Diverse lesende APL-Zugriffe auf Datenbanken

Dr. Markos Mitsos
markos.mitsos@ergo.de

Deutsche Krankenversicherung AG DKV - ERGO, Aktuariat Gesundheit

APL-Tagung Bingen 2016

ERGO

Einleitung

Art des Vortrages:

- technisches Thema
- Details zur Datengrundlage von Simulationen

Einleitung

Art des Vortrages:

- technisches Thema
- Details zur Datengrundlage von Simulationen

Einleitung

Art des Vortrages:

- technisches Thema
- Details zur Datengrundlage von Simulationen

ERGO

Einleitung

Art des Vortrages:

- technisches Thema
- Details zur Datengrundlage von Simulationen

Worum geht es:

- lesende Verbindung zu diversen Datenbanken
- Optimierung von Laufzeit und Datenvolumen

ERGO

Einleitung

Art des Vortrages:

- technisches Thema
- Details zur Datengrundlage von Simulationen

Worum geht es:

- lesende Verbindung zu diversen Datenbanken
- Optimierung von Laufzeit und Datenvolumen

ERGO

Einleitung

Art des Vortrages:

- technisches Thema
- Details zur Datengrundlage von Simulationen

Worum geht es:

- lesende Verbindung zu diversen Datenbanken
- Optimierung von Laufzeit und Datenvolumen

ERGO

Übersicht

ERGO

Übersicht

ERGO

Übersicht

ERGO

Übersicht

1. Beispiele relevanter Datenbanken und oft benutzte Bestände
2. Bevorzugte Art der Verbindung von Daten und präferierter Datentyp
3. Unterteilung von Beständen nach Nummernkreisen
4. Sukzessives Einlesen von Beständen über offene Abfragen

ERGO

Übersicht über Datenbanken und Bestände

Allgemeine Informationen

Datenarten	Arten benötigter (Teil-) Bestände
Nebenbedingungen	technische und fachliche Nebenbedingungen
Datenbanken	Beispiele oft benutzter Datenbanken und Tabellen

Übersicht über Datenbanken und Bestände

Allgemeine Informationen

- Datenarten: Arten benötigter (Teil-) Bestände
- Nebenbedingungen: technische und fachliche Nebenbedingungen
- Datenbanken: Beispiele oft benutzter Datenbanken und Tabellen

Übersicht über Datenbanken und Bestände

Allgemeine Informationen

Datenarten	Arten benötigter (Teil-) Bestände
Nebenbedingungen	technische und fachliche Nebenbedingungen
Datenbanken	Beispiele oft benutzter Datenbanken und Tabellen

Übersicht über Datenbanken und Bestände

Allgemeine Informationen

Datenarten	Arten benötigter (Teil-) Bestände
Nebenbedingungen	technische und fachliche Nebenbedingungen
Datenbanken	Beispiele oft benutzter Datenbanken und Tabellen

ERGO

Übersicht über Datenbanken und Bestände

Allgemeine Informationen

Datenarten	Arten benötigter (Teil-) Bestände
Nebenbedingungen	technische und fachliche Nebenbedingungen
Datenbanken	Beispiele oft benutzter Datenbanken und Tabellen

Material zum Verständnis von Beispielen und Problemen.

ERGO

- Sehr häufig benötigte Daten
  - zu fachlichem Stichtag (Gültigkeit) und
  - zu technischem Stichtag (Sicht)
- und zwar
  - Gesamtbestand (z.B. Limitierungs-Simulation) oder
  - Teilbestand (z.B. vorbereitende Tests zu Beitragsanpassung), meistens über Liste von Versicherungsnummern definiert

- Sehr häufig benötigte Daten
  - zu fachlichem Stichtag (Gültigkeit) und
  - zu technischem Stichtag (Sicht)
- und zwar
  - Gesamtbestand (z.B. Limitierungs-Simulation) oder
  - Teilbestand (z.B. vorbereitende Tests zu Beitragsanpassung), meistens über Liste von Versicherungsnummern definiert

- Sehr häufig benötigte Daten
  - zu fachlichem Stichtag (Gültigkeit) und
  - zu technischem Stichtag (Sicht)
- und zwar
  - Gesamtbestand (z.B. Limitierungs-Simulation) oder
  - Teilbestand (z.B. vorbereitende Tests zu Beitragsanpassung), meistens über Liste von Versicherungsnummern definiert

