriffe: Viele Datenbankoperationen müssen als Gruppe entweder ganz oder gar nicht ausgeführt werden. Eine Kontobuchung z.B. muss unbedingt einem Konto gut, dem anderen last geschrieben werden. Würde dieser Vorgang nach einer Teiloperation unterbrochen werden, wären anschließend die Daten inkonsistent. Datenbanken kennen dafür ein Transaktionskonzept. Eine Transaktion wird mit einer Anweisung begonnen, dann folgen alle möglichen Teiloperationen und schließlich löst ein COMMIT aus, das die Aktion als ganzes ausgeführt wird. Geht bei einer der Teiloperationen etwas schief, wird die Transaktion als ganzes nicht ausgeführt (respektive die augeführten Teile rückgängig gemacht). Für solche Transaktionen ist ein anderes Konzept notwendig.

SQL kennt erheblich detailliertere Datentypen als ein Rechenblatt. Zahlen kommen in diversen Genauigkeiten vor. Bei der Abfrage bilden wir sie auf die passenden Rechenblatt-Datentypen ab. Für Schreibaktionen gibt es ein Abbildungsproblem, wie z.B. Zahlen mit der RagTime-Rechenblattgenauigkeit auf Zahlen mit höherer Genauigkeit in der Datenbank abgebildet werden.

Aus diesen Gründen ist mit der jetzigen Version von RagTime Connect ausschließlich ein Abfragen der Datenbank möglich, aber kein Schreibzugriff.

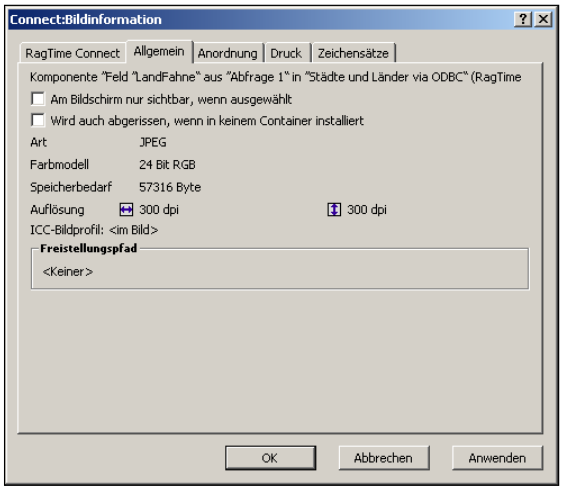
RagTime Connect-Felder installieren vs. Formelzugriff

Eine RagTime Connect-Komponente erzeugt im Inventar einen gegliederten Eintrag. Unterhalb einer Komponente gibt es eine oder mehrere Abfragen. Nach dem Ausführen der Abfragen enthält jede auf der nächsten Ebene die Namen der von der Datenbank gelieferten Feldnamen.

Auf die Felder kann mit zwei Techniken zugegriffen werden: Die Felder lassen sich in Containern installieren und auf die Felder kann mit Formeln zugegriffen werden. Der Formelzugriff erlaubt meist die flexiblere Lösung.

Ziehen eines Feldes aus dem Inventar auf einen leeren Container installiert dieses Feld. Der Inhalt des Containers ist dann eine Connect-Componente. Hat die Datenbank mehrere Datensätze zurückgeliefert, zeigt der Container den „aktuellen“ Datensatz. Mit der Palette „RagTime-Connect-Steuerung“ kann man blättern, d.h. andere Datensätze zum aktuellen machen.

Wenn ein Datenbankfeld Informationen liefert, die RagTime als Komponente inmportieren kann (Bilder z.B.), dann hat die installierte Komponente einen Zwittercharakter: Es handelt sich um eine RagTime Connect-Komponente, aber zugleich um beispielsweise ein Bild. Der Informationendialog für ein solches Objekt reflektiert dies:



Doppelpunkt-getrennt zeigt der Dialogtitel beide Inhaltsarten. Dasselbe gilt für den Namen des Komponenten-Menüs, wenn das Objekt ausgewählt ist. Menü und Informationsdialog enthalten die Anweisungen / Informationen sowohl zur RagTime Connect-Komponente als auch zum importierten Inhaltstyp.

Formelbezüge auf Felder können über die Formelpalette erzeugt werden (klicken Sie auf einen Feldnamen im Inventar während die Einfügemarke in der Formelpalette blinkt, um einen Bezug anzulegen) oder auch durch Drag & Drop. Ziehen eines Feldes in einen Text oder in eine Rechenblattzelle legt einen Formelbezug an.

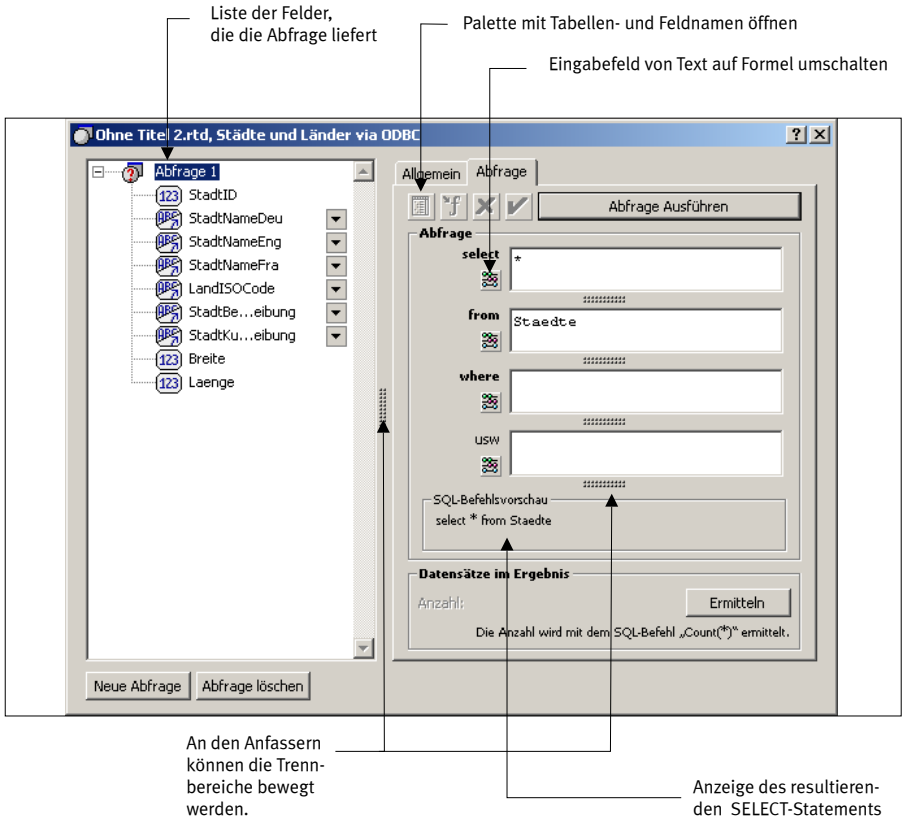
Durch Mausklick oder ziehen angelegte Formelbezüge beziehen sich immer auf das Feld im aktuellen Datensatz. Andere Bezüge und Bereichsreferenzen müssen durch manuelle Änderung in der Formelpalette erzeugt werden. Details zu den Referenzformen finden Sie weiter unten.S

Der Abfrage-Editor

Der Abfrage-Editor besteht aus zwei Tafeln: Abfrage und Allgemein. Im Bereich Abfrage befinden sich die Eingabefelder für das SQL-Statement.

Die Tafel „Abfrage“

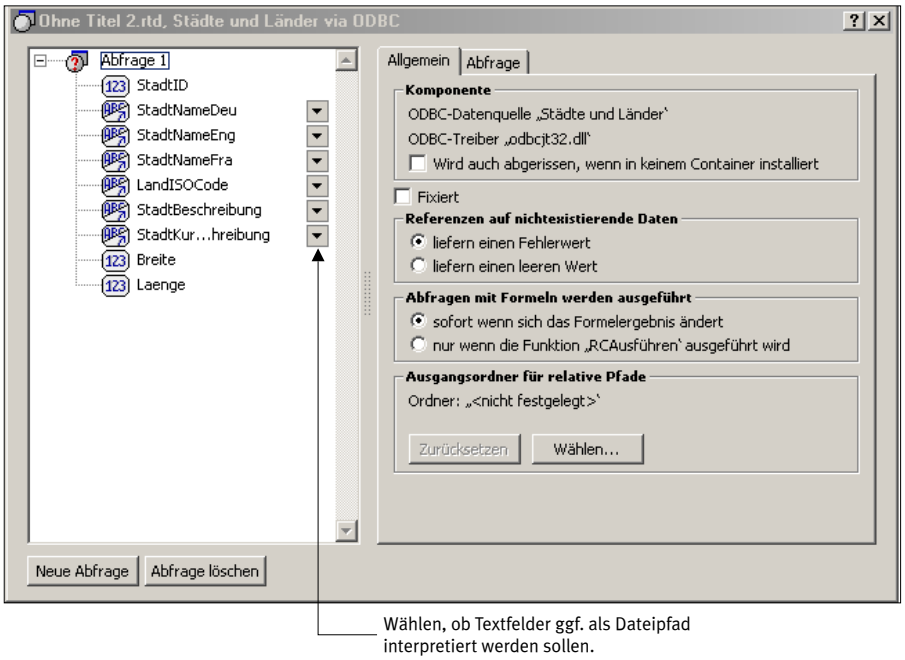
Die vorgegebenen Schlüsselwörter SELECT, FROM und WHERE geben einem Anfänger ein Gerüst an die Hand. Vorführungen auf der CeBIT 2001 haben gezeigt, dass Anwender ohne viel SQL-Hintergrund dies als sehr angenehme Hilfestellung empfinden (zusammen mit der Liste der Tabellen- und Feldnamen) und SQL nicht mehr ganz so erschreckend aussieht.



In vielen Lösungen wird die SQL-Abfrage das Resultat einer Berechnung sein. Der Anwender des Formulars gibt Suchbegriffe in Rechenblattzellen ein und daraus wird die fertige SQL-Abfage ermittelt.

