Fachbereichsspezifischer Entwicklungsansatz zur bedarfsgerechten Analyse SQL-basierter Datenquellen

Christoph Wildensee

Viele datenanalytisch arbeitende Unternehmensbereiche führen nicht selten an, dass sie, was den lesenden IT-System-Datenzugang angeht, nicht autark arbeiten können. Sie sind in vielen Häusern auf die Datenlieferung zentral durch den IT-Betrieb o.Ä. angewiesen und weisen insoweit auch selten Mittel auf, hier aktiver für eine Änderung zu sorgen. Dabei gibt es sehr wohl Wege, Datenzugriffe trotz lediglich einer minimalen Unterstützung durch den IT-Betrieb bedarfsgerecht zu etablieren. Vorausgesetzt, die Unternehmensbereiche sind bereit, in gewissem Umfang Eigenentwicklung zu betreiben und nicht nur Standardsoftwareproduktfunktionalität zu nutzen, können sie technisch anspruchsvolle und dabei von der Fragestellung her nahezu beliebig maßgeschneiderte Lösungen konzipieren und realisieren. Auch wenn in den meisten Häusern das Thema dezentrale Softwareentwicklung ein rotes Tuch ist, birgt es - in sinnvolle Grenzen eingebettet - die Möglichkeit, die IT zu entlasten und den Fachbereichsfunktionsumfang bedarfsgerecht zu optimieren.

Grundlagen

In der Literatur wird häufig zwischen „Continuous Monitoring“ (CM) und „Continuous Auditing“ (CA) unterschieden. Preuss [(2012), S. 5] hat dies kurz und prägnant dargestellt. Während CM eine grundlegende Aufgabe des Linien-Managements im Rahmen des sicherzustellenden IKS ist (die Revision kann natürlich bei der Entwicklung beratend zur Seite stehen) und als Beurteilung und Bearbeitung von identifizierten und bestätigten Exceptions charakterisiert werden kann, ist CA eine prozessunabhängige, regelmäßig wiederkehrende, also permanent automatisierte Identifizierung von Control Exceptions in den Prozessen und dazugehörigen Kontrollen durch die Revision. Die Feststellungen müssen mit Fachkenntnis beurteilt werden, sodass daraus eine Maßnahmenableitung erfolgen kann. Der „Continuous Auditing“-Ansatz bezeichnet also einen „systematischen Prozess, um zeitnah unbekannte Risiken und Schwächen von internen Kontrollen festzustellen.“ [Gehrig-Ehrenzeller (2011), S. 132] Dies ist allerdings nicht ausreichend differenziert. “CM is an automated, ongoing process that enables management to: Access the effectiveness of controls and detect associated risk issues; Improve business processes and activities while adhering to ethical and compliance standards; Execute more timely quantitative and qualitative risk-related decisions; Increase the cost-effectiveness of controls and monitoring through IT solutions. CA is an automated, ongoing process that enables internal audit to: Collect from processes, transactions, and accounts data that supports internal and external auditing activities; Achieve more timely, less costly compliance with policies, procedures, and regulations; Shift from cyclical or episodic reviews with limited focus to continuous, broader, more proactive reviews; Evolve from traditional, static annual audit plan to a more dynamic plan based on CA results; Reduce audit costs while increasing effectiveness trough IT solutions.” [Deloitte (2010), S. 3] (CM ist ein automatisierter, kontinuierlicher Prozess, der es dem Fachbereichsmanagement ermöglicht: Feststellung der Wirksamkeit von Kontrollen und Erkennen damit verbundener Sachverhalte; Verbesserung von Geschäftsprozessen und -aktivitäten unter Einhaltung von Ethik- und Compliance-Standards; zeitnahe quantitativ und qualitativ risikobezogene Entscheidungen; Erhöhung der Kosteneffizienz von Kontrollen und Überwachung durch IT-Lösungen. CA ist ein automatisierter, laufender Prozess, der es der Internen Revision ermöglicht: Prozess-, Transaktions- und Buchhaltungsdaten zu nutzen, um die interne und externe Revisionstätigkeit zu unterstützen; schnellere Überprüfung der Einhaltung von Richtlinien, Verfahren und Vorschriften; Umstellung von zyklischen oder episodischen Überprüfungen mit eingeschränktem Fokus auf kontinuierliche, von der Datenbasis her breitere, proaktive Überprüfungen; Entwicklung vom traditionellen, statischen Jahresprüfungs-

plan der Revision hin zu einem dynamischen Plan auf der Grundlage von CA-Ergebnissen; Reduzierung der internen Kosten bei gleichzeitiger Steigerung der Effektivität durch IT-Lösungen.) Diese Klarstellung ist wesentlich deutlicher.

