einfach. digital. agil.

PRÄSENTIERT

Verlagerung von Programm-Logik in die Datenbank

Birgitta Hauser
Diplom-Betriebswirt (BA)
Software and Database Architect
Hauser@SSS-Software.de

Agenda

Anwendungszentriertes versus Datenzentriertes Denken

Auslagerung von Datei-Zugriffen

Views

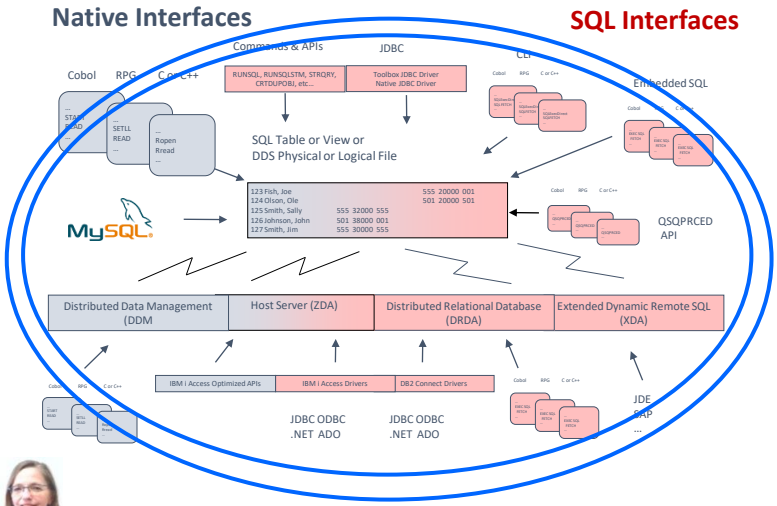
Gewährleistung der Daten-Konsistenz

- Check Constraints
- Referentielle Integritäten
- Trigger-Programme
- Commitment Control

Row and Column Access Control – Datenzugriffsbeschränkung

Anwendungszentriertes Denken versus Datenzentriertes Denken

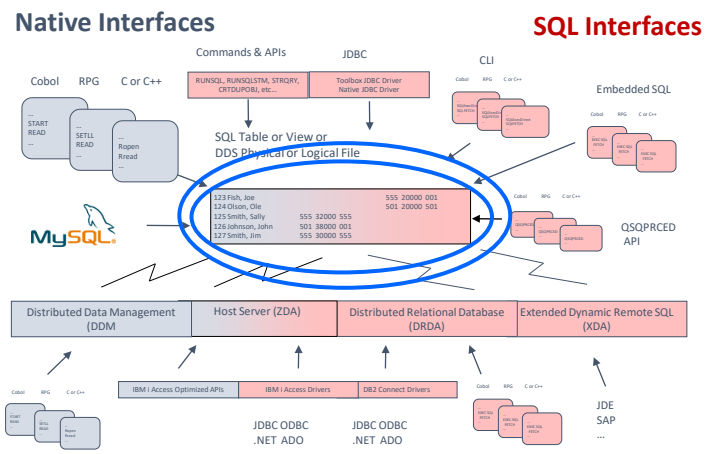
Anwendungszentriertes Denken (Application-Centric)



- Aufwändig in der Wartung
  - Logik im Programm-Code
  - Gleiche Funktionalität in unterschiedlichen Programmier-Sprachen auscodiert
- Daten-Sicherheit nicht gewährleistet!

Quelle: Db2 Ping Pong - POW3R 2018
Scott Forstie (IBM) / Birgitta Hauser

Datenzentriertes Denken (Application-Centric)



- Verlagerung von Programm-Logik in die Datenbank
  - Aktiviert/erzwungen durch die Datenbank/Datanbanken-Manager
  - Gleiche Funktionalität in allen Programmier-Sprachen
  - Reduzierung des Quell-Codes auf ein Minimum
- Daten-Sicherheit gewährleistet