Die Teilung in vier Felder, die einzeln auf statischen Text oder Formel geschaltet werden können, hat dabei Vorteile: Es führt zu sehr unübersichtlichen Ausdrücken, wenn man versucht, ein komplettes SELECT-Statement per Formel zu berechnen. Man muss in der Syntax der RagTime-Formeln die Syntax einer anderen formalen Sprache, SQL, erzeugen. Besonders bei den Anführungszeichen wird das schnell sehr unübersichtlich. In den meisten Fällen ist nur die WHERE-Klausel variabel. Alle anderen Teile des Statements betreffen die Struktur der angeforderten Tabelle und sind deshalb bei den meisten Anwendungen konstant. Die Teilung in vier Felder erlaubt es, die konstanten Teile auch als solche zu tippen und nur die wirklich variablen Teile als Formel zu konstruieren. Im vierten Feld können beliebige Ergänzungen wie ORDER BY untergebracht werden, die oft wieder konstanter Text sind.

Die Tafel „Allgemein“



Fixiert

Die Erfahrungen mit FileTime (Anbindung von RagTime an die Mac OS-Version von FileMaker) haben gezeigt, dass die automatische Neuberechnung von Datenbankverbindungen ein großes Problem ist. Anwender haben im einfachsten Fall einen Brief geschrieben und auf Festplatte gesichert. Einen Monat später wird er wieder geöffnet, nimmt neu die Verbindung zur Datenbank auf und ändert seine Adresse. Für Publishing-Zwecke musste noch zusätzlich sichergestellt werden, dass auch ohne DB belichtet werden kann.

Folgende Lösung haben wir implementiert: Das Dokument cashed alle Daten, die es aktuell braucht. Damit kann es auch ohne Datenbank vollständig belichtet werden. Wird das Dokument geschlossen und wieder geöffnet, springt jede RagTime-Connect-Abfrage in dem Moment auf fixiert. Erst auf Antrag (Mausklick in der Palette oder hier im Abfrage-Dialog) wird wieder die Verbindung zur Datenbankneu aufgebaut und die Werte werden aktualisiert.

Beim Abriss von Formularblöcken wird eine RagTime Connect Komponente nicht fixiert. Man muss aber darauf achten, dass die Abfragen nicht bereits fixiert sind, wenn man die Formularblöcke erzeugt.

Abfragen mit Formeln werden sofort ausgeführt

Will man, dass eine Abfrage erst auf Knopf-Klick hin erfolgt, wird hier die sofortige Neuberechnung abgeschaltet. Ein Knopf mit der Funktion "RCAusführen" kann dann dem Anwender eines Dokuments das Interface schaffen, um die Abfrage per Mausklick im Dokument zu starten. RCAusführen könnte die Abfrage auch dann automatisch anstoßen, wenn alle nötigen Felder ausgefüllt sind.

Ausgangsordner für relative Pfade

RagTime Connect kann auch BLOB-Daten (Binary Large OBjects) aus einer Datenbank auslesen. Als BLOB werden dabei alle Formate akzeptiert, die RagTime auch importieren kann. Ob ein JPEG direkt als JPEG-Datenstrom in der DB liegt oder aber — wie bei Microsoft Access — als OLE-Objekt verpackt ist, ist dabei gleichgültig. In einer Reihe von Fällen aber liegen Bilddaten gar nicht in der Datenbank sondern dort wird nur die Position auf einem Server abgelegt. RagTime Connect kann deshalb auch einen Dateipfad interpretieren und die Datei automatisch laden. Voreingestellt für Textfelder ist, dass RagTime Connect versucht, sie als Pfad zu interpretieren. Gelingt dies, wird die Datei importiert, anderenfalls der Text als Ergebnis geliefert. Die Automatik kann in der Feldliste im Abfrage-Editor abgeschaltet werden.

Pfade können im Systemstandard vorliegen oder als File-URL. ftp-URLs werden nicht interpretiert. In vielen Fällen werden solche Pfade als "relative Pfade" von einem bestimmten Ausgangsordner aus sein. Diesen Ausgangsordner kann man im Allgemein-Teil des RagTime Connect Dialogs festlegen.

Die Übernahme von BLOBs und der automatische Import auf Basis eines Dateipfads sind nicht auf Bilder beschränkt. Auch Texte, Excel-Tabellen und was RagTime sonst noch importieren kann, kann so in ein Dokument geladen werden.

Referenzen auf nicht existierende Daten

Bezieht sich eine Formel auf einen Bereich der Datenbank-Antwort, der nicht existiert, ist es konsequent, genau wie im Rechenblattfall einen Fehlerwert zu liefern. Da aber die Anzahl der gelieferten Datensätze bei Anfragen sich regelmäßig ändert, sind diese Fehlermeldungen in der Praxis oft sehr nervig und wären nur mit aufwendigen Wenn-Formeln zu unterdrücken. Deshalb bietet der Abfrage-Editor die Alternative, in solchen Fällen leer zurückzuliefern. Derselbe Unterschied gilt für NULL-Werte aus der Datenbank.

Meistens ist die Technik sinnvoll, in der Design-Phase die Anzeige aller Fehler eingeschaltet zu lassen und sie für das fertige Formular abzuschalten.

Die Palette zum Blättern und der aktuelle Datensatz

Von den Datensätzen, die RagTime auf eine Anfrage hin erhalten hat, ist immer einer für RagTime der „aktuelle“. Diese Auszeichnung eines bestimmten Datensatzes der gelieferten Tabelle ist notwendig, weil wir Felder in Komponenten platzieren können. Dort kann aber immer nur ein Feldinhalt erscheinen und nicht eine ganze Spalte.

Im Beratungs- und Trainings-Kontext mit SQL-Kennern ist wichtig, dass dieser aktuelle Datensatz nichts zu tun hat mit dem SQL-Cursor. Ein SQL-Server liefert die gefundene Tabelle nicht als ganzes an das fragende Programm (es könnten auch sehr viele Datensätze sein), sondern stellt die Tabelle zum schrittweisen Abholen bereit. Das Programm kann einen sogenannten Cursor durch die Tabelle bewegen und erhält dabei jedesmal einen einzelnen Datensatz. Je nach Implementation kann der Cursor nur vorwärts bewegt werden (dies ist ODBC-Standard) oder auch in beide Richtungen. Dies sind jedoch Details, die RagTime Connect dem Anwender gegenüber vollständig versteckt. Je nach Verwendung der Formeln im Dokument versucht RagTime

Connect abzuschätzen, wieviele Datensätze gebraucht werden und cashed die entsprechende Anzahl. Alle Cursor-Aktionen werden also je nach Situation implizit ausgeführt. Der aktuelle Datensatz dagegen ist ein rein RagTime Connect-internes Konzept.

Die Referenznotation in Formeln

Da SQL-Server auf eine Anfrage immer mit Tabellen antworten, war es naheliegend, auf die Antworten auch mit einer Referenznotation zuzugreifen, die der im Rechenblatt ähnlich ist. Es gibt aber ein paar Unterschiede: Die Spalten einer SQL-Tabelle haben Titel, die man in Formeln nutzen können will. Buchstabennotationen für Spalten sind sehr unglücklich, da sie mit Namen kollidieren können. Das Konzept des aktuellen Datensatzes legt es nahe, Zeilen sowohl in Bezug auf die ganze Tabelle als auch auf diesen Datensatz referenzieren zu können. Folgende Notation gilt in RagTime Connect;

Die Bezüge haben die Form: Komponentenname!Feldbezug.Datensatzbezug

Feldbezug ist dabei:	`(<Feldname>|[$]<Spaltennnummer>)`
Datensatzbezug ist dabei:	`([$](<Zeilennummer>|0|(+|-)<Zeilennummer>)|n)`

Bei den Bezügen wirken $-Zeichen wie bei Spaltenbezügen im Rechenblatt. Bezüge über Namen sind immer absolut. Bei Datensatzbezügen gibt es keine Namen. Die Nummern o und Zahlen mit einem Vorzeichen beziehen sich auf den aktuellen Datensatz. Nummern ohne Vorzeichen beziehen sich auf die Nummer des Datensatzes in der Antwort der Datenbank. n ist der letzte Datensatz.

Beispiele

Datenbank via ODBC!Feldname.o	Referenziert das Feld „Feldname“ und dort den aktuellen Datensatz, reagiert also auf blättern. Beim Auffüllen nach oben / unten verhält sich die o relativ. Beim Auffüllen in die nächste Zeile wird der Ausdruck also zu „Datenbank via ODBC!Feldname.+1“
Datenbank via ODBC!1.$o	Referenziert das erste Feld des aktuellen Datensatzes. Die 1 verhält sich beim Auffüllen nach links / rechts relativ. Die $o ist absolut beim Auffüllen nach oben / unten.
Datenbank via ODBC!$1.$1	Referenziert das erste Feld im ersten gefundenen Datensatz. Der Datensatzbezug berücksichtigt nicht den aktuellen Datensatz, die Formel reagiert also nicht auf blättern. Beim Auffüllen nach unten und rechts verhalten sich Feld- und Datensatzbezug absolut.
Datenbank via ODBC!$2.-3	Das zweite Feld des Datensatzes 3 Nummern vor dem aktuellen. Der Feldbezug ist absolut, der Zeilenbezug relativ.
Datenbank via ODBC!2.n	Das zweite Feld des letzten gefundenen Datensatzes. Der Feldbezug ist relativ.

Gibt es mehrere Abfragen in einer RagTime Connect Komponente, enthält der Bezug jeweils noch den Abfrage-Namen:

Datenbank via ODBC!Abfrage 1!$1.$1

Bereichsreferenzen

Funktionen, die im Rechenblatt mit Bereichen arbeiten, können sich auch auf RagTime Connect-Bereiche beziehen. Dafür gibt es eine angepasste Notation für Bereichsreferenzen.

Beispiele für Anwendungen: Seriendrucke über ein Abfrageresultat sind einfach mit der normalen Funktion „Serienbrief“ möglich. Infografiken können sich auf eine Spalte einer Datenbankabfrage beziehen. Die Anzahl der Datenpunkte wächst und schrumpft dann mit dem Abfrageresultat.

Hinweis zu Rechenblatt-Funktionen und Bereichsreferenzen: Bei Rechenfunktionen stehen einige in Konkurrenz zu SQL-Aggregaten. Die „Summe“-Funktion von RagTime auf eine Abfragespalte anzuwenden, scheint selten sinnvoll. Die SQL-Funktion „SUM“ wird schneller arbeiten, da der Server keine Liste senden muss. Funktionen wie „Median“ etc. sind aber in SQL-Datenbanken selten. Werden in einer Auswertung also subtilere Funktionen gebraucht, sollte der Weg über eine RagTime-Funktion sehr nützlich sein.