Die Themen CA und CM sind ein weiterer Baustein zur Professionalisierung der Bereichsaufgabenerfüllung. Die Tools für solche Analysen sind heute leistungsfähiger als noch vor wenigen Jahren, bedingen jedoch auch erhöhte Anforderungen an deren Beherrschung. Dies führt als Nebenaspekt zu einer nachvollziehbaren Spezialisierung in Richtung Data Analytics auch in anderen Unternehmensbereichen als der IT und einer Ausdifferenzierung (Job-Enrichment) analytischer Aufgabenwahrnehmungen innerhalb der Bereiche (Data Analyst, Data Scientist).

Zur Unterstützung der Aufgabenerfüllung gibt es am Markt eine Vielzahl an spezialisierten Softwareprodukten bezüglich Daten- und Prozessanalysen, die sowohl in der prüfenden als auch operativen Fachbereichsebene zur Stärkung des IKS eingesetzt werden können, z.B. CheckAud, Celonis (Process Mining) usw. Seeber (2019) hat zuletzt für die Revision mit „ActiveData für Excel“ eine günstige und gelungene Alternative für die Datenanalyse vorgestellt. „Die beabsichtigte Sammlung von Erfahrungswissen und Nutzung erwünschter Effizienzvorteile wird jedoch nur gelingen, wenn die hierfür zu nutzende Prüfsoftware auch weniger IT-affinen Revisoren eine intuitive Anwendung ohne umfassende Schulung ermöglicht, Software-Funktionen speziell auf Prüferfordernisse ausgerichtet sind und sich die Lizenzgebühren auch bei einfachen Anwendungsszenarien in einem überschaubaren Rahmen bewegen.“ [Seeber (2019), S. 254] Dies lässt sich auch auf andere Bereiche und ihre oben angeführte Anforderungsäußerung und -erfüllung übertragen.

Für manipulationsfreie Auswertungen weitgehend zahlenorientierter Datenhaltungen mit generell artähnlichen Daten kann – trotz eines ggf. großen Datenaufkommens in den Primärsystemen – **ergänzend** zu spezifischen und häufig auch teuren Toolsets wie IDEA, ACL, QLIK und Ähnliche auch ein „Make“-Ansatz sinnvoll sein, sofern folgende Rahmenbedingungen gelten:

- Im Fokus stehen vornehmlich strukturierte Geschäfts(abschluss)- und zugehörige Protokoll-/ Log-Daten (getätigte Transaktionen, Storno, Änderungen etc.)
- das DBMS ist Oracle oder Ähnliches
- insoweit steigt die Anzahl der Datensätze innerhalb eines größeren Zeitraums über ein kritisches Maß, sodass Office-Analytics allein an Performance-Grenzen stößt
- Alternative „Make“-Wege sind möglich, spezifisches Know-how vorhanden oder darf aufgebaut werden
- interne Verrechnungen / Kosten sind zweitrangig.

Sofern diese Bedingungen akzeptiert werden, offeriert ein **begrenzter „Make“-Ansatz** alternative Wege. Der folgende Artikel erläutert spezifische Aspekte einer solchen Möglichkeit.

Datenquellen

Für MS Access (syn.) und ähnliche Funktionalität gilt sicherlich, dass es unbestreitbar in den Unternehmen für kleine und mittlere Anwendungen in häufigem Einsatz ist. Programmierung, VBA, Prüfung auf referenzielle Integrität, Formularerstellung, Makros usw., alles Themen, die wir kennen. Prozesskritische / -treibende Datenverarbeitungen werden allerdings selten hierüber abgedeckt. Meist geht es eher um eine nachgelagerte Verarbeitung oder Datenveredelung, da es ansonsten bei Prozessführerschaft hinsichtlich Compliance, Schnittstellen, Security etc. zu Konflikten kommen kann. Insoweit findet sich als Datenhaltung denn auch zumeist Oracle im Hintergrund, wenn es um professionelle Softwaresysteme geht.