- Sehr häufig benötigte Daten
  - zu fachlichem Stichtag (Gültigkeit) und
  - zu technischem Stichtag (Sicht)
- und zwar
  - Gesamtbestand (z.B. Limitierungs-Simulation) oder
  - Teilbestand (z.B. vorbereitende Tests zu Beitragsanpassung), meistens über Liste von Versicherungsnummern definiert

- Sehr häufig benötigte Daten
  - zu fachlichem Stichtag (Gültigkeit) und
  - zu technischem Stichtag (Sicht)
- und zwar
  - Gesamtbestand (z.B. Limitierungs-Simulation) oder
  - Teilbestand (z.B. vorbereitende Tests zu Beitragsanpassung), meistens über Liste von Versicherungsnummern definiert

- Sehr häufig benötigte Daten
  - zu fachlichem Stichtag (Gültigkeit) und
  - zu technischem Stichtag (Sicht)
- und zwar
  - Gesamtbestand (z.B. Limitierungs-Simulation) oder
  - Teilbestand (z.B. vorbereitende Tests zu Beitragsanpassung), meistens über Liste von Versicherungsnummern definiert

- Nicht so häufig
  - Bestand über fachlichen Zeitraum, z.B. 1 Jahr, (Kontrolle Tarifwechsel — in Planung)
  - Bestand über gesamte fachliche Laufzeit (z.B. Bestandsmigration)
- Selten
  - Bestand über mehrere Sichten
  - Bestand über alle Gültigkeiten und Sichten (z.B. Auflistung aller Tarifkombinationen)

- Nicht so häufig
  - Bestand über fachlichen Zeitraum, z.B. 1 Jahr, (Kontrolle Tarifwechsel — in Planung)
  - Bestand über gesamte fachliche Laufzeit (z.B. Bestandsmigration)
- Selten
  - Bestand über mehrere Sichten
  - Bestand über alle Gültigkeiten und Sichten (z.B. Auflistung aller Tarifkombinationen)

- Nicht so häufig
  - Bestand über fachlichen Zeitraum, z.B. 1 Jahr, (Kontrolle Tarifwechsel — in Planung)
  - Bestand über gesamte fachliche Laufzeit (z.B. Bestandsmigration)
- Selten
  - Bestand über mehrere Sichten
  - Bestand über alle Gültigkeiten und Sichten (z.B. Auflistung aller Tarifkombinationen)

- Nicht so häufig
  - Bestand über fachlichen Zeitraum, z.B. 1 Jahr, (Kontrolle Tarifwechsel — in Planung)
  - Bestand über gesamte fachliche Laufzeit (z.B. Bestandsmigration)
- Selten
  - Bestand über mehrere Sichten
  - Bestand über alle Gültigkeiten und Sichten (z.B. Auflistung aller Tarifkombinationen)

- Nicht so häufig
  - Bestand über fachlichen Zeitraum, z.B. 1 Jahr, (Kontrolle Tarifwechsel — in Planung)
  - Bestand über gesamte fachliche Laufzeit (z.B. Bestandsmigration)
- Selten
  - Bestand über mehrere Sichten
  - Bestand über alle Gültigkeiten und Sichten (z.B. Auflistung aller Tarifkombinationen)

- Nicht so häufig
  - Bestand über fachlichen Zeitraum, z.B. 1 Jahr, (Kontrolle Tarifwechsel — in Planung)
  - Bestand über gesamte fachliche Laufzeit (z.B. Bestandsmigration)
- Selten
  - Bestand über mehrere Sichten
  - Bestand über alle Gültigkeiten und Sichten (z.B. Auflistung aller Tarifkombinationen)

Wichtigste Nebenbedingungen

- Speicherplatz beschränkt
- Laufzeiten *müssen* vernünftig sein — sollen aber gut sein!
- fachlich zusammengehörende Sätze müssen gleichzeitig bearbeitet werden

Wichtigste Nebenbedingungen

- Speicherplatz beschränkt
- Laufzeiten *müssen* vernünftig sein — sollen aber gut sein!
- fachlich zusammengehörende Sätze müssen gleichzeitig bearbeitet werden

Wichtigste Nebenbedingungen

- Speicherplatz beschränkt
- Laufzeiten *müssen* vernünftig sein — sollen aber gut sein!
- fachlich zusammengehörende Sätze müssen gleichzeitig bearbeitet werden