Bereiche können wie in RagTime üblich mit einer Doppelpunkt-Notation angesprochen werden. Ganze Zeilen und Spalten werden mit * notiert. n bezieht sich je auf die letzte existierende Zeile oder Spalte.

Die Bezüge haben die Form: Abfragename!Bereichsangabe

Bereichsangabe: `(<Zellbezug>:<Zellbezug>|<Spaltenbezug>.*|*.<Zeilenbezug>|*.*)`

Beispiele (die Benutzung von $-Zeichen und Spaltennamen ist hier ebenso möglich, wie bei den bisher angeführten Bezügen und wird hier nicht weiter berücksichtigt):

Datenbank via ODBC!1.*	Die erste Spalte in Datenbank via ODBC.
Datenbank via ODBC!1.0:1.n	Der Bereich vom aktuellen Datensatz bis zum letzten in Spalte 1
Datenbank via ODBC!*.*	Die ganze Abfrage
Datenbank via ODBC!5.*	Der fünfte Datensatz der Abfrage, alle Felder.

Referenzen auf Abfragebereiche können in Funktionen benutzt werden, wie Bereichsreferenzen auf Rechenblätter. "Verbinden(', ';Datenbank via ODBC!1.*) erzeugt einen Komma-getrennten Text mit allen Werten aus der ersten Spalte der Abfrage.

Abfragen mit Formeln konstruieren

Die meisten Formulare, die mit RagTime Connect entworfen werden, werden Eingabebereiche für den Benutzer haben (Rechenblattzellen) und aufgrund der Eingabewerte in der Datenbank suchen. Für einen ersten Zugriff folgendes Beispiel: In einem Rechenblatt "Rechenblatt 1" wird in Zelle A1 der Anfang eines Städtenamens eingegeben. Auf der Seite sollen dann die Bilder, die für diese Stadt vorhanden sind, dargestellt werden.

Für die Eingabefelder "select" und "from" ist der Formelknopf ausgeschaltet. In den Eingabefeldern stehen die festen Bestandteile der SQL-Abfrage. Für den "where"-Bereich wurde die Formeleingabe eingeschaltet. Das Eingabefeld enthält deshalb eine RagTime-Formel, die einen Text abliefert. (Wurde in der Zelle Rechenblatt 1!A1 z.B. "Par" eingegeben, liefert diese Formel: "StadtNameDeu LIKE 'Par%'".)

Aus einer Mischung von festen Texten und der Formel entstehen also je nach Benutzereingabe Abfragen wie:
`SELECT BildTitel, BildInhalt FROM Bilder INNER JOIN Staedte ON Bilder.StadtID = Staedte.StadtID WHERE StadtNameDeu LIKE 'Par%'`

Anführungszeichen

Ein bischen der Übung bedürfen die Anführungszeichen. SQL kennt sowohl doppelte Anführungszeichen wie auch einfache. Die Bedeutung ist ähnlich wie in RagTime-Formeln (ausgenommen die CH-Version): Einfache Anführungszeichen begrenzen eine Zeichenkette, doppelte Anführungszeichen werden für "Quoting" benutzt: Enthält ein Bezeichner (eines Feldes z.B.) Zeichen, die in SQL-Bezeichnern nicht erlaubt sind, kann man die doppelten Anführungszeichen benutzen.

Heißt ein Feld der Datenbank `PLZ Ort`, führt wegen des Leerzeichens `SELECT PLZ ORT FROM...` zu einer Fehlermeldung. `SELECT "PLZ ORT" FROM...` ist dagegen zulässig. Dies entspricht recht genau dem RagTime-Verfahren. Gibt es z.B. ein Rechenblatt namens `Tabelle - Anfang`, so lautet ein Bezug auf diese Tabelle z.B. `"Tabelle - Anfang"!$A$1`. Mit dem Quoting vermeidet RagTime die Doppeldeutigkeit des Bindestrichs.

Langer Rede Sinn: Wer SQL-Anweisungen mit RagTime-Formeln aufbaut, hat leider damit zu kämpfen, dass Anführungszeichen sowohl in der Syntax von SQL als auch der von RagTime eine Sonderbedeutung haben. Soll eine RagTime-Formel einen Text mit einem einfachen Anführungszeichen erzeugen, muss dieses doppelt getippt werden!

`'StadtNameDeu LIKE ''&Rechenblatt 1!$A$1&'%''`

Unterstrichen sind jeweils die doppelten Anführungszeichen innerhalb der RagTime-Zeichenkette, die in SQL ein Anführungszeichen ergeben. (Wer das Chaos steigern möchte, benennt Rechenblatt 1 um in z.B. `Rechenblatt-1` und benutzt in der Datenbank Feldnamen wie `Stadt Name`. Wegen des Bindestrichs quoted dann RagTime, wegen des Leerzeichens im Feldnamen muss für SQL ein quoting erfolgen und die Formel wird zu:

```
'"Stadt Name" LIKE '''&"Rechenblatt-1"!$A$1&'%'''
```

Wer sich nicht gerade im Umgang mit Ebenen und Metaebenen trainieren will, sollte solche Benennungen vermeiden.)

Bei komplizierteren Abfragen ist es oft sinnvoll, sich den SQL-Ausdruck in dem Rechenblatt vorzubereiten. Das klingt zunächst wie ein Umweg, ist aber dennoch übersichtlicher. Die Zelle A2 des Rechenblatts enthält dann den Anfang: `StadtNameDeu LIKE '`, in Zelle A3 kommt der Schlussteil: `%'`.

In A4 wird alles mit einer Formel zusammengerechnet: `A2&A1&A3`. Man erkennt dann sofort das Resulat und sieht Fehler leichter. Bei den mit RagTime Connect gelieferten Beispielen diskutiert der zweite Abschnitt des Beispiels „Seriendruck“ diese Technik ausführlich. Hier können Informationen zu Städten in einem Seriendruck ausgegeben werden. Dabei werden die Städte gesucht, die innerhalb eines Bereichs von Längen- und Breitengraden liegen. Je nach mehr oder weniger vollständiger Benutzereingabe werden unterschiedliche WHERE-Klauseln ermittelt. Dies direkt innerhalb der RagTime Connect-Abfrage lösen zu wollen führt nur noch zu unvertändlichem Formelgewirr.

Formate von Zahlen, Daten, Zeiten

Die Beispieldatenbank enthält bei den Städten die Angabe zum Längen- und Breitengrad in Dezimalschreibweise: Länge 10,5 bedeutet 10° 30" Ost. Will man die Städte in dem Bereich 0,3 bis 15,6 haben, lautet die SQL-Abfage: SELECT StadtNameDeu FROM Staedte WHERE Laenge BETWEEN 0.3 AND 15.6.

Wichtig sind dabei die Punkte als Dezimaltrenner. Die Form, in der bei europäischen Zahlenformaten eine RagTime-Rechenblattzelle diese Zahlen darstellt, 15,6, wird von SQL nicht akzepiert. Deshalb gibt es in RagTime Connect spezielle Umwandlungsfunktionen, die einen Wert in eine SQL-verträgliche Zeichenfolge umwandeln.

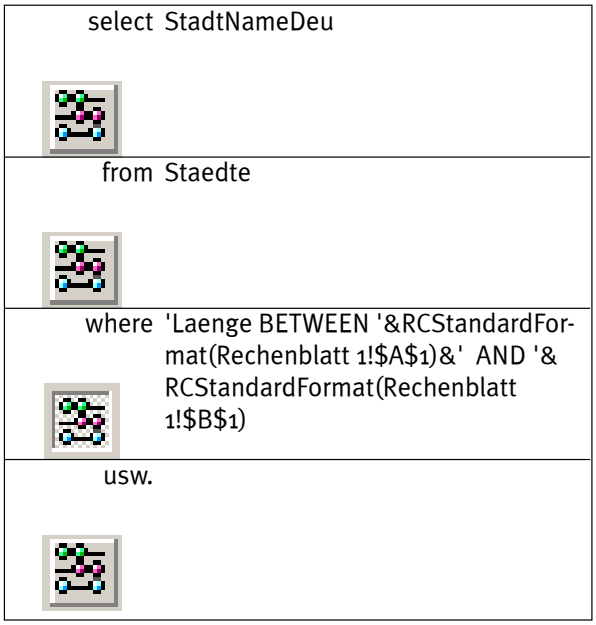
Funktion	Feldwert	konvertierter Text
RCStandardFormat	15,6	15.6
	21.12.2002	{d '2002-12-21'}
	15:20	{INTERVAL '0 15:20:00' DAY(1) TO SECOND(2)}
	21.12.2002 15:20:00	{ts '2002-12-21 15:20:00'}
	01.01.1904 15:20:00	{t '15:20:00'}
	Ein Text	'Ein Text'
RCStandarddatum	15,6	15.6
	21.12.2002	{ts '2002-12-21 00:00:00'}
	15:20	{INTERVAL '0 15:20:00' DAY(1) TO SECOND(2)}
	21.12.2002 15:20:00	{ts '2002-12-21 15:20:00'}
	01.01.1904 15:20:00	{ts '1904-01-01 15:20:00'}
	Ein Text	'Ein Text'
RCStandardtag	15,6	15.6
	21.12.2002	{d '2002-12-21'}
	15:20	{INTERVAL '0 15:20:00' DAY(1) TO SECOND(2)}
	21.12.2002 15:20:00	{d '2002-12-21'}
	01.01.1904 15:20:00	{d '1904-01-01'}
	Ein Text	'Ein Text'
RCStandarduhrzeit	15,6	15.6
	21.12.2002	{t '00:00:00'}
	15:20	{INTERVAL '0 15:20:00' DAY(1) TO SECOND(2)}
	21.12.2002 15:20:00	{t '15:20:00'}
	01.01.1904 15:20:00	{t '15:20:00'}
	Ein Text	'Ein Text'

RCStandardFormat ist die Funktion, die im Regelfall einfach die richtige Konvertierung abliefert. Die grau hervorgehobenen Felder zeigen, in welchen Fällen sich die drei folgenden Varianten der Funktion unterscheiden. Hinweis: Von den ODBC-Typen DATE (d), TIME (t), TIMESTAMP (ts) und INTERVAL kennt RagTime intern die Varianten TIMESTAMP und INTERVAL: Daten mit Uhrzeit und Zeitspannen. Die Funktion RCStandardFormat konvertiert einen RagTime-Zeitpunkt zum DATE, wenn der Uhrzeitanteil 0 beträgt. Der Zeitpunkt wird zu TIME, wenn der Datumsanteil 0 beträgt (1.1.1904).