Dies kann ein Ansatzpunkt sein, um in der Linie eigenständig Datenanalysen durchzuführen. Entsprechend relevante Daten können identifiziert und als Regel in einem eigenen CA- / CM-Schema – täglich getriggert per Stored Procedure [Bedeutung z.B. auch Meyer (2013), S. 12ff.] – als maßgeschneiderte Kopie abgespeichert werden. Auf diese Daten wird dann aus **eigens programmierten Tools**, die bedarfsgerecht entwickelt werden, per ODBC zugegriffen.

Datenzugriff

Voraussetzung für ein integratives Analyse-Frontend

Sofern in einem Analyse-Frontend ein Link auf die Tabellen im Schema per ODBC gelegt wird, können die Tabellen mit komplexen SQL-Statements bedarfsgerecht ausgewertet werden. Dies entspricht einem SQL Pass-Through; im einfachsten Fall also beispielsweise Oracle mit Zugriff aus MS Access: Externe Daten => ODBC-Datenbank => Erstellen einer Verknüpfung zur Datenquelle => Datenquelle auswählen => Oracle ODBC Driver Connect. SQL, das inzwischen einen mächtigen Funktionsumfang aufweist [vgl. z.B. JOOQ (2016)], wird serverseitig ausgeführt (Server-Vorselektionen möglich), um die notwendige Performance zu gewährleisten, und clientseitig in der Präsentationsfunktionalität lediglich zur Anzeige gebracht. Bedeutsam sind dabei in den Skripten liegende, zweckmäßige temporale Eingrenzungen, damit sie in einer ansprechenden Ausführungszeit in MS Access ablaufen, da es ansonsten hier in der Präsentationsschicht zu fehlenden Fensterrückmeldungen und Datenanzeigeüberläufen (fehlende oder fehlerhafte Datenanzeigen) kommen kann.

Bei komplexeren Massendatenauswertungen, insbesondere bei Betrachtungen über längere Zeiträume, **kommt Office Analytics / MS Access jedoch grundsätzlich schnell an seine Grenzen**.

Zugriff über externen ODBC-Provider

Tatsächlich ergibt sich alternativ über die Nutzung einer **IDE** (Integrated Development Environment) **und Programmierung wie in Lazarus** [vgl. z.B. Lazarus (2016); TODBCConnection; analog auch Embarcadero (2009), S. 1478f. mit TADOConnection; ADO = ActiveX Data Objects] **die direkte Anbindungsmöglichkeit eines ODBC-Providers** (in Lazarus ist TOracleConnection ebenfalls geeignet, benötigt jedoch die oci.dll, diese ist nicht auf jedem Client vorhanden). Eine Oracle-Anbindung erfolgt beispielsweise über

```
Provider=MSDASQL.1;Mode=Read;Extended Properties="DSN=<DNS-Name>;UID=<USER-ID>;PWD=<Password>";
```

Dabei kann der Zugriff eine Geschwindigkeitssteigerung in der Präsentationsebene um weit mehr als den Faktor 1.000 gegenüber eines MS Access-Zugriffs bedeuten. Die Kennwortangabe kann dabei im Programm maskiert werden.

Schema-Name

Es ist darauf zu achten, dass die Datenquellangaben (Tabellen) in den SQL-Abfragen, die in MS Access bei ODBC-Linkherstellung den Aufbau <Schema-Name_Tabellenname> aufweisen, bei einem Oracle-Direktzugriff per ODBC keine Angabe des Schema-Namens wie unter MS Access aufweisen dürfen, es ist nur die Tabellennamensangabe notwendig. Bei fehlerhaftem Statement erfolgt eine SQL_ERROR-Fehlermeldung.

```
Could not execute statement. ODBC error details:
LastReturnCode: SQL_ERROR; Record 1: SqlState: 42000;
NativeError: 972; Message:
[Oracle][ODBC][Ora]ORA-00972: Bezeichner ist zu lang
```

Abb. 1: Error-Message aus SQL bei fehlerhafter Datenquellenangabe.

Die Angabe eines Schema-Namens bei Tabellen ist immer nur dann erforderlich, wenn aus einem anderen Teil (also aus einem anderen Schema) der Datenbank oder von extern auf Objekte des Schemas zugegriffen werden soll. Bei ODBC-Linkherstellung kann in MS Access der Schema-Name entfallen, da er in der Linkdefinition enthalten ist. Dies ist allerdings manuell zu ändern.