ERGO

Wichtigste Nebenbedingungen

- Speicherplatz beschränkt
- Laufzeiten *müssen* vernünftig sein — sollen aber gut sein!
- fachlich zusammengehörende Sätze müssen gleichzeitig bearbeitet werden

ERGO

- Fachlich zusammengehörende Sätze
  - alle Sätze zu einer Versicherungsnummer (`VNR`) bzw. Teil davon
  - deshalb Sortierung nach `VNR` (quasi) immer notwendig
  - Sortierung führend nach `VNR`, ... sehr bequem
- Typischer Zugriff

```
SELECT ...
FROM ... T1
...
ORDER BY T1.VNR, T1.POS_LNR, T1.TA_ID, ...
```

- Fachlich zusammengehörende Sätze
  - alle Sätze zu einer Versicherungsnummer (`VNR`) bzw. Teil davon
  - deshalb Sortierung nach `VNR` (quasi) immer notwendig
  - Sortierung führend nach `VNR`, ... sehr bequem
- Typischer Zugriff

```
SELECT ...
FROM ... T1
...
ORDER BY T1.VNR, T1.POS_LNR, T1.TA_ID, ...
```

- Fachlich zusammengehörende Sätze
  - alle Sätze zu einer Versicherungsnummer (`VNR`) bzw. Teil davon
  - deshalb Sortierung nach `VNR` (quasi) immer notwendig
  - Sortierung führend nach `VNR`, ... sehr bequem
- Typischer Zugriff

```
SELECT ...
FROM ... T1
...
ORDER BY T1.VNR, T1.POS_LNR, T1.TA_ID, ...
```

- Fachlich zusammengehörende Sätze
  - alle Sätze zu einer Versicherungsnummer (`VNR`) bzw. Teil davon
  - deshalb Sortierung nach `VNR` (quasi) immer notwendig
  - Sortierung führend nach `VNR, ...` sehr bequem
- Typischer Zugriff

```
SELECT ...
FROM ... T1
...
ORDER BY T1.VNR, T1.POS_LNR, T1.TA_ID, ...
```

- Fachlich zusammengehörende Sätze
  - alle Sätze zu einer Versicherungsnummer (`VNR`) bzw. Teil davon
  - deshalb Sortierung nach `VNR` (quasi) immer notwendig
  - Sortierung führend nach `VNR, ...` sehr bequem
- Typischer Zugriff

```
SELECT ...
FROM ... T1
...
ORDER BY T1.VNR, T1.POS_LNR, T1.TA_ID, ...
```

Häufig eingesetzte Datenbanken und Tabellen

- dispositive Bestände in DB2 (fast immer Core Data Warehouse und sehr selten Data Marts)
- operative Bestände in DB2 (Krankenversicherungs-Datenbank)
- Kalkulationsdaten in SQL Server (Datenbank Aktuariat)

Häufig eingesetzte Datenbanken und Tabellen

- dispositive Bestände in DB2 (fast immer Core Data Warehouse und sehr selten Data Marts)
- operative Bestände in DB2 (Krankenversicherungs-Datenbank)
- Kalkulationsdaten in SQL Server (Datenbank Aktuariat)

Häufig eingesetzte Datenbanken und Tabellen

- dispositive Bestände in DB2 (fast immer Core Data Warehouse und sehr selten Data Marts)
- operative Bestände in DB2 (Krankenversicherungs-Datenbank)
- Kalkulationsdaten in SQL Server (Datenbank Aktuariat)

Häufig eingesetzte Datenbanken und Tabellen

- dispositive Bestände in DB2 (fast immer Core Data Warehouse und sehr selten Data Marts)
- operative Bestände in DB2 (Krankenversicherungs-Datenbank)
- Kalkulationsdaten in SQL Server (Datenbank Aktuariat)

- Standard für einzelvertragliche Simulationen dispositiv
  - zweidimensionale Zeitführung
  - leichter Stichtag
- Typischer Zugriff

```
SELECT ...
FROM &CREATOR.CDTBxxxx T1
WHERE T1.GUELTIG_AB    <= &DATUM
AND   T1.UNGUELTIG_AB >  &DATUM
AND   T1.BEST_EIN      <= &SICHT
AND   T1.BEST_AUS      >  &SICHT
...
```