Bei allen datum- und zeitbezogenen Werten ist das Resultat eine ODBC-Escape-Sequenz. Datenbankdialekte unterscheiden sich bei diesen Feldtypen erheblich in der Notation. Eine einheitliche Konvertierung (d.h. ohne Spezialfunktionen für Oracle, Microsoft, Sybase…) ist nur so möglich. Die Escape-Sequenzen werden dann vom ODBC-Treiber in den Datenbank-Dialekt übersetzt.

Diese Normung in ODBC ist dennoch nutzlos, wenn eine Datenbank einen bestimmten Feldtyp gar nicht unterstützt. Besonders bei den INTERVAL-Typen (Zeitspannen in RagTime) sollte man damit rechnen.

Das Ausgangsbeispiel, SELECT StadtNameDeu FROM Staedte WHERE Laenge BETWEEN 0.3 AND 15.6, könnte im RagTime Connect Dialog wie folgt aussehen, wenn die Zahlenwerte für die Länge in den Zellen A1 und B1 von Rechenblatt Rechenblatt 1 eingegeben werden:



In den Eingabe-Feldern können dann normale Zahlenwerte benutzt werden, die Funktion RCStandardFormat wandelt sie in der WHERE-Klausel in geeigneten Text um.

Parametrisierte Abfragen

Die Aufgabe

Der folgende Abschnitt wird wahrscheinlich deutlicher, wenn zuerst die Aufgabe beschrieben wird, die diese Option in RagTime Connect lösen soll.

Angenommen sei, jemand will eine Preisliste gestalten. In Spalte A sollen Produktcodes eingetippt werden, in den Spalten rechts daneben sollen aus der Datenbank die Produktdaten wie Name, Preis, Abbildung... erscheinen.

In den bisher diskutierten Techniken mit RagTime Connect gibt es ein Problem in diesem Fall: Eine Abfrage in einer RagTime Connect Komponente liefert das Resultat einer Suche. Hier soll pro benutzter Rechenblattzeile ein andere Suche erfolgen. Es wäre sicher nicht handhabbar, pro Zeile eine eigene Abfrage einzurichten.

Für solche Fälle sieht RagTime Connect das Konzept vor, dass eine Abfrage von Funktionen aus parametrisiert wird und dann das Resultat für diese Parameter bei der Funktion abliefert. Funktionen in verschiedenen Zellen können unterschiedliche Parameter übergeben und erhalten dann von derselben Abfrage auch unterschiedliche Antworten. Man kann also eine Abfrage in unterschiedliche Varianten aufteilen. In der Beispieldatenbank kann die Konstruktion ähnlich nachgestellt werden: In Spalte A werden ISO-Codes für Länder eingegeben, in den Spalten B und C erscheinen der Ländername und die Fahne. Das mitgelieferte Beispiel heißt „Ausgewählte Länder“.

RCParamRef und RCParameter

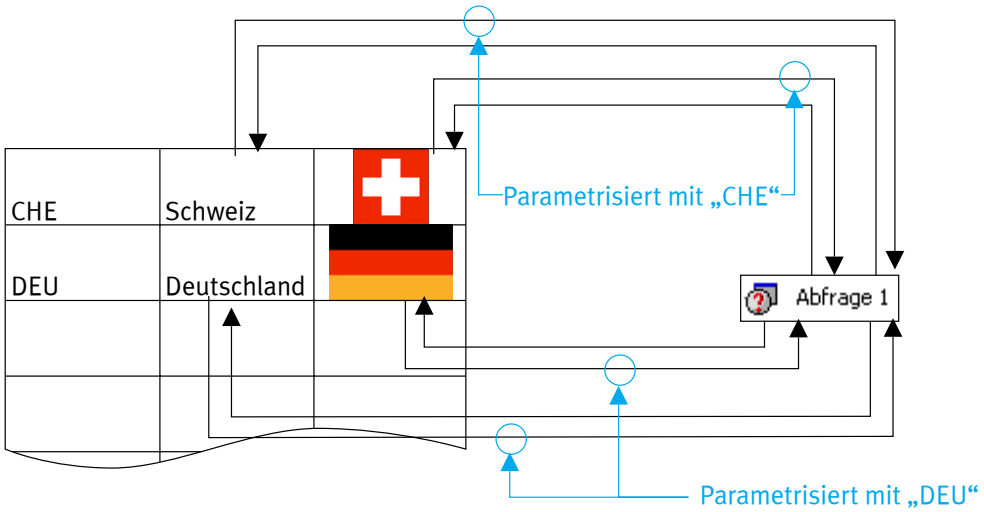
Versuch einer schematischen Darstellung: In der folgenden Abbildung stehen in der ersten Spalte eines Rechenblatts die beiden Ländercodes CHE und DEU. Zelle B1 enthält folgende Formel:

```
RCParamRef(Städte und Länder via ODBC!1.$1;$A1)
```

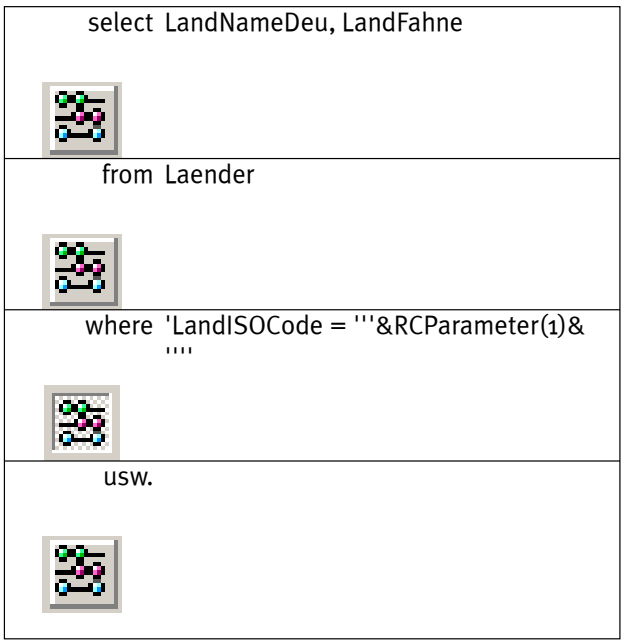
Die Funktion RCParamRef leistet die eine Seite des Vorgangs. Sie parametrisiert die Abfrage "Städte und Länder via ODBC" mit dem Wert aus A1. Als Ergebnis liefert sie den Wert, den die Abfrage mit dieser Parametrisierung liefert. In Zelle C1 steht fast dieselbe Formel:

```
RCParamRef(Städte und Länder via ODBC!2.$1;$A1)
```

Hier wird die zweite Spalte der Abfrage zurückgeliefert, sonst ist alles gleich. (Absolute und relative Bezüge wurden in diesen Formeln je so gewählt, dass ein Auffüllen nach rechts und unten möglich ist.)



In den Zellen in der zweiten Zeile stehen wieder fast dieselben Formeln, nur, dass sie sich auf A2 beziehen. Die zugehörige Abfrage sieht wie folgt aus:



Im WHERE-Teil steht die Funktion RCParameter(1). Diese Funktion ist nichts als ein Platzhalter. Jedesmal, wenn im Rechenblatt eine der Zellen mit der RCParamRef-Funktion gerechnet wird, nimmt die Platzhalter-Funktion den dort gesetzten Wert an. Mit diesem Wert rechnet dann die Abfrage. Das Resultat dieser konkreten Abfrage-Berechnung übernimmt dann RCParamRef.

Schrittweise aufgelistet: Wenn die Zelle B1 rechnet, übergibt sie an RCParameter den Wert CHE. Die Abfrage wird mit diesem Wert ausgeführt (SELECT LandNameDeu, LandFahne FROM Laender WHERE LandISOCode = 'CHE'). Das Ergebnis ist der Datensatz mit der Schweiz. Zelle, B1 erhält "Schweiz" als Antwort. Rechnet C1, übergibt sie wieder CHE an RCParameter, holt sich aus der Antwort aber die zweite Spalte, das JPEG mit der Fahne. Rechnet B2, übergibt die RCParamRef-Funktion DEU als Wert an RCParameter. Mit diesem Wert rechnet die Abfrage und findet den Datensatz zu Deutschland. Das erste Feld der ersten Spalte wird zurückgeliefert an die RCParamRef-Funktion und in der Zelle erscheint Deutschland. Usw.

Anders formuliert: Wird in einer Abfrage die Platzhalter-Funktion RCParameter benutzt, kann jede Zelle eines Rechenblatts mit Hilfe der Funktion RCParamRef diese Abfrage in einer eigenen Variante ausformen und auf das Ergebnis dieser Variante zugreifen. Füllt man die Formeln in den Spalten B und C nach unten auf, können entsprechend viele Landesinformationen über die eine Abfrage ermittelt werden.

Details

"Rechnen einer Abfrage" bedeutet nicht, dass jedesmal eine Anforderung an die Datenbank geschickt wird. Angenommen, man ersetzt DEU durch FIN, dann rechnet die Zelle C2 neu. Sie muss eine Frage an die Datenbank senden, um die Daten für Finnland zu gekommen. Wenn dann Zelle B2 rechnet, liegen diese Daten noch vor und die Abfrage liefert die Antwort aus ihrem Cache. Sollte RagTime zufällig die umgekehrte Rechenreihenfolge benutzen, ändert dies nichts. Auch wenn noch weitere Spalten benutzt werden, führt in dieser Konstruktion die Eingabe eines Ländercodes deshalb nur zu einem Datenbank-Kontakt.

Die Funktion RCParameter hat in unserem Beispiel das Argument 1. Es können in der Abfrage mehrere Platzhalter benutzt werden, diese sind dann RCParameter(2), RCParameter(3) usw. Die Werte, die sie annehmen, stehen als weitere Argumente in der Funktion RCParamRef. Grob also: RCParamRef(Referenz-auf-Abfrage;Wert-für-Parameter-1;Wert-für-Parameter-2;Wert-für-Parameter-3;...).