Verschiedene Connections

Die Nutzung des Direktzugriffs auf Oracle sollte jedoch flexibel ausgewählt werden können, da es auch Auswertungen geben kann, die eine Vorselektion (z.B. in MS Access) nutzen müssen. Dies ist über Oracle nicht zu gewährleisten, wenn keine eigenen Auswertungen auf der Oracle-Ebene implementiert werden dürfen. Insoweit sollte zu jedem SQL-Statement, das ausgeführt werden soll, flexibel auszuwählen sein, mit welcher Connection (MS Access oder Oracle) gestartet wird. Des Weiteren ist darauf zu achten, dass insbesondere beim Nutzen von Sortierungsaufrufen sowie Datums- und Textfiltern die Konvertierungsfunktionsaufrufe und Formatvorgaben in PL/SQL und MS Access differieren. Insoweit können ggf. auch aufrufspezifische Auswertungen vorliegen.

Besonderheit Fetching

Es gilt ebenso zu verhindern, dass beim Empfang von Ergebnisdatensätzen vom Provider aus einer SQL-Abfrage heraus in ein anzeigendes Daten-Grid das Fenster wartet, bis die Menge in Gänze aufgenommen wurde, da in dem Fall die Performance einbricht. Es ist sinnvoller, dass die Daten nach und nach in Blöcken, also asynchron, in das Daten-Grid geladen werden, sodass das Fenster-Handle trotz nachladender Datensätze wieder für den Zugriff frei wird und die Datensätze bei ansprechender Geschwindigkeit und ohne Blockieren des Fensters zur Verfügung stehen. In Lazarus ist dies über die kostenfreie Universal Data Access-Komponentenbibliothek UniDAC über die spezifische Objekteigenschaft *FetchAll* in TUniQuery (z.B. Setzen von UniQueryAuswertung.SpecificOptions.Values['FetchAll']:='True';) bzw. erweitert über die kostenpflichtige Oracle Data Access-Bibliothek ODAC von Devart über die Eigenschaften *FetchAll* und *NonBlocking* in TOraQuery sowie TOraTable möglich ([auch TOraStoredProc]; vgl. z.B. Kosch (2001), CREFIRD (2011), hinsichtlich des asynchronen Fetchings bei ADO-Zugriff, *eoAsyncFetchNonBlocking*).

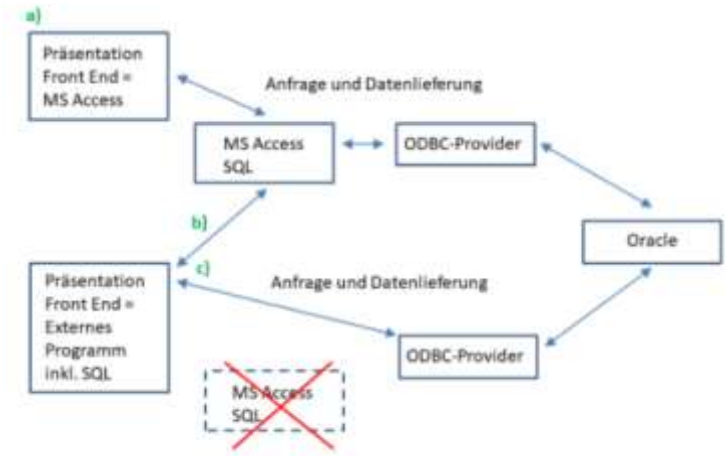

Abb. 2: Aufrufwege des Datenlieferungsprozesses (vereinfacht).

Code-Beispiele

Um einen Einblick zu erhalten, wie ein erstelltes Programm an einigen Stellen arbeitet, können beispielhaft Code-Snippets angeführt werden. Das erste Beispiel geht von dem Umstand aus, dass bei einem Zugang auf eine Oracle-Datenbank Daten per Query in Blöcken eingelesen werden. Sofern dieses Einlesen noch nicht beendet, die Datenquelle also nicht bei End-of-File angelangt ist (Fetching), sollen die Elemente des Formulars auch noch nicht wieder freigegeben werden, da ansonsten ggf. eine Aktivität durch den Benutzer angestoßen wird, obwohl noch Daten gelesen