- Standard für einzelvertragliche Simulationen dispositiv
  - zweidimensionale Zeitführung
  - leichter Stichtag
- Typischer Zugriff

```
SELECT ...
FROM &CREATOR.CDTBxxxx T1
WHERE T1.GUELTIG_AB    <= &DATUM
AND   T1.UNGUELTIG_AB >  &DATUM
AND   T1.BEST_EIN      <= &SICHT
AND   T1.BEST_AUS      >  &SICHT
...
```

- Standard für einzelvertragliche Simulationen dispositiv
  - zweidimensionale Zeitführung
  - leichter Stichtag
- Typischer Zugriff

```
SELECT ...
FROM &CREATOR.CDTBxxxx T1
WHERE T1.GUELTIG_AB    <= &DATUM
AND   T1.UNGUELTIG_AB >  &DATUM
AND   T1.BEST_EIN      <= &SICHT
AND   T1.BEST_AUS      >  &SICHT
...
```

- Standard für einzelvertragliche Simulationen dispositiv
  - zweidimensionale Zeitführung
  - leichter Stichtag
- Typischer Zugriff

```
SELECT ...
FROM &CREATOR.CDTBxxxx T1
WHERE T1.GUELTIG_AB    <= &DATUM
AND   T1.UNGUELTIG_AB  >  &DATUM
AND   T1.BEST_EIN      <= &SICHT
AND   T1.BEST_AUS      >  &SICHT
...
```

- Manchmal notwendig operativ
  - optimiert zur Bearbeitung einzelner Versicherungsnummern
  - einige Fragen müssen aber operativ beantwortet werden
- Typischer Zugriff

```
SELECT ...
FROM &CREATOR.KVTBxxxx T1
WHERE T1.V_VERW_SYS_SL = 'KV'
AND T1.GUELTIG_AB     <= &DATUM
AND T1.GUELTIG_BIS     ???  -- Stornoinformation
AND T1.HIST_LNR        ???  -- Bearbeitungs-
                            -- reihenfolge
AND T1.STATUS_MERKMAL  ???  -- technisch gültig
...
```

- Manchmal notwendig operativ
  - optimiert zur Bearbeitung einzelner Versicherungsnummern
  - einige Fragen müssen aber operativ beantwortet werden
- Typischer Zugriff

```
SELECT ...
FROM &CREATOR.KVTBxxxx T1
WHERE T1.V_VERW_SYS_SL = 'KV'
AND T1.GUELTIG_AB     <= &DATUM
AND T1.GUELTIG_BIS     ???  -- Stornoinformation
AND T1.HIST_LNR        ???  -- Bearbeitungs-
                            -- reihenfolge
AND T1.STATUS_MERKMAL  ???  -- technisch gültig
...
```

- Manchmal notwendig operativ
  - optimiert zur Bearbeitung einzelner Versicherungsnummern
  - einige Fragen müssen aber operativ beantwortet werden
- Typischer Zugriff

```
SELECT ...
FROM &CREATOR.KVTBxxxx T1
WHERE T1.V_VERW_SYS_SL = 'KV'
AND T1.GUELTIG_AB     <= &DATUM
AND T1.GUELTIG_BIS    ???  -- Stornoinformation
AND T1.HIST_LNR       ???  -- Bearbeitungs-
                           -- reihenfolge
AND T1.STATUS_MERKMAL ???  -- technisch gültig
...
```

- Manchmal notwendig operativ
  - optimiert zur Bearbeitung einzelner Versicherungsnummern
  - einige Fragen müssen aber operativ beantwortet werden
- Typischer Zugriff

```
SELECT ...
FROM &CREATOR.KVTBxxxx T1
WHERE T1.V_VERW_SYS_SL = 'KV'
AND T1.GUELTIG_AB      <= &DATUM
AND T1.GUELTIG_BIS     ???  -- Stornoinformation
AND T1.HIST_LNR        ???  -- Bearbeitungs-
                            -- reihenfolge
AND T1.STATUS_MERKMAL ???   -- technisch gültig
...
```

- Lösung nach Schema

```
SELECT ...
FROM &CREATOR.KVTBxxxx T1,
 ( SELECT ..., MAX(HIST_LNR)
   FROM &CREATOR.KVTBxxxx
   WHERE ...
   AND GUELTIG_AB <= &DATUM
   ... ) T2
WHERE T2.VNR = T1.VNR
AND ...
AND T2.HIST_LNR = T1.HIST_LNR
...
```

- nur theoretisch möglich
- praktisch zu langsam (und teuer)