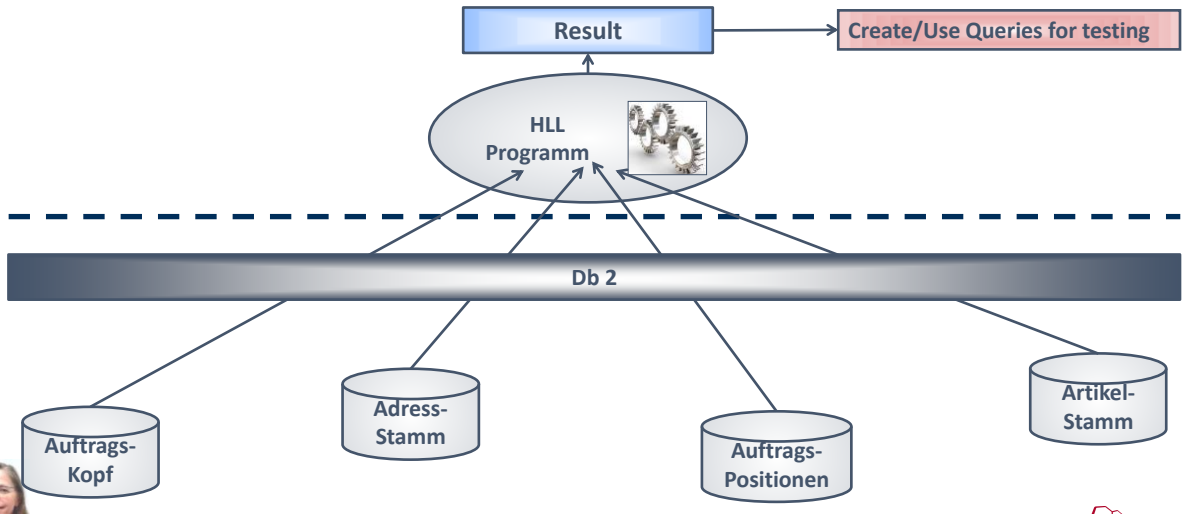
Quelle: Db2 Ping Pong - POW3R 2018
Scott Forstie (IBM) / Birgitta Hauser

POW3R einfach digital agil

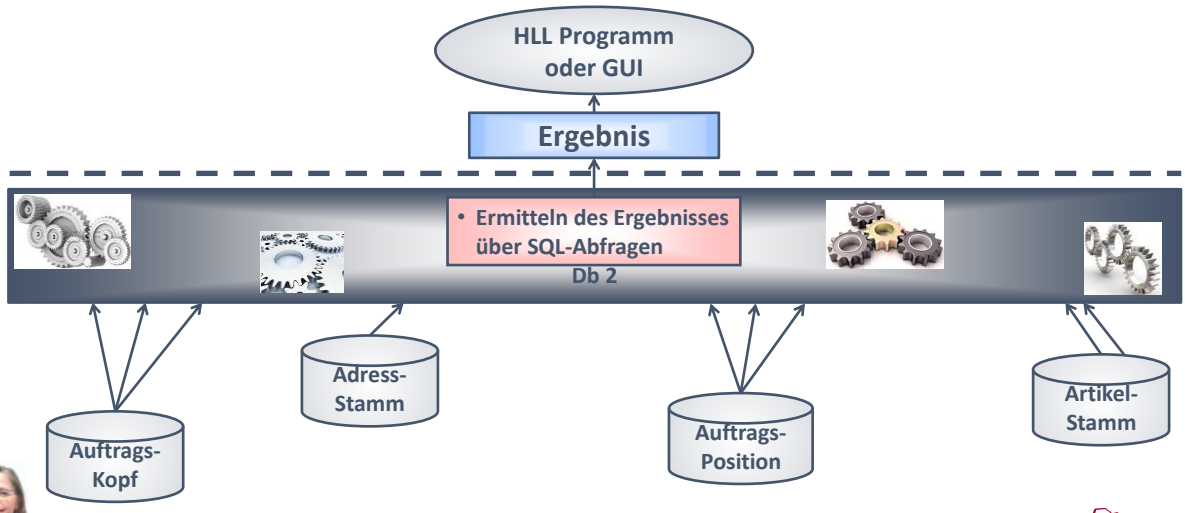
Record Level Access (RLA) versus SQL-Zugriff

POW3R einfach digital agil

Traditionelle Zugriffsmethode: Record Level Access



SQL – Datenorientierte Programmierung



Auslagerung von Datei-Zugriffen

Datei-Zugriffe auslagern – Insert / Update / Delete

Standard-Prozeduren in Service-Programmen für Insert/Update/Delete und Einzelsatz-Zugriffe (Chain)