und dargestellt werden. Dies lässt sich mit einem Timer steuern. Er wird gestartet, wenn das SQL-Statement ausgeführt wird (TimerDatenquelle.Enabled:=true;). Bis er inaktiv gesetzt wird, triggert er jede halbe Sekunde die Prüfung, ob die Datenquelle noch Daten erhält. Sofern sie wie in unserem Beispiel nicht auf *dsBrowse* steht (Default-Wert der Datenquelle), ist die Quelle noch nicht zur Ruhe gekommen, der Vorgang ist nicht beendet [vgl. Lazarus (2017)]. Wird die Datenquelle für das Fetching verwendet, erhält sie üblicherweise während des Zugriffs den Status *dsOpening* [vgl. Embarcadero (2015)], für andere Aktivitäten einen anderen Status ungleich *dsBrowse*. Danach wird der Timer bis zum nächsten Anstoß einer Auswertung wieder deaktiviert.

```
procedure …TimerDatenquelleTimer(Sender: TObject); // aktiv nur beim Fetching
begin
   if (ora = true) then // Datenquelle ist eine Oracle-DB, wird zuvor festgestellt
      begin
       if (adsQueryAuswertung.State <> dsBrowse) then
        begin
         if (adsQueryAuswertung.RecordCount > 0) then // Datenquelle beinhaltet Daten
          begin
           // Warten, bis die Datenquelle keine Daten mehr erhält
              SperrungElementeAuswertung(true); // Prozedur, die ausgesuchte Elemente sperrt (Timer läuft)
          end
          else
          begin
           // Freigabe aller Elemente nach Ende adsQueryAuswertung
              SperrungElementeAuswertung(false); // Elementsperrung wird aufgehoben
              TimerDatenquelle.Enabled:=false; // Timer beenden
          end;
        end
        else
        begin
           SperrungElementeAuswertung(false); // Elementsperrung wird aufgehoben
           TimerDatenquelle.Enabled:=false; // wenn State = dsBrowse (Grundeinstellung) = Timer beenden
        end;
      end
      else
      begin
       TimerDatenquelle.Enabled:=false; // DB nicht Oracle? Dann Timer beenden
      end;
end;
```

Analog ODAC:

```
// Mit der Oracle-Session und Oracle-Query wird auch der TimerDatenquelle aktiviert
OraSessionPrimary.Connected:=true;
OraQueryAuswertung.Active:=true;
TimerDatenquelle.Enabled:=true;
…
procedure …TimerDatenquelleTimer(Sender: TObject);
begin
// Datenquelle ist explizit eine Oracle-DB
  if (OraDataSourceAuswertung.State <> dsBrowse) then
   begin
    // Warten, bis die Datenquelle keine Daten mehr erhält
    SperrungElementeAuswertung(true); // Prozedur, die ausgesuchte Elemente sperrt (Timer läuft)
   end
   else
   begin
    SperrungElementeAuswertung(false); // Elementsperrung wird aufgehoben
    TimerDatenquelle.Enabled:=false; // wenn State = dsBrowse (Grundeinstellung) = Timer beenden
   end;
end;
```

Ein weiteres Beispiel geht davon aus, dass in einem Daten-Grid, das Daten aktuell beinhaltet und anzeigt, ein Feldwert gewählt wird. Dieser Wert soll gemeinsam mit der zugehörigen Spaltenüberschrift genutzt werden, um einen Filter zu setzen.

```
procedure …TasteF10Execute(Sender: TObject); // Filter setzen per Taste F10
…
nummer_ds:= adsQueryAuswertung.RecNo; // Nummer des Datensatzes
anzfelderado:= adsQueryAuswertung.IndexFieldCount; // Anzahl Felder aktueller Index
begrenzerS:='';
begrenzerE:='';
…
  if ((nummer_ds > 0) and (anzfelderado > 0)) then
    begin
     if (ora = true) then // Unterscheidung, ob z.B. Oracle oder MS Access
        begin
          begrenzerS:='['; // Feldname wird bei Oracle in eckige Klammern gesetzt
          begrenzerE:=']';
        end;
…
 spaltenueberschrift:=DBGridErgebnisanzeige.SelectedField.FullName; // Feldname
 feldwert:=QuotedStr(DBGridErgebnisanzeige.SelectedField.AsString);
…
filterauswahl:=begrenzerS+spaltenueberschrift+begrenzerE+' = '+feldwert; // Filterwert zusammensetzen
…
adsQueryDetails.Filter:=filterauswahl; // Filter zuweisen und aktiv setzen
adsQueryDetails.Filtered:=true;
…
```