Wird eine RCParameter-Funktion in einer Abfrage benutzt und es gibt noch keine RCParamRef-Funktion, die ihren Wert setzt, fragt die Funktion in einem Dialog nach dem Wert, sobald die Abfrage ausgeführt wird. Diesen Dialog kann man mit einem Text versehen. Den Text bestimmt das optionale zweite Argument von RCParameter.

Damit sich durch Parametrisierungen die Struktur der Antworttabelle nicht ändert, darf RCParameter nicht im SELECT- und FROM-Teil einer Abfrage benutzt werden.

In unserer Beispiel-Konstruktion sind nach einer Ländercode-Eingabe immer genau die zugehörigen RCParamRef-Zellen von der Änderung abhängig. Gleich, wie RagTime seine Rechenreihenfolge in der Tabelle erzeugt, reicht immer ein Datenbank-Kontakt. Es gibt eine Situation, in der man dennoch auf Reihenfolgen achten sollte: Angenommen sei, es gibt im Dokument noch einen gemeinsamen Schalter, von dem alle Zellen abhängen. Ein Beispiel einer Preisliste könnte die Sprache sein. Je nach Sprachwahl sollen aus der Datenbank unterschiedliche Spalten benutzt werden. Wird dieses Beispiel angenommen, rechnet RagTime nach einer Änderung der Sprachwahl alle Zellen neu. Für den Cache-Mechanismus ist jetzt nicht unwichtig, ob je die Zellen zu einem Datensatz nacheinander gerechnet werden oder nicht. Das Resultat ist zwar dasselbe, nicht aber die Performanz.

Als Faustregel gilt: Gibt es für die Rechenreihenfolge keine besonderen Gründe basierend auf Abhängigkeiten, benutzt RagTime die Reihenfolge, in der die Formeln erzeugt wurden. Die zuletzt angelegte Formel rechnet zuerst. In einer Anordnung wie in unserem Beispiel erhält man eine günstige Reihenfolge, wenn man die Zelle B1 zuerst nach rechts auffüllt und dann die Zeile gemeinsam nach unten. Geht man anders herum vor, erhält man eine ungünstige Reihenfolge. N.b.: Dies spielt alles nur eine Rolle unter der Annahme, dass alle Zellen noch einmal von einer gemeinsamen Eingabe abhängen.

Bemerkungen zu FileMaker

Seit Version 5 kennt FileMaker Pro ODBC-Verbindungen. Zwei Richtungen sind möglich: Wie RagTime Connect kann FileMaker ein ODBC Client sein. Diese Funktionalität befindet sich im Import-Dialog. Umgekehrt kann FileMaker eine ODBC-Datenquelle sein. Programme wie RagTime können dann auf Daten in FileMaker-Dateien zugreifen. Die notwendige Software wird bei der einfachen Installation mit installiert. Bei den von uns getesteten Installern konnte die ODBC-Software für FileMaker nicht über eine angepasste Installation installiert werden.

FileMaker unterscheidet sich im Design — meist aus guten Gründen — erheblich vom SQL-Design, das die ODBC-Grundlage bildet. Deshalb gibt es eine Reihe von Besonderheiten.

Die FileMaker-Datenquelle

Damit auf FileMaker-Daten zugegriffen werden kann, muss unter "Bearbeiten ⇢ Voreinstellungen ⇢ Programm ⇢ Plugins" der "Local Data Access Companion" eingeschaltet werden. Danach kann unter "Datei/Ablage ⇢ Sharing" für jede einzelne FileMaker-Datei der Local Data Access Companion freigegeben werden. Die Gesamtheit aller von FileMaker geöffneten und unter Sharing freigeschalteten Dateien können zusammen als eine ODBC-Datenquelle benutzt werden. Greift man von RagTime Connect auf diese Datenquelle zu, erscheinen die FileMaker-Dateien als Tabellen in der Palette und unter jeder Tabellenbezeichnung die Felder der FileMaker-Datei. Es können keine Datenquellen für bestimmte Dateien definiert werden und es können keine geschlossenen Dateien benutzt werden.

Besonderheiten der ODBC-Einstellung

FileMaker-Textfelder enthalten bis zu 64000 Zeichen. Im Unterschied zu jeder SQL-Datenbank verwaltet FileMaker für jedes Textfeld dynamisch, wieviel Speicher gebraucht wird. Um Probleme beim ODBC-Zugriff zu vermeiden, begrenzt das ODBC-Kontrollfeld für FileMaker die Textlänge. Voreingestellt ist 255 Zeichen. Für Adressen u.ä. ist dies realistisch. Wer längere Texte aus FileMaker auslesen will, muss diese Zahl auf einen höheren Wert setzten.

Aufgerufene und ausgewählte Datensätze

Anwender von FileTime sind es gewohnt, dass sie in RagTime auf die in FileMaker aufgerufenen Datensätze bzw. den ausgewählten Datensatz zugreifen können. Die Auswahl in einem anderen Programm ist ein SQL-fremdes Konzept. RagTime Connect kann nicht abfragen, welche Datensätze in FileMaker ausgewählt oder aufgerufen sind. Resultat eines Zugriffs ist immer eine Suche, die von RagTime ausgehen muss.

Textsuchen

FileMaker hat für Textsuchen ein Indizierungsschema, dass kaum eine andere Datenbank kennt. Bei der Abbildung auf SQL ergibt dies Probleme. Namentlich Suchen mit LIKE werden extrem langsam abgearbeitet. Bei umfangreichen FileMaker-Dateien ist eine LIKE-Suche schlicht nutzlos.

Beziehungen

In FileMaker definierte Beziehungen zwischen Dateien können von ODBC aus nicht angesprochen werden und werden auch nicht gebraucht: Die Verknüpfung entsteht ad hoc durch das Join in der SQL-Anweisung. FileMaker unterstützt nicht die JOIN-Syntax im FROM-Teil, es muss die klassische Form mit der gleich-Abfrage im WHERE-Teil benutzt werden. Dabei sind auch Join-Operationen über mehr als zwei Dateien möglich, eine Technik, die FileMaker direkt nicht unterstützt. Wir haben für diese Technik keine Performanztests durchgeführt. Für Outer Joins steht die *=-Notation zur Verfügung.

FileMaker-Variablen

Aus ODBC-Sicht ist eine Variable in einer FileMaker-Datei ein Feld jedes Datensatzes. Variablen bilden damit eine Möglichkeit, Parameter-Übergaben von FileMaker zu RagTime Connect zu schaffen. Ein FileMaker-Script könnte z.B. eine Datensatznummer auf Knopfklick in eine Variable schreiben. Ein RagTime-Formular fragt in einer ersten Abfrage diese Variable für einen beliebigen Datensatz ab und sucht dann in einer zweiten Abfrage den Datensatz mit dieser Nummer. Damit kann ein Zugriff auf ausgewählte Datensätze zum Teil simuliert werden.

Anhang

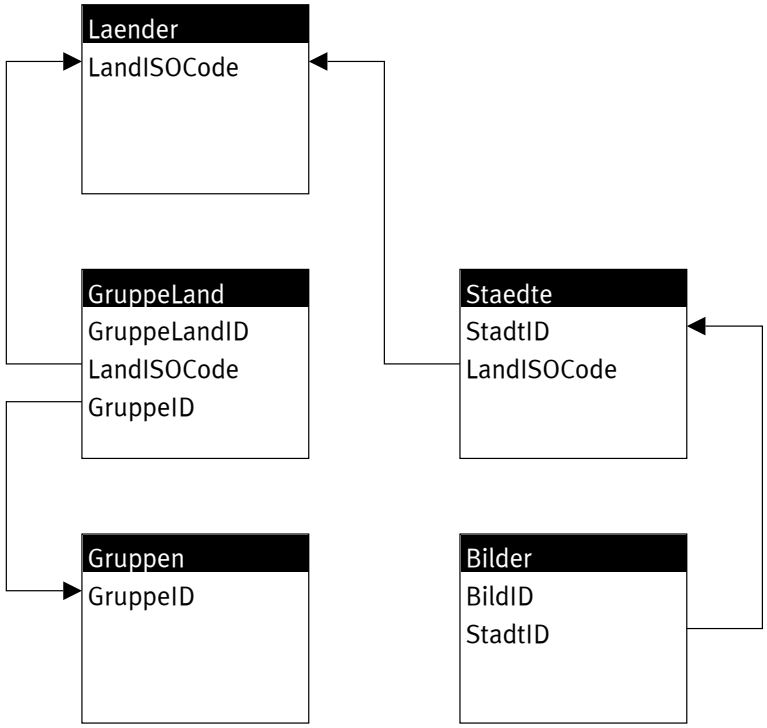
Struktur der Beispieldatenbank

Die Beispieldatenbank hat folgende Tabellenstrukur:

```
Tabelle Staedte
          StadtID                 (Ganzzahl)
          StadtNameDeu            (Text)
          StadtNameEng            (Text)
          StadtNameFra            (Text)
          LandISOCode             (Text)
          StadtBeschreibung       (Text)
          StadtKurzbeschreibung   (Text)
          Breite                  (Fließkommazahl)
          Laenge                  (Fließkommazahl)

Tabelle Laender
          LandISOCode             (Text)
          LandNameDeu             (Text)
          LandNameEng             (Text)
          LandNameFra             (Text)
          LandBeschreibung        (Text)
          LandKurzbeschreibung    (Text)
          LandFahne               (BLOB)
          LandFlaecheSI           (Fließkommazahl)
          LandFlaecheUS           (Fließkommazahl)
          LandGDPDollar           (Fließkommazahl)
          LandBevoelkerung        (Fließkommazahl)

Tabelle Bilder
          BildID                  (Ganzzahl)
          StadtID                 (Ganzzahl)
          BildTitel               (Text)
          BildText                (Text)
          BildInhalt              (BLOB)

Tabelle Gruppen
          GruppeID                (Ganzzahl)
          GruppeName              (Text)


Tabelle GruppeLand
          GruppeLandID            (Ganzzahl)
          GruppeID                (Ganzzahl)
          LandISOCode             (Text)
```

(Hinweis: SQL kennt wesentlich differenziertere Datentypen: Texte mit fester oder variabler Länge, Zahlen unterschiedlicher Genauigkeit etc. Für die konzeptionelle Einführung ist das nicht weiter wichtig.)