- Parameter für Insert/Update: Datenstruktur für **Basis-View**
- Rückgabe-Wert für Insert: Automatisch generierte **eindeutige Id** (z.B. Identity Column)
- Parameter für Delete/Chain: **Eindeutiger Schlüssel** und/oder **Relative Satz-Nr.**
- Rückgabe-Wert Chain: **Indikator** (Satz vorhanden oder nicht)
- Ausgabe-Parameter Chain: (Initialisierte) Datenstruktur für **Basis-View**
- **Weitere** Parameter: Behandlung Satz-Sperre, Ausführung unter Commitment Control, Senden Abbruchnachricht im Fehlerfall, Update auf alle Fälle durchführen

→ **Basis-Quell-Code kann mit Hilfe von Templates automatisch generiert werden.**

Weitere Prozeduren (müssen manuell vervollständigt werden)

- Delete: Prüf-Prozedur, ob ein Datensatz gelöscht werden darf oder nicht
- Insert/Update: Prüf-Prozedur für Eingabe-Werte – Initialisierung mit Default-Werten

Aufruf von weiteren Procedures (z.B. für cascadierendes Löschen)

Alle Insert-/Update-Delete Operationen dürfen nur noch über die Standard-Prozeduren erfolgen.

Datei-Zugriffe auslagern – Data Centric

Erstellen von Views zur Maskierung der Koplexität

- Erstellen einer **View** für **jede Aufgabe** und/oder **(verschachtelte) Standard-Views**

Service-Programm pro View

- Enthält **alle Prozeduren**, in denen auf die **View zugegriffen** wird
- Wird die gleiche Schleife mehrfach verwendet, sollte **Call Back Processing** verwendet werden.

  Verarbeitungs-Prozedur wird als Procedure Pointer übergeben

Views

View - ungeschlüsselte logische Datei

Erstellung über SQL-Befehl **CREATE VIEW**

- Beschreibung des Datenzugriffs durch **SELECT-Statement**
- Enthält **keine Daten**
- Äquivalent zu einer **ungeschlüsselten logischen Datei**

View – Ungeschlüsselte logische Datei

Erlaubt alles, was in einem SELECT-Statement möglich ist mit einer Ausnahme: ORDER BY

- **Spalten-Auswahl** und Erstellung zusätzlicher Spalten
- Alle **JOIN-Anweisungen** (Inner Join, Left/Right/Full Outer Join, Left/Right Exception Join, Cross Join)
- **WHERE**-Bedingungen
- **GROUP BY** (inkl. Multi-dimensionalem Grouping) und HAVING-Anweisungen
- **Skalare Funktions** (SQL ca. 180 scalare Funktionen) / User Defined (Table) Functions
- User Defined (Table) Functions zur Erstellung und Verarbeitung von **XML und JSON Daten**
- **CASE**-Anweisungen
- **UNION** / EXCEPT / INTERSECT
- **Common Table Expressions** (inkl. rekursiver CTEs)
- Verschachtelte Sub-Selects
- Verwendung von **globalen Variablen**

Verwendung von Views in anderen Views → Verschachtelte Views

Ungeschlüsselt → Zugriffspfad-Wartung *REBLD

- Tausende Views auf gleicher Tabelle **ohne Performance-Einbußen**

View – Ungeschlüsselte logische Datei - Beispiele

View: Quartals-Umsätze

```
Create View HSCOMMON10.SalesQuart as
Select   CustNo,
         Year(SalesDate) as SalesYear,
         Cast(sum(case when Quarter(SalesDate)= 1 then Amount else 0 end) as Dec(11, 2)) as Q1,
         Cast(sum(case when Quarter(SalesDate)= 2 then Amount else 0 end) as Dec(11, 2)) as Q2,
         Cast(sum(case when Quarter(SalesDate)= 3 then Amount else 0 end) as Dec(11, 2)) as Q3,
         Cast(sum(case when Quarter(SalesDate)= 4 then Amount else 0 end) as Dec(11, 2)) as Q4,
         Cast(sum(Amount) as Dec(13, 2)) as Total
   from  Sales
   group by CustNo, Year(SalesDate);;
```