Ein letztes Beispiel soll zeigen, dass, obwohl Tabellenspalten typabhängig unterschiedlich lang angezeigt werden, alle Spaltenbreiten einen möglichst einheitlichen Wert aufweisen, damit beim Ausführen unterschiedlicher Auswertungen mit differenten Feldern die Sichtgewohnheit des Nutzers, das „Look & Feel“, nicht gebrochen und die Darstellung somit einheitlich ist.

```
procedure …WidthSetzenDBGrid(dbg: TDBGrid; ads: TADODataSet);
var
 i: integer;
 d: integer;
begin
 d:=0;
 i:=ads.Fields.Count; // Anzahl vorhandener Felder in der Grid-Darstellung
 if (i > 0) then
  begin
   for d:=0 to i -1 do
    begin
     dbg.Columns[d].Width:=100; // alle Felder werden auf die Spaltenbreite 100 gesetzt
    end;
  end;
end;
…
WidthSetzenDBGrid(DBGridErgebnisanzeige, adsQueryAuswertung); // Nach ads-Aktivierung und
// vor Grid-Refresh
…
```

Grundsätzlich ist die Umsetzung solcher Programmabschnitte für den Zugriff auf MS Access unkompliziert, allerdings muss die Umsetzung für Oracle und Ähnliche eindeutig im Quellcode unterschieden werden, da sie sich zu MS Access in einer Vielzahl an Punkten unterscheidet.

Anforderungen an die Softwarequalität (inklusive Template-Herstellung, Dokumentationsvorgaben und Testverfahren) müssen dabei natürlich ebenfalls erfüllt werden [vgl. z.B. Taentzer (2015)].

Mögliche Programmentwicklung

Die Entwicklung solch spezifischer Datenanalysesoftwareprodukte zur Unterstützung der Aufgabenwahrnehmung ist insoweit interessant, weil der fachbereichsinterne Entwickler auf Besonderheiten sowohl der Datenbereitstellung als auch der Logik als geäußerte Prämisse Rücksicht nehmen kann. Dies kann dazu führen, dass innerhalb eines solchen Programms ergänzende Verarbeitungsschritte - auch als Schnittstelle - bearbeitet werden können, die in den gekauften Standardsoftwareprodukten nicht enthalten sind. Das Ergebnis kann beispielsweise ein SQL-basierter Datenextraktor sein, der die zugrundeliegenden Daten bedarfsgerecht aus den angebundenen Datenbanken sowohl selektiert als auch aufbereitet und dabei flexibel in Retrieval und Präsentation jederzeit angepasst werden kann. Voraussetzung ist, dass fachliche Ressourcen in der Linie vorhanden sind und genutzt werden dürfen.

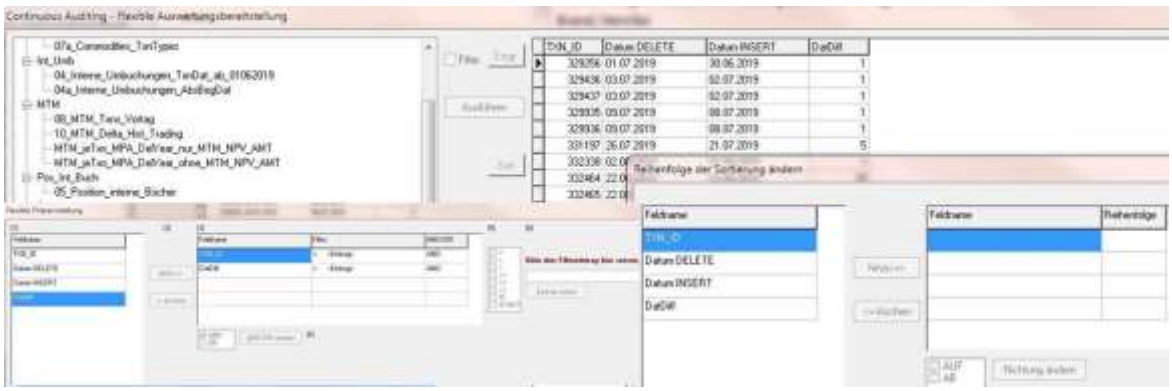

Abb. 3: Beispiel für eine flexible Sortier- und Filterfunktion in einem in Lazarus entwickelten Auswertungstool.

Fazit

Es ist sinnvoll, neben SAP auch weitere (rechnungslegungsrelevante) IT-Systeme datenanalytisch betrachten zu können. Während es Prüfansätze, -tools und auch Daten- und Schnittstellenbeschreibungen für die ERP-Landschaft gibt (z.B. Importhilfen für SAP in IDEA Tax Audit), existieren für „exotische“ Systeme kaum Ansätze. Aber auch hier müssen die Unternehmensbereiche Wege finden, sei es durch zugekaufte Analytics-Funktionalität oder durch Eigenentwicklung, systematisch ergebnisoffene Analyseansätze zu etablieren.