Die Idee dieser Struktur: Bilder sind Bilder von Städten. StadtID stellt die Verbindung zur Stadt her. Städte liegen in Ländern. Die Verbindung entsteht über LandISOCode. Länder gehören politischen Bündnissen an, dabei kann ein Land mehreren Bündnissen angehören, ein Bündnis umfasst mehrere Länder. Die Verbindung stellt die Tabelle GruppeLand her mit den beiden Schlüsseln GruppeID und LandISOCode. Grafisch dargestellt (Pfeile zeigen je vom Foreigh Key auf den Primary Key):



Die Tabelle „GruppeLand“ illustriert dabei die typische SQL-Modellierung einer viele-zu-viele-Beziehung: Eine Gruppe enthält mehrere Länder, ein Land gehört zu mehreren Gruppen. Ohne solche Zwischentabellen entstehen immer 1-zu-viele-Beziehungen.

Terminologie

Ein bischen Sprachregelung vorweg: SQL (Structured Query Language, übliche englische Aussprache: Sequel) ist eine Steuersprache für relationale Datenbanken. Zu Ihr gehören die Data Definition Language, mit der Datenbanken aufgebaut werden, die Data Management Language, mit der z.B. Zugriffsrechte bearbeitet werden und die Data Manipulation Language, mit der Daten abgefragt und bearbeitet werden. Im Kontext mit RagTime interessiert nur die Abfrage.

Eine Datenbank ist aus einer beliebigen Anzahl von Tabellen aufgebaut. Tabellen können untereinander Beziehungen haben. In der Beispieldatenbank ist für jedes Bild die Stadt als deren ID vermerkt. Eine eindeutige ID (oder andere Kennung) eines Datensatzes heißt Primärschlüssel (Primary Key). Wird ein Primary Key einer Tabelle in einer anderen benutzt, um eine Beziehung herzustellen, heißt er dort Foreign Key. Für FileMaker-User: Eine Tabelle einer relationalen Datenbank entspricht einer FileMaker-Datei.

Diese Beziehungen (englisch: relationship) haben terminologisch nichts mit dem Wort „relational“ zu tun. Relation (englisch: relation) ist ein anderer Terminus für Tabelle, aber auf einer mathematischen Sprachebene. „größer“ ist eine Relation, die u.a. zwischen natürlichen Zahlen definiert ist. Man kann die Relation auch als eine Menge repräsentieren: Die Menge aller Zahlenpaare, die z.B. [1;2], [1;3] enthält, aber nicht [2;2] oder [2;1]. Eine dreistellige Relation wie „liegt zwischen“ kann man als Menge der Tripel wie [2:1:3] darstellen. Das relationale Datenbankkonzept wurde aus diesem Modell von Relationen entwickelt. Jede Relation ist eine Menge von n-tupeln. Alle Elemente einer Menge sind dabei gleich strukturiert.

Die traditionellen Sprachebenen bei der Beschreibung relationaler Datenbanken sind dabei: a)Relation, Tupel und Attribut, b) Tabelle, Zeile und Spalte, c) Datei, Datensatz und Feld. Die erste Form betont das mathematische Konzept und wirkt oft akademisch. Das zweite ist am gängigsten, wenn von SQL-Daten die Rede ist. Es betont, dass die Tabellen technisch gleichgeordnet sind, durchaus ein neues Konzept als SQL eingeführt wurde. Vorherige Konzepte hatten meist eine hierarchische Struktur der Dateien. Der Nachteil dieser Sprechweise im Kontext mit RagTime ist klar: Man muss laufend klar machen, ob von einer RagTime-Tabelle oder einer SQL-Tabelle die Rede ist. Die Verwendung von Datei, Datensatz und Feld ist damit oft einfacher, aber etwas untypisch im Kontext relationaler Datenbanken. In der Literatur werden die Ebenen Tabelle / Datei selten sauber getrennt, die Ebene Relation aber schon. In einem Atemzug von Relation und Zeile oder Datensatz und Attribut zu sprechen, ist etwas ungeschickt. Ebenenmischungen wie "Feld einer Zeile" werden auch in diesem Text benutzt.

Mengen haben zwei Eigenschaften, die sie von Tabellen unterscheiden: Sie können keine Elemente doppelt enthalten und die Elemente haben keine Reihenfolge. Im Rechner sind die Daten naturgemäß eher wie Tabellen strukturiert: Doppeleinträge sind technisch möglich und Einträge liegen in einer Reihenfolge vor. Dennoch gilt die Regel: Auf die Reihenfolge der Datensätze in der Datenbank darf man sich nicht verlassen. Für jede Abfrage darf das System eine andere Strategie und damit ggf. andere Reihenfolge liefern. Nur wenn ausdrücklich mit einer ORDER BY-Klausel eine Sortierung erzeugt wurde, ist die Reihenfolge definiert. Mit dem Schlüsselwort DISTINCT können Doppel unterdrückt werden. In dem Fall verhalten sich die Operationen auf der Datenbank tatsächlich wie Mengenoperationen.

Alle Suchen auf einer SQL-Datenbank werden mit der SELECT-Anweisung ausgeführt. Da der RagTime-Abfrage-Editor immer am Anfag das Schlüsselwort SELECT einsetzt, sind andere Anweisungen wie INSERT, UPDATE, DELETE nicht möglich.

Select-Statements

```
SELECT *
FROM Staedte
```

liefert die ganze Tabelle, so, wie in Datenbank technisch repräsentiert. Außerhalb von Testsituationen sollten die Felder explizit genannt werden, weil dann in RagTime eine definierte Liste inklusive Reihenfolge ankommt, die sich auch bei späteren Erweiterungen der Datenbanktabelle nicht ändert:

```
SELECT StadtNameDeu, Breite, Laenge
FROM Staedte
```

Drei Spalten der ganzen Tabelle werden zurückgeliefert.

```
SELECT StadtNameDeu, Breite, Laenge
FROM Staedte
WHERE StadtNameDeu LIKE 'New%'
```

Aus der Tabelle werden die Zeilen herausgesucht, in denen der Name mit „New“ anfängt und von dieser Teiltabelle werden drei Felder zurückgeliefert.

Logische Operatoren im Suchkriterium:

```
=
<> oder !=
<
>
<=
>=
LIKE  (Nur in Text)
```
Wildcards in LIKE-Text: _: ein Buchstabe, %: o oder mehrere Buchstaben
```
IN <Liste>
```
Liste ist dabei ein Klammerausdruck mit Werten, Kommagetrennt oder ein SELECT-Statement, das ein Feld abliefert.
```
BETWEEN <kleinster Wert> AND <höchster Wert>
AND
OR
NOT
IS [NOT] NULL
```

Tabellen verknüpfen

Eine Idee relationaler Datenbanken ist, dass in einer Tabelle die Werte einer Spalte nicht notwendig die einer anderen implizieren dürfen. Wenn zu einem Bild vermerkt ist, dass es ein Objekt in New York zeigt, darf in derselben Tabelle nicht mehr stehen, welche Einwohnerzahl New York hat. Dies gehört in eine separate Tabelle. Anderenfalls könnten verschiedene Bilder aus New York unterschiedliche Einwohnerzahlen enthalten. Dann wäre die Konsistenz der Daten nicht mehr sichergestellt.

Obwohl die Beziehungen zwischen Tabellen ein zentrales Konstrukt für relationale Datenbanken bilden, enthielten die ersten SQL-Versionen keine eigene Anweisung dafür. Die Verknüpfung zwischen Tabellen wurde implizit in der Suchbedingung hergestellt. Später wurde SQL um ein explizites JOIN erweitert. Ein Zitat aus dem aktuellen Benutzerhandbuch von PostgreSQL zu diesen zwei Techniken:

„Welche von diesen Sie benutzen ist hauptsächlich eine Sache des Stils. Die JOIN-Syntax in der FROM-Klausel ist wahrscheinlich nicht so portabel auf andere Produkte.“

Diese fehlende Portabilität gilt unter anderem für FileMaker. Deshalb sind alle Beispiele in der „Dinosaurier“-Syntax angelegt. Für einen SQL-Server sollte die Syntax keine Rolle spielen, wie die Autoren des PostgreSQL-Handbuchs bemerken. Man sollte aber nicht ausschließen, dass einige Server mit einem ausdrücklichen JOIN klüger umgehen und die Anweisung schneller abarbeiten.

Die drei Ausgangstabellen, auf denen ein Join ausgeführt wird

Laender

ID	Name
1	USA
2	Sverige

Stadte

ID	Name	LandID
1	Stockholm	2
2	New York	1
3	New Orlea	1
4	Borås	2

Bilder

ID	Name	StadtID
1	WTC	2
2	Vieux Carré	3
3	Gamla Stan	1

Kreuzprodukt komplett: Alle möglichen Kombinationen aus den Tabellen wurden gebildet
Zeilen, bei denen StadtID und Staedte.ID gleich sind, wurden markiert

Laender.ID	Laender.Name	Staedte.ID	Staedte.Name	LandID	Bilder.ID	Bilder.Name	StadtID
1	USA	1	Stockholm	2	1	WTC	2
1	USA	1	Stockholm	2	2	Vieux Carré	3
1	USA	1	Stockholm	2	3	Gamla Stan	1
1	USA	2	New York	1	1	WTC	2
1	USA	2	New York	1	2	Vieux Carré	3
1	USA	2	New York	1	3	Gamla Stan	1
1	USA	3	New Orleans	1	1	WTC	2
1	USA	3	New Orleans	1	2	Vieux Carré	3
1	USA	3	New Orleans	1	3	Gamla Stan	1
1	USA	4	Borås	2	1	WTC	2
1	USA	4	Borås	2	2	Vieux Carré	3
1	USA	4	Borås	2	3	Gamla Stan	1
2	Sverige	1	Stockholm	2	1	WTC	2
2	Sverige	1	Stockholm	2	2	Vieux Carré	3
2	Sverige	1	Stockholm	2	3	Gamla Stan	1
2	Sverige	2	New York	1	1	WTC	2
2	Sverige	2	New York	1	2	Vieux Carré	3
2	Sverige	2	New York	1	3	Gamla Stan	1
2	Sverige	3	New Orleans	1	1	WTC	2
2	Sverige	3	New Orleans	1	2	Vieux Carré	3
2	Sverige	3	New Orleans	1	3	Gamla Stan	1
2	Sverige	4	Borås	2	1	WTC	2
2	Sverige	4	Borås	2	2	Vieux Carré	3
2	Sverige	4	Borås	2	3	Gamla Stan	1