View: Verknüpfung der Quartals-Umsätze mit Adress-Stamm

```
Create View HSCOMMON10.SalesQCust
    as Select a.CustNo as ACustNo, SalesYear, Q1, Q2, Q3, Q4, Total,
              b.*
         from SalesQuart a Left outer join AddressX b on a.CustNo = b.CustNo;;
```

Zugriff auf View SalesQCust

```
Select SalesYear, ACustNo, CustName1, City, Q1, Q2, Q3, Q4, Total
from SalesQCust
Where SalesYear = 2009 and ACustNo in ('10003', '10005')
```

SALESYEAR	ACUSTNO	CUSTNAME1	CITY	Q1	Q2	Q3	Q4	TOTAL
2009	10003	... Goldbach GmbH	... Alzenau...	733,00	1057,91	2227,45	571,50	4589,86
2009	10005	... Alzenauer Dönertreff	... Alzenau...	1475,00	285,75	1587,50	393,70	3741,95

View - ungeschlüsselte logische Datei

Verwendung? Wann und warum?

- **Umbenennung** von Spalten oder Generierung **neuer Spalten**
- **Datenkonvertierung** z.B. numerische Datums-Felder
- **Zugriffskontrolle**
  - View **ohne Spalten** mit sensiblen Spalten
  - Einschränkung der Zeilen durch **WHERE-Bedingungen**
  - Spezielle **Zugriffsberechtigungen** für des **View-Objekt**
- **Verlagerung von Programm-Logik in die Datenbank**
  - **Verwendung Views** überall, wo eine Tabelle verwendet werden kann
  - **Maskieren** von komplexen Abhängigkeiten
  - **Minimierung von Quellcode** unabhängig von der Programmiersprache
  - **Mehrfach-/Wiederverwendung** in beliebigen Tools, Programmiersprachen etc.
- **Programme/Prozeduren** von **Änderungen** in **Datenbank nicht betroffen**
  - Änderung/Anpassung/Umwandlung nicht erforderlich

View – Modernisierung → Was ist zu tun und wo anfangen?

Erstellen Basis-View: alle Spalten in gleicher Reihenfolge wie physische Datei

- **Zugriff** erfolgt **nicht mehr** auf **physische Datei/Tabelle,** sondern auf diese **View**
  → Einsatz **Instead Of Triggers** erlaubt ein zukünftiges Datenbanken-Redesign

Alle weiteren Views für die physische Datei/Tabelle basieren auf Basis-View

- Joins mit anderen (Basis-)Views → Beispiel: Join Auftragskopf / Adress-Stamm / Auftrags-Positionen / Artikel-Stamm

Erstellen einer View für jede zu erledigende Aufgabe

- Verlagerung von Programm-Logik in Datenbank
- Beispiel: Offene Posten, Kumulierte Bestände verknüpft mit Artikel-Stamm
- Bei Änderung/Erweiterung müssen lediglich Views angepasst werden

Programmierer und Query Anwender dürfen nur noch auf Views zugreifen

- Verwendung in embedded SQL, ODBC/JDBC, Query/400, DB2 WebQuery
- Beim Zugriff auf Views sollte SELECT * vermieden werden
  - Rekompilierung der Programme ist nur erforderlich, bei Verwendung neuer Spalten
  - Bessere Performance: weniger Datenbanken-Zugriffe → Optimizer kann u.U. IOA (Index Only Access) verwenden

Datenintegrität

Datenintegrität

Gewährleistung von Daten-Integrität

- **Unique/Primary Key Constraints (Integrität)**
  - **Schlüssel-Wort UNIQUE** in physischen/logischen Dateien / SQL-Indices
  - Definition von **Unique Key Constraints** in SQL-Tabellen
  - CL-Befehl **ADDPFCST** oder SQL-Befehl **ALTER TABLE**
- **Check Constraints – Prüf Integritäten**
  - **Prüfungen auf Feld-Ebene**
    - Vorgabe einer Liste von Werten, Festlegung von Bereichen
    - Vergleich mit anderen Spalten/Feldern im gleichen Satz
  - CL-Befehl **ADDPFCST** oder SQL-Befehl **ALTER TABLE**
  - SQL-Abfragen mit SQE → **Constraint Awarness**