- Lösung nach Schema

```
SELECT ...
FROM &CREATOR.KVTBxxxx T1,
 ( SELECT ..., MAX(HIST_LNR)
   FROM &CREATOR.KVTBxxxx
   WHERE ...
   AND GUELTIG_AB <= &DATUM
   ... ) T2
WHERE T2.VNR = T1.VNR
AND ...
AND T2.HIST_LNR = T1.HIST_LNR
...
```

- nur theoretisch möglich
- praktisch zu langsam (und teuer)

- Lösung nach Schema

```
SELECT ...
FROM &CREATOR.KVTBxxxx T1,
 ( SELECT ..., MAX(HIST_LNR)
   FROM &CREATOR.KVTBxxxx
   WHERE ...
   AND GUELTIG_AB <= &DATUM
   ... ) T2
WHERE T2.VNR = T1.VNR
AND ...
AND T2.HIST_LNR = T1.HIST_LNR
...
```

- nur theoretisch möglich
- praktisch zu langsam (und teuer)

ERGO

Übersicht über Datenverwaltung

Entscheidungen zur Bereitstellung und Führung von Daten

Datenverbindung Verbindung von Daten aus verschiedenen Quellen

Datentyp bei Bearbeitung bevorzugter Datentyp

Übersicht über Datenverwaltung

Entscheidungen zur Bereitstellung und Führung von Daten

Datenverbindung Verbindung von Daten aus verschiedenen Quellen

Datentyp bei Bearbeitung bevorzugter Datentyp

Übersicht über Datenverwaltung

Entscheidungen zur Bereitstellung und Führung von Daten

Datenverbindung Verbindung von Daten aus verschiedenen Quellen

Datentyp bei Bearbeitung bevorzugter Datentyp

Übersicht über Datenverwaltung

Entscheidungen zur Bereitstellung und Führung von Daten

Datenverbindung Verbindung von Daten aus verschiedenen Quellen

Datentyp bei Bearbeitung bevorzugter Datentyp

Grundlegende Entscheidungen zur Datenverwaltung.

- Wie Daten aus verschiedenen Quellen verbinden
  - in APL oder
  - per **JOIN** in Datenbank?
- Beispiel aktuarielle Daten aus einer Tabelle, Personendaten aus anderer

```
SELECT ...
FROM &CREATOR.CDTB0105 T1, &CREATOR.KVTB0103 T2
WHERE T2.VNR     = T1.VNR
AND   T2.POS_LNR = T1.POS_LNR
...
```

- Wie Daten aus verschiedenen Quellen verbinden
  - in APL oder
  - per **JOIN** in Datenbank?
- Beispiel aktuarielle Daten aus einer Tabelle, Personendaten aus anderer

```
SELECT ...
FROM &CREATOR.CDTB0105 T1, &CREATOR.KVTB0103 T2
WHERE T2.VNR     = T1.VNR
AND   T2.POS_LNR = T1.POS_LNR
...
```

- Wie Daten aus verschiedenen Quellen verbinden
  - in APL oder
  - per **JOIN** in Datenbank?
- Beispiel aktuarielle Daten aus einer Tabelle, Personendaten aus anderer

```
SELECT ...
FROM &CREATOR.CDTB0105 T1, &CREATOR.KVTB0103 T2
WHERE T2.VNR     = T1.VNR
AND   T2.POS_LNR = T1.POS_LNR
...
```

- Wie Daten aus verschiedenen Quellen verbinden
  - in APL oder
  - per **JOIN** in Datenbank?
- Beispiel aktuarielle Daten aus einer Tabelle, Personendaten aus anderer

```
SELECT ...
FROM &CREATOR.CDTB0105 T1, &CREATOR.KVTB0103 T2
WHERE T2.VNR      = T1.VNR
AND   T2.POS_LNR = T1.POS_LNR
...
```

Allgemeine Argumente

1. Pro **JOIN**
   - besser verständlich für Außenstehende
   - (sonst) geeignete Hilfsfunktionen in APL benötigt
2. Neutral
   - Zeiten in vielen Fällen vergleichbar ($m \times n$-Verbindung selten)
   - in APL exakt festgelegter Durchführungspfad für Verbindung (aber nicht für **SELECT**)
3. Pro APL
   - für **JOIN** in DB2 teure z/OS-CPU-Zeit, APL auf gewöhnlichen Windows-Servern
   - mainframe primär für Betrieb, Laufzeiten für dynamisches SQL instabil und nicht vorhersehbar