Werden wie oben beschrieben Schema-Aufbauten in den originären Systemen vorgenommen (Oracle), die regelmäßig per Stored Procedure – getriggert mit Echtdaten versorgt – gefüllt werden, können diese, ohne die Primärsysteme zu belasten, ausgewertet werden. Darauf aufbauend ist es z.B. möglich, Auswertungstools zu schaffen, die in nach Belieben involvierten Datenbanken weitgehend **beliebige Statements** – thematisch angeordnet und mit Varianten gespeichert – unter Wahrung der Sortier- und Filterbarkeit der Datenbasis bzw. der identifizierten Datensätze flexibel read-only prozessieren, sodass dies zu einer Systematisierung der Analysen verschiedener Fragestellungen beiträgt – entsprechend für alle sinnvoll anzubindenden Datenbanken als generischer Ansatz. Das Nachfassen zu jeweiligen Ergebnisdaten – nach Drill-Down auf die geschäftsbezogenen Datensätze – kann wiederum nicht durch Automatismen substituiert werden, sofern das originäre System nicht über eine Schnittstelle angebunden ist. Ergebnisdatensätze liefern dann aber wichtige Hinweise für genaue Recherchen im Primärsystem. Die Einbindung solch externer Datenschnittstellen und -verarbeitungen, z.B. per DLL-Projektimplementierung, ist allerdings ebenso möglich [vgl. Wildensee (2020)].

Der beschriebene Ansatz beinhaltet das Vorgehen für einen vergangenheitsbezogenen und keinen Predictive oder Simulationsansatz. Das Datenschutzrecht ist dabei selbstverständlich zu beachten [vgl. z.B. DIIR (2017), S. 7ff., hinsichtlich Zweckbestimmung und Verhältnismäßigkeit].

Lazarus ist ein Beispiel für eine **kostenfreie**, **moderne** und **plattformoffene** (cross-platform) Entwicklungsumgebung (Rapid Application Development; kostenpflichtige Komponenten sind nicht zwingend erforderlich; analog kostenpflichtig Delphi RAD Studio mit ADO), mit der auch im Unternehmen fachlich wie lizenzrechtlich unproblematisch gearbeitet werden kann. Es ist leicht zu erlernen und zu bedienen, produziert leistungsfähige Kompilate, und diese sind nicht von ausgerollten DLLs oder Frontend- / Backend-Produktversionen abhängig, sondern laufen dauerhaft stabil. So ist Lazarus ideal für die Read-Only-Datenbankanbindungsprogrammierung analytisch arbeitender Bereiche außerhalb der IT.

Zwar besteht die Gefahr des dezentralen Entwicklungswildwuchses, der schon in der Vergangenheit häufig beklagt wurde, doch durch klare Vorgehensfestlegungen / -richtlinien, einer ausschließlichen Entwicklung im Analytics-Umfeld und Ausschluss der Übernahme der Prozessführerschaft einer solchen Softwareentwicklung können ggf. doch mehr Vor- als Nachteile für die operative Ebene konstatiert werden. Dies sollte zumindest einbezogen werden, wenn es um die Frage der Funktionalitätserweiterung zur Unterstützung der Aufgabenwahrnehmung im Fachbereich geht.

Es wird m.E. zukünftig an Bedeutung gewinnen, sich mit grundlegenden Programmierkonzepten und den beschriebenen Grundlagen zu beschäftigen, auch um mit Entwicklungsbereichen ansatzweise auf Augenhöhe sprechen zu können. Vor dem Hintergrund zunehmender agiler Entwicklung zur Kundenansprache, Produktplatzierung usw. entstehen neue Umgebungen in den Unternehmen. Schnelle Anpassbarkeit in der Softwarelogik ist elementar. Beschäftige als Key-User-Teams zu entwickeln, die entsprechende Skills aufbauen, ist zeitgemäß und bewährt, wenn auch nicht kostenlos zu erhalten. Dieser Beitrag ist entsprechend ein Plädoyer für die mutige Nutzung der eigenen Innovationskraft und Sprengung vorgegebener Limitierungen durch zugekaufte Softwaresysteme, sodass sich „Make“ und „Buy“ ergänzen.

(*Die Ausführungen stellen ausdrücklich die Ansichten des Autors und nicht die Haltung des Unternehmens dar*.)