Datenintegrität

Gewährleistung von Daten-Integrität

- **Referential Integrities - Referentielle Integritäten**
  - **Abhängigkeiten zwischen Dateien**
  - **Prüfungen/Aktionen** beim **Hinzufügen/Löschen** Zeilen
  - CL-Befehl **ADDPFCST** oder SQL-Befehl **ALTER TABLE**
- **Trigger**
  - Programme aktiviert durch Database Manager
  - **Erweiterte Funktionalität** zu Check Constraints

POW3R Virtual - 09.-11.November 2020 - Verlagerung von Programm-Logik in die Datenbank– Birgitta Hauser

Referentielle Integritäten

Referentielle Integritäten

Definition und Prüfung von Abhängigkeiten zwischen Tabellen

- Beispiele: Keine Auftragsposition ohne zugehörigen Auftragskopf
  Artikelstamm kann nicht gelöscht werden,
  solange Bestand für den Artikel vorhanden ist.

Vorteile

- **Weniger Kodierung** Regeln sind **direkt** in der Datenbank hinterlegt
- **Bessere Performance** DBMS verarbeitet die Regeln schneller als individuell geschriebene Programme
- **Portabilität** Geschäftslogik ist **nicht** im Programm-Code versteckt
- **Sicherheit** Geschäftsregeln und Datenintegrität können **nicht umgangen** werden, weder zufällig noch mutwillig

Sorgfältige Planung erforderlich

- Abhängigkeiten zwischen den Tabellen **müssen bekannt sein**
- Ausführung von Inserts/Updates/Deletes in einer **festgelegten Reihenfolge**

Trigger

Was sind Trigger?

Programme verbunden mit physischen Dateien/Tabellen

- **Externe Trigger:** Programme in **High Level Language** (HLL) z.B. RPG / Cobol
  Registrierung über **CL-Befehl ADDPFTRG oder SQL-Befehl CREATE TRIGGER**
  Aktivierung **nur auf Satz / Zeilen-Ebene**
- **SQL-Trigger:** Erstellung mit **SQL-Befehl CREATE TRIGGER**
  Aktivierung auf **Satz-Ebene** - Update-Trigger können auch auf **Feld-Ebene** aktiviert werden
  Bedingte Trigger / Instead of Trigger (nur auf SQL Views)
  Aktivierung **pro Zeile** oder **pro SQL-Statement**

Aktivierung durch Datenbanken-Manager

- Abhängig von **Ereignis/Event:** Insert / Update / Delete / Read (nur System-Trigger)
- Abhängig von **Zeitpunkt/Time:** Before / After / Instead Of
- **Unabhängig** vom verwendeten Interface: Interaktiv, embedded SQL / ODBC / JDBC / Native I/O / UpdDta usw.

Bis zu 300 (System-) Trigger (extern+SQL) pro physische Datei/Tabelle

- **Aktivierung nach Erstellungs-Datum (LIFO)**

Einsatz von Triggern

Erzwingen von Geschäftsregeln
- Regeln, die über Check Constraints nicht abgedeckt werden können
- Beispiel: Nach Auslieferung kopieren Auftrags-Kopf und Auftrags-Positionen in History-Datei

Prüfen von Daten-Integritäten
- Beispiel: Sachbearbeiter erfasst Auftrag → Prüfen für Kunde zuständig

Datenkonsistenz über verschiedene Dateien/Tabellen
- Beispiel: Liefertermin in mehreren Dateien/Tabellen gespeichert
  bei Änderung → Änderung in allen Dateien/Tabellen

Integration neuer Technologien in vorherigen Anwendungen
- Beispiel: Senden einer automatischen eMail an Kunden
  als Empfangsbestätigung nach Auftrags-Übernahme

Commitment Control

Commitment Control

Zusammenfassung von Änderungen zu einer Transaktion

- Eine **Transaktion** umfasst **mehrere individuelle Änderungen** an Objekten, die als **eine einzige Aktion** behandelt werden
- **Beispiele** für Transaktionen:
  - Buchung: Zugang / Abgang
  - Auftrag: Keine Teillieferungen erlaubt

Commitment Control stellt sicher, dass

- **ALLE Änderungen** innerhalb einer Transaktion ausgeführt werden
- **KEINE Änderung** innerhalb einer Transaktion wird ausgeführt
- Bei einem **Abbruch** werden **ALLE bereits getätigten Aktionen zurückgesetzt**