Nach Reduktion auf StadtID = Staedte.ID
(Zeilen markiert, in denen LandID = Laender.ID)

Laender.ID	Laender.Name	Staedte.ID	Staedte.Name	LandID	Bilder.ID	Bilder.Name	StadtID
1	USA	1	Stockholm	2	3	Gamla Stan	1
1	USA	2	New York	1	1	WTC	2
1	USA	3	New Orleans	1	2	Vieux Carré	3
2	Sverige	1	Stockholm	2	3	Gamla Stan	1
2	Sverige	2	New York	1	1	WTC	2
2	Sverige	3	New Orleans	1	2	Vieux Carré	3

Nach Reduktion auf LandID = Laender.ID

Laender.ID	Laender.Name	Staedte.ID	Staedte.Name	LandID	Bilder.ID	Bilder.Name	StadtID
1	USA	2	New York	1	1	WTC	2
1	USA	3	New Orleans	1	2	Vieux Carré	3
2	Sverige	1	Stockholm	2	3	Gamla Stan	1

Nach Suche auf USA

Laender.ID	Laender.Name	Staedte.ID	Staedte.Name	LandID	Bilder.ID	Bilder.Name	StadtID
1	USA	2	New York	1	1	WTC	2
1	USA	3	New Orleans	1	2	Vieux Carré	3

Inner Joins

Verknüpfungen von Tabelle werden nach alter Syntax erzeugt, indem im FROM-Teil mehrere Tabellen aufgeleistet werden. Entstehen bei den Feldnamen dadurch Doppeldeutigkeiten, wird im SELECT- und WHERE-Teil der Tabellenname mit einem Punkt abgetrennt vorangestellt. Um Tippen zu sparen, können dabei im FROM-Teil Aliasnamen zugeordnet werden.

```
SELECT StadtNameDeu, StadtKurzbeschreibung, LandNameDeu
FROM Staedte S, Laender L
WHERE (S.LandISOCode = L.LandISOCode) AND (LandNameDeu like 'Fr%')
```

Konzeptionell erzeugt hier der FROM-Teil das Kreuzprodukt aus zwei Mengen: Die Einträge aller Tabellen werden mit den Einträgen der beiden anderen Tabellen zu einer neuen kombiniert. Der Teil „(S.LandISOCode = L.LandISOCode)“ im WHERE-Teil eliminiert daraus alle sinnlosen Zeilen. Es bleibt pro Stadt ein Datensatz übrig, der um die zugehörigen Informationen zu Stadt und Land erweitert ist. Auf dieser Liste wird die eigentliche Suche durchgeführt: `(LandNameDeu like 'Fr%')`. Das Resultat sind alle Städte der Datenbank, die in Frankreich liegen. Zurückgeliefert werden aus dieser Liste nur die Felder StadtNameDeu und StadtKurzbeschreibung aus Staedte sowie der Landesname aus Laender.

Am Beispiel sehr einfacher Datenbanktabellen haben wir den konzeptionellen Vorgang in einem Rechenblatt veranschaulicht. Das Datenbank-Programm sorgt dafür, dass das Resulat dem Konzept entspricht, aber das Kreuzprodukt nicht wirklich erzeugt wird.

Seit der Einführung von SQL wurde an dieser Konstruktion als Mangel empfunden, dass der WHERE-Teil die eigentliche Suche und die Verknüpfung der Tabellen mischt. Der SQL-92-Standard schlägt deshalb eine explizite JOIN-Anweisung im FROM-Teil vor:

```
SELECT StadtNameDeu, StadtKurzbeschreibung, LandNameDeu
FROM Staedte S INNER JOIN Laender L ON S.LandISOCode = L.LandISOCode
WHERE (LandNameDeu like 'Fr%')
```

In dieser Syntax wird im FROM-Teil explizit eine neue Tabelle als Verknüpfung der beiden Ursprungstabellen gebildet und auf dieser die Suche durchgeführt. Nicht alle Datenbankprogramme unterstützen diese Syntax.

Es kann mehr Varianten geben. Die folgende Syntax mit dem Schlüsselwort NATURAL INNER JOIN akzeptiert beispielsweise PostgreSQL:

```
SELECT StadtNameDeu, StadtKurzbeschreibung, LandNameDeu
FROM Staedte NATURAL INNER JOIN Laender
WHERE (LandNameDeu like 'Fr%')
```

Die Datenbank erkennt dann, dass in den zwei Tabellen Staedte und Laender genau die Spalte LandISOCode mit gleichem Namen vorkommt und als Join-Kriterium benutzbar ist. Lesen Sie ggf. in der Dokumentation der Datenbank nach.

Zur Terminologie: Die Operation, eine Tabelle auf bestimmte Spalten zu reduzieren (die Feldnamenliste im SELECT-Teil) läuft unter „Projektion“. Die Bildung des Kreuzprodukts im FROM-Teil heißt Join (oft wird Join auch für das Kreuzprodukt zusammen mit der Beschränkung auf den sinnvollen Teil der Tabelle im WHERE-Abschnitt bezeichnet.) Die Auswahl bestimmter Datensätze im WHERE-Teil heißt Selektion.
JOINs können mehr als zwei Tabellen umfassen. In der Beispieldatenbank gibt es eine "viele zu viele"-Beziehung, die Zugehörigkeit von Ländern zu Bündnissen. Solche Beziehungen werden durch eine Zwischentabelle modelliert. Als Beispiel sollen die Länder gefunden werden, die zur Gruppe "G7" gehören:

```
SELECT LandNameDeu
```

```
FROM (Gruppen G INNER JOIN GruppeLand GL ON G.GruppeID = GL.GruppeID)
         INNER JOIN Laender L ON GL.LandISOCode = L.LandISOCode
WHERE GruppeName = 'G7'
```

Hier wird die Verknüpfung über drei Tabellen geführt: Zuerst wird aus "Gruppen" und "GruppeLand" eine Tabelle erzeugt. Diese Tabelle enthält u.a. den Länderschlüssel aus GruppeLand und den Gruppennamen in Gruppen. Diese Tabelle wird dann wieder mit den Ländern verknüpft, um auch die Ländernamen zu erhalten. In klassischer Syntax (ohne JOIN):

```
SELECT LandNameDeu
FROM Gruppen G, GruppeLand GL, Laender L
WHERE (G.GruppeID = GL.GruppeID) AND (GL.LandISOCode = L.LandISOCode)
         AND GruppeName = 'G7'
```

Outer Joins

Betrachten wir eine Suche, die alle Städte und die Titel der zu ihnen vorliegenden Bilder zurück liefert:

```
SELECT StadtNameDeu, BildTitel
FROM Staedte S INNER JOIN Bilder B ON B.StadtID = S.StadtID
```

Das Ergebnis ist in einem Punkt fragwürdig: In der Beispieltabelle gibt es Städte, Dublin und Berlin, zu der kein Bild vorliegt. Sie fehlen in der Ergebnistabelle. Ist das Ziel, dass alle Städte erscheinen, ggf. mit den Bildinformationen, braucht man einen "Outer Join"

```
SELECT StadtNameDeu, BildTitel
FROM Staedte S LEFT OUTER JOIN Bilder B ON B.StadtID = S.StadtID
```

"Left" legt dabei fest, dass es die Tabelle "STAEDTE" ist, von der jeder Datensatz berücksichtigt werden soll. Bei Datenbankprogrammen, die diese Syntax nicht unterstützen, wird im WHERE-Teil ein spezieller Gleichheits-Operator benutzt. Verbreitet sind *= und =* für left und right Join.:

```
SELECT StadtNameDeu, BildTitel
FROM Staedte S, Bilder B
WHERE S.StadtID *= B.StadtID
```

Einige Datenbanken kennen auch ein Full Outer Join. Es wird dann jede Zeile aus jeder am Join beteiligten Tabellen benutzt.

Verknüpfung einer Tabelle mit sich selbst

Alias-Namen für Tabellen sind in einem Fall nicht nur bequem, sondern auch syntaktisch notwendig: wenn eine Tabelle mit sich selbst verknüpft werden soll. Für eine solche innere Beziehung gibt die Beispieldatenbank nicht viel her, deshalb konstruiere ich hier ein ad-hoc-Beispiel:

Eine Tabelle enthalte Textabschnitte, von denen jeder ein Unterpunkt zu einem anderen sein kann:

```
Tabelle TextAbschnitte
ID
Titel
```

```
Body
Parent
```

Ist ein Eintrag ein Unterabschnitt zu einem anderen, steht in der Spalte Parent die ID des übergeordneten Abschnitts. Nur Top-Level Abschnitte (=Einleitungen zu Artikeln) haben in dieser Spalte einen NULL-Wert. Zu einem beliebigen Abschnitt — nehmen wir willkürlich Eintrag Nr. 150 — sollen der übergeordnete Abschnitt und alle untergeordneten gesucht werden.

```
SELECT B.ID, B.Titel
FROM TextAbschnitte A, TextAbschnitte B
WHERE (A.ID = B.Parent) AND (A.ID = 150)
```

liefert die Liste der Unterabschnitte.

```
SELECT B.ID, B.Titel
FROM TextAbschnitte A, TextAbschnitte B
WHERE (B.ID = A.Parent) AND (A.ID = 150)
```

liefert dagegen den übergeordneten Abschnitt zu Abschnitt 150.

Dieses Beispiel illustriert, wie man allgemein eine Hierarchie in einer Tabelle modellieren kann. Wichtig dabei ist: In diesem Modell enthält kein Eintrag eine Information darüber, auf welcher Hierarchie-Ebene er sich befindet (ausgenommen Top-Level-Einträge, die klar durch den NULL-Wert in der Spalte Parent gekennzeichnet sind). Spezielle Anweisungen zum Abarbeiten solcher Baum-Strukturen kennen nur einige Datenbanken.

Viele-zu-Viele-Beziehungen

Bezieht sich ein Foreign Key einer Tabelle auf einen Primary Key einer anderen, kann damit nur eine 1-zu-viele-Beziehung modelliert werden. In der Beispiel-Datenbank ist das Verhältnis von Staedte und Laender dieser Art. Der dreibuchstabige ISO-Code eines Landes bildet den Primary Key in der Laender-Tabelle. Derselbe Key in der Tabelle Staedte ist dort nicht eindeutig. Deshalb können sich mehrere Städte auf ein Land beziehen, aber nicht umgekehrt.