Literatur

CREFIRD (2011): DBGrid mit Read-Ahead-Funktion unter Verwendung von ADO, https://bestecode.com/question/8547299-dbgrid-mit-read-ahead-funktion-unter-verwendung-vo .

Deloitte (2010): Continuous monitoring and continuous auditing - From idea to implementation, https://www2.deloitte.com/content/dam/Deloitte/uy/Documents/audit/Monitoreo%20continuo%20y%20auditoria%20continua.pdf .

DIIR (2017): Leitfaden Interne Revision und Datenschutz, https://www.diir.de/fileadmin/arbeitskreise/Leitfaden_Interne_Revision_und_Datenschutz.pdf .

Embarcadero (2009): RAD Studio, http://docs.embarcadero.com/products/rad_studio/delphiAndcpp2009/HelpUpdate2/EN/pdf/devwin32.pdf .

Embarcadero (2015): Determining DataSet States, http://docwiki.embarcadero.com/RADStudio/Rio/en/Determining_Dataset_States .

Gehrig-Ehrenzeller (2011): Die Interne Revision als Führungsinstrument des Verwaltungsrates und des Prüfungsausschusses, Dissertation Nr. 3892 aus 2011, Universität St. Gallen, https://www1.unisg.ch/www/edis.nsf/SysLkpByIdentifier/3892/$FILE/dis3892.pdf .

JOOQ (2016): 10 SQL Tricks that you didn't think were possible, https://blog.jooq.org/2016/04/25/10-sql-tricks-that-you-didnt-think-were-possible/ .

https://jaxenter.com/10-sql-tricks-that-you-didnt-think-were-possible-125934.html .

Kosch (2001): Datensätze paketweise in DBGrid laden, https://entwickler-forum.de/forum/archiv/andere-sprachen-aa/delphi-aa/datenbankentwicklung-aa/18852-datensätze-paketweise-in-dbgrid-laden .

Lazarus (2016): ODBCConn, https://wiki.freepascal.org/ODBCConn .

Aktuelle IDE: https://www.lazarus-ide.org/ .

Lazarus (2017): TDataSetState, https://www.freepascal.org/docs-html/current/fcl/db/tdatasetstate.html .

Meyer (2013): Visuelle Analyse von Stored Procedures in Datenbanksystemen, Diplomarbeit 2013, Universität Stuttgart, https://elib.uni-stuttgart.de/bitstream/11682/3241/1/DIP_3487.pdf .

Preuss (2012): Überwachung - Continuous Monitoring - Continuous Auditing, Automatisierung im Rahmen des Jahresabschlussprüfung, ISACA/SVIR Fachtagung, Zürich, 5.11.2012, http://www.isaca.ch/images/downloads/downloads/svir/IA_SVIR_ISACA_12.11.05/5_Preuss.pdf .

Seeber (2019): ActiveData für Excel - die innovative Prüfsoftware-Alternative, PRev 5/2019, S. 254-258.

Skantze (2017): Continuous Auditing – Internal Audit at a Crossroads, Master-Thesis 2017, Stockholm Business School, https://pdfs.semanticscholar.org/d0bc/ee1437ba2ce2dd66cf9f99d46add231599f7.pdf .

Taentzer (2015): Softwarequalität, Universität Marburg, 20.01.2015, https://www.uni-marburg.de/fb12/arbeitsgruppen/swt/lehre/files/est1415/EST150120.pdf .

Wildensee (2020): Lazarus mit Aufruf von SAP-Connect, http://www.wildensee.de/lazsapconn.zip .

Ausgabe 1 Februar 2020

Revisionspraxis

Journal für Revision, IT-Sicherheit, SAP-Sicherheit und Datenschutz

PRev

Christoph Wildensee
Fachbereichsspezifischer Entwicklungsansatz zur bedarfsgerechten Analyse SQL-basierter Datenquellen

Nicole Adloff
Emotet – Erfolgsgeschichte einer Malware

Gerald Schrott
Protokollierungsmechanismen in SAP

Maximilian Schmidt
Rechtliche Zulässigkeit von Penetrationstests

Greindl/Mayer/Nauß
IT-Datenschutz Management System (IT-DMS) als Möglichkeit zur Umsetzung der Anforderungen des Art. 32 EU-DSGVO

Bettina Buczman
Das Auskunftsrecht gem. Art 15 DSGVO im Bereich des Beschäftigtendatenschutzes

Rechtsprechung und Aktuelles zum Datenschutz

www.prev.de

ISSN 1862-9032

BOORBERG