Beginn/Ende einer Transaktion: `COMMIT`
Zurücksetzen innerhalb einer Transaktion: `ROLLBACK`

Commitment Control mit SQL

Voraussetzung: Tabellen im Journal aufgezeichnet

STRCMTCTL wird von SQL automatisch gestartet

- **Achtung:** Start mit Default-Werten → **CMTSCOPE – Aktivierungs-Gruppe**

Setzen Commitment-Steuerung:

- Interaktives SQL: F13=Service → 1.Sitzungsattribute ändern
  → Commit-Steuerung = *CHG
- ACS/Client Access: Connection → JDBC Config./Setup → Server
  → Commit Mode = *CHG
- Insert / Update / Delete-Statements: Hinzufügen von **WITH CS**

Row and Column Access Control (RCAC)

Was ist RCAC?

RCAC = Row and Column Access Control

- **Zusätzliche Schicht** um mit der Db2 **Datensicherheit** zu gewährleisten
  - Direkt mit den Tabellen/physischen Dateien verbunden
- **Zusätzlich** zu den **Objekt-Berechtigungen**
- **Beschränkt** den Zugriff auf die **eigentlichen Daten**
  - Kontrolliert den Zugriff auf **Zeilen und Spalten-Inhalte**
  - ***ALLOBJ Benutzer** können **nicht mehr beliebig** auf **alle Daten** zugreifen

2 unterschiedliche Möglichkeiten

- Beschränkung des Zugriffs auf Zeilen → `CREATE OR REPLACE PERMISSION`
- Beschränkung des Zugriffs auf Spalten-Inhalte → `CREATE OR REPLACE MASK`

Erfordert IBM Advanced Data Security feature for i

- **Installation erforderlich** → Kostenfreies Feature, Option 47
- Sowohl für **Entwicklungs-** als auch **Produktiv-Systeme** erforderlich

Noch Fragen?

References

SQL Reference
https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/db2/rbafzintro.htm

PDF Files for Database
https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/rzatd/rzatdprintable.htm

Database Information Finder
https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/rzatd/rzatdfinder.htm

Special Thanks to

Holger Scherer – RZKH Rechenzentrum Kreuznach

- For providing an IBM i-System enabling the creation of the samples/code used in my presentations
- http://www.rzkh.de

Your data is save! … in the bunker

Kurz-Biographie: Birgitta Hauser

Birgitta Hauser
Diplom-Betriebswirt (BA)
Database and Software Architect

Birgitta Hauser arbeitet seit 1992 auf der AS/400 und deren Nachfolger. Sie arbeitete sowohl in der Programmierung als klassischer RPG-Programmierer aber auch als Datenbanken- and Software-Architekt mit dem Schwerpunkt der Anwendungsmodernisierung von klassischen IBM i-Anwendungen. Seit Juli 2019 ist sie selbständig und im Bereich SQL, Anwendungs- und Datenbanken-Modernisierung (Db2 for i) tätig. Sie ist außerdem zeitweise als Contractor für Fresche Solutions Inc. (in Montreal) tätig.

Daneben hält sie weltweit (kundenindividuelle) Schulungen und Workshops für SQL- und RPG-Programmierer.

Seit 2002 referiert sie regelmäßig bei COMMON User Groups und IBM i – Veranstaltungen in Deutschland, anderen europäischen Ländern, sowie in den USA und Canada.

Desweiteren ist sie Co-Autorin von zwei IBM Redbooks. Außerdem schreibt sie regelmäßig zum Thema SQL, Datenbank und RPG für IBM Develperworks, sowie den ITP-Verlag (in Deutschland).

2015 hat sie den John Earl Speaker Scholarship Award und 2018 hat sie den Al Barsa Memorial Scholarship Award erhalten.

IBM Champion 2020

Vielen Dank

Verlagern von Programm-Logik in die Datenbank? Yes i can!

Bei Interesse an weiterführenden (kundenindividuellen) Schulungen vor Ort oder auch remote, oder an Unterstützung bei der Anwendungs- und Datenbanken-Modernisierung auf IBM i
Wenden Sie sich bitte direkt am mich

Birgitta.Hauser@FrescheSolutions.com
Hauser@SSS-Software.de