Politische Bündnisse sind in der Beispieldatenbank ein Fall einer viele-zu-viele-Beziehung. Jedes Bündnis besteht aus mehreren Ländern, aber jedes Land gehört auch mehreren Bündnissen an. In relationalen Datenbanken wird dies durch eine „Zwischentabelle“ modelliert. Eine eigene Tabelle – GruppeLand in unserem Fall – enthält pro Land/Bündnis-Paar einen Datensatz. Dieser Datensatz vermerkt den Primärschlüssel des Landes und den des Bündnisses. Solche Zwischendateien können zusätzliche Spalten enthalten, wenn es Informationen gibt, die diese Paarung betreffen. Unser Beispiel könnte erweitert werden um das Eintrittsdatum in der Tabelle GruppeLand.

Beispiel: Es sollen alle Länder der G7-Gruppe herausgesucht werden. Der Name der Gruppe ist in der Tabelle Gruppen enthalten, die Namen der Länder in Laender. Die Verbindung wird von GruppeLand hergestellt.

```
SELECT LandNameDeu
FROM (Laender L INNER JOIN GruppeLand GL ON L.LandISOCode = GL.LandISOCode)
      INNER JOIN Gruppen G ON GL.GruppeID = G.GruppeID
WHERE GruppeName = 'G7'
```

Der From-Teil gleicht zunächst die Länder gegen die Zwischentabelle ab, dann wird das Resultat gegen die Gruppen abgeglichen.

In klassischer Form ohne JOIN lautet die Abfrage:

```
SELECT LandNameDeu
FROM Laender L, GruppeLand GL, Gruppen G
WHERE (L.LandISOCode = GL.LandISOCode) AND (GL.GruppeID = G.GruppeID) AND
(GruppeName = 'G7')
```

Weitere Elemente des SELECT-Teils

Duplikate vermeiden

Die Projektion hat ein Problem. Der Idee nach sind Relationen Mengen. Technisch sind sie Tabellen. Ein begrifflicher Unterschied: In Tabellen kann dieselbe Zeile mehrfach vorkommen, eine Menge kann aber nicht zweimal dasselbe Element enthalten. Auch, wenn in einer Ursprungstabelle alle Zeilen eindeutig sind, ist das nach weglassen einiger Spalten nicht mehr unbedingt der Fall. Mit dem Schlüsselwort DISTINCT werden Doppel unterdrückt.

```
SELECT DISTINCT LandISOCode
FROM Staedte
```

liefert die Liste aller Länder-IDs, für die Städte in der Datenbank vorhanden sind. Keine ID wird dabei doppelt aufgeführt.

Felder umbenennen

Die Feldnamen in der Datenbank müssen nicht unbedingt in RagTime benutzt werden. Mit AS können im SELECT-Teil Umbenennungen vorgenommen werden.

```
SELECT StadtNameDeu AS StadtName, LandISOCode as LaenderCode
FROM Staedte
```

liefert die Felder Name und ID aus der Tabelle Staedte. Im RagTime-Inventar erscheinen sie aber als StadtName und LaenderCode. Solche Umbenennungen sind nötig, wenn Berechnungen in der Abfrage durchgeführt werden oder wenn Namensdoppeldeutigkeiten vermieden werden sollen.

Berechnungen

Mit Feldern kann direkt in der Abfrage gerechnet werden:

```
SELECT LandGDPDollar / LandBevoelkerung * 1000000000 AS ProKopfGDP
from Laender
```

liefert in RagTime ein Feld namens ProKopfGDP für alle Länder ab. Die Berechnung wird dabei von der Datenbank und nicht von RagTime ausgeführt.

Rechenoperatoren sind:

```
+
-
*
/
|| (oder + oder &) (= Konkatenation)
```

Es gibt daneben diverse Standardfunktionen wie POWER, ROUND, TRUNC, MOD, ...

Aggregate

SQL bietet eine Reihe von Funktionen, die eine ganze Tabellenspalte auswerten. SUM ist ein Beispiel:

```
SELECT SUM(LandBevoelkerung) AS AlleMenschen
from Laender
```

Standard-Aggregatfunktionen sind:

```
AVG
SUM
MIN
MAX
COUNT
```

Die WHERE-Klausel

Die Vergleichsoperationen im WHERE-Teil können wiederum SELECT enthalten. Häufig ist die Kombination mit IN:

```
SELECT StadtNameDeu
FROM Staedte
WHERE LandISOCode IN
          (SELECT LandISOCode
          FROM Laender
          WHERE LandBevoelkerung > 100000)
```

sucht alle Städte aus Ländern mit mehr als 100000 Einwohnern.

Weitere Elemente des SELECT-Statements sind:

GROUP BY

Mit der Anweisung GROUP BY wird die Antworttabelle eines SELECT-Statements anhand von Feldwerten zu Gruppen zusammengefasst. Diese Gruppen haben insbesondere eine Wirkung auf Aggregatfunktionen. Die Aggregate werden für jede Gruppe einzeln berechnet:

```
SELECT GruppeID, AVG(LandGDPDollar) AS Durchschnitt
FROM Laender L INNER JOIN GruppeLand GL ON L.LandISOCode = GL.LandISOCode
GROUP BY GruppeID
```

Für jedes der in GruppeLand verzeichneten politischen Bündnisse wird das durchschnittliche GDP ermittelt.

Solche Gruppen können wieder durch eine Suche geschickt werden. Das Schlüsselwort ist HAVING.

```
SELECT GruppeID, AVG(LandGDPDollar) AS Durchschnitt
FROM Laender L INNER JOIN GruppeLand GL ON L.LandISOCode = GL.LandISOCode
GROUP BY GruppeID
HAVING COUNT(*) > 5
```

Das Ergebnis listet nur das Durchschnitts-GDP für Bündnisse mit mehr als 5 Mitgliedern. Der Stern im Argument ist dabei in syntaktischer Trick: Hier steht eine Spaltenbezeichnung. Will man alle Datensätze zählen, unabhängig davon, ob irgendwo NULL-Werte stehen, benutzt man den Stern.

Standard-Mengenoperationen

Wenn zwei SELECT-Statements Tabellen derselben Struktur abliefern (gleiche Anzahl von Feldern, gleiche Datentypen), kann man sie vereinigen, den Durchschnitt bilden oder die Differenzmenge. Die Syntax ist:

```
(UNION | INTERSECT | MINUS) SELECT...
```

Es wird also mit dem entsprechenden Schlüsselwort einfach ein zweites SELECT-Statement angehängt.

Sortieren

Eine Sortieranweisung muss immer ganz am Ende stehen. Auch bei Aktionen wie UNION kann nur nach dem letzten SELECT eine Sortierung aufgerufen werden. Die Syntax ist:

```
ORDER BY <Feldname> [ASC | DESC][, <Feldname2> [ASC | DESC]]…
```

„Feldliste“ ist dabei liberal gemeint. Es können auch Rechenoperationen zwischen Feldern benutzt werden. Die Angabe ASC(ending) oer DESC(ending) ist optional. Fehlt die Angabe, wird meist aufsteigend sortiert.

SQL und ODBC-Escape-Sequenzen

Bei aller Normierung unterliegt SQL einer Reihe von Dialekten. Nebenbei erwähnt wurde der Konkatenationsoperator: In SQL 92 ist es ein doppelter vertikaler Strich (||). So muss er auch mit einem Oracle-Server benutzt werden. Bei Microsoft SQL muss es + heißen. Access akzeptiert + und &.

Am weitesten auseinander gehen die Formen, in denen Kalenderdaten und Uhrzeiten dargestellt werden müssen. Um zugleich zu erlauben, alle Features einer bestimmten Datenbank zu nutzen aber zugleich ein vom konkreten DBMS unabhängiges Vorgehen zu erlauben, verfolgt ODBC einen Kompromiss:

Der SQL-Code in einem Select-Statement wird vom ODBC-Treiber einfach durchgereicht, ohne weitere Veränderung. Man kann also eine Anfrage im Dialekt des Servers stellen. Zusätzlich gibt es spezielle ODBC-Escape-Sequenzen. Sie werden vom Treiber in den jeweiligen Dialekt der benutzten Datenbank übersetzt.

Escape-Sequenzen stehen in geschweiften Klammern (ein Zeichen, dass in SQL nicht vorkommt). Nach der öffnenden Klammer stehen ein oder zwei Buchstaben, die die Art der Escape-Sequenz festlegen. Beispiele:
Für Kalenderdaten kann die Schreibweise
{d 'yyyy-mm-dd'}
benutzt werden. Zeiten lassen sich eingeben als
{t 'hh:mm:ss}
und Kombinationen aus beidem (Timestamp) als
{ts 'yyy-mmm-hh:mm:ss}

Beim Timestamp kann optional nach den Sekunden noch eine Kommastelle folgen, eingeben als „.fxxx“ (wobei x für Ziffern steht).

Funktionen haben die Kennung „fn“. Beispiel: Gegeben sei eine Tabelle „Namen“ mit u.a. den Spalten „Datensatznr“, „Nachname“ und „Vorname“. Vor- und Nachname sollen mit einem Leerzeichen dazwischen verbunden werden. In ODBC ist unabhängig von der Datenbank die Syntax möglich:

```
SELECT Datensatznr, {fn CONCAT(Vorname, ' ', Nachname)} AS GanzerName
FROM Namen
```

Schickt der ODBC-Treiber diese Anfrage an einen Oracle-Server, übersetzt er in:

```
SELECT Datensatznr, Vorname || ' ' || Nachname AS GanzerName
FROM Namen
```

Für einen Microsoft SQL-Server wird dieselbe Sequenz übersetzt als:

```
SELECT Datensatznr, Vorname + ' ' + Nachname AS GanzerName
FROM Namen
```

Zusammengefasst: Benutzt man in einem ODBC-Client-Programm SQL, reicht der Treiber die Anweisungen einfach durch. Mit Escape-Sequenzen kann man explizit die Übersetzung einer Standard-Operation in den jeweiligen SQL-Dialekt in Auftrag geben. Das Client-Programm ist gegenüber dem Wechsel zwischen SQL- und ODBC-Syntax wiederum agnostisch und reicht einfach an den Treiber weiter.