ERGO

Allgemeine Argumente

1. Pro **JOIN**
   - besser verständlich für Außenstehende
   - (sonst) geeignete Hilfsfunktionen in APL benötigt
2. Neutral
   - Zeiten in vielen Fällen vergleichbar ($m \times n$-Verbindung selten)
   - in APL exakt festgelegter Durchführungspfad für Verbindung (aber nicht für **SELECT**)
3. Pro APL
   - für **JOIN** in DB2 teure z/OS-CPU-Zeit, APL auf gewöhnlichen Windows-Servern
   - mainframe primär für Betrieb, Laufzeiten für dynamisches SQL instabil und nicht vorhersehbar

ERGO

Allgemeine Argumente

1. Pro **JOIN**
   - besser verständlich für Außenstehende
   - (sonst) geeignete Hilfsfunktionen in APL benötigt
2. Neutral
   - Zeiten in vielen Fällen vergleichbar ($m \times n$-Verbindung selten)
   - in APL exakt festgelegter Durchführungspfad für Verbindung (aber nicht für **SELECT**)
3. Pro APL
   - für **JOIN** in DB2 teure z/OS-CPU-Zeit, APL auf gewöhnlichen Windows-Servern
   - mainframe primär für Betrieb, Laufzeiten für dynamisches SQL instabil und nicht vorhersehbar

ERGO

Allgemeine Argumente

1. Pro **JOIN**
   - besser verständlich für Außenstehende
   - (sonst) geeignete Hilfsfunktionen in APL benötigt
2. Neutral
   - Zeiten in vielen Fällen vergleichbar ($m \times n$-Verbindung selten)
   - in APL exakt festgelegter Durchführungspfad für Verbindung (aber nicht für **SELECT**)
3. Pro APL
   - für **JOIN** in DB2 teure z/OS-CPU-Zeit, APL auf gewöhnlichen Windows-Servern
   - mainframe primär für Betrieb, Laufzeiten für dynamisches SQL instabil und nicht vorhersehbar

ERGO

Allgemeine Argumente

1. Pro **JOIN**
   - besser verständlich für Außenstehende
   - (sonst) geeignete Hilfsfunktionen in APL benötigt
2. Neutral
   - Zeiten in vielen Fällen vergleichbar ($m \times n$-Verbindung selten)
   - in APL exakt festgelegter Durchführungspfad für Verbindung (aber nicht für **SELECT**)
3. Pro APL
   - für **JOIN** in DB2 teure z/OS-CPU-Zeit, APL auf gewöhnlichen Windows-Servern
   - mainframe primär für Betrieb, Laufzeiten für dynamisches SQL instabil und nicht vorhersehbar

ERGO

Allgemeine Argumente

1. Pro **JOIN**
   - besser verständlich für Außenstehende
   - (sonst) geeignete Hilfsfunktionen in APL benötigt
2. Neutral
   - Zeiten in vielen Fällen vergleichbar ($m \times n$-Verbindung selten)
   - in APL exakt festgelegter Durchführungspfad für Verbindung (aber nicht für **SELECT**)
3. Pro APL
   - für **JOIN** in DB2 teure z/OS-CPU-Zeit, APL auf gewöhnlichen Windows-Servern
   - mainframe primär für Betrieb, Laufzeiten für dynamisches SQL instabil und nicht vorhersehbar

ERGO

Allgemeine Argumente

1. Pro **JOIN**
   - besser verständlich für Außenstehende
   - (sonst) geeignete Hilfsfunktionen in APL benötigt
2. Neutral
   - Zeiten in vielen Fällen vergleichbar ($m \times n$-Verbindung selten)
   - in APL exakt festgelegter Durchführungspfad für Verbindung (aber nicht für **SELECT**)
3. Pro APL
   - für **JOIN** in DB2 teure z/OS-CPU-Zeit, APL auf gewöhnlichen Windows-Servern
   - mainframe primär für Betrieb, Laufzeiten für dynamisches SQL instabil und nicht vorhersehbar

ERGO

Allgemeine Argumente

1. Pro **JOIN**
   - besser verständlich für Außenstehende
   - (sonst) geeignete Hilfsfunktionen in APL benötigt
2. Neutral
   - Zeiten in vielen Fällen vergleichbar ($m \times n$-Verbindung selten)
   - in APL exakt festgelegter Durchführungspfad für Verbindung (aber nicht für **SELECT**)
3. Pro APL
   - für **JOIN** in DB2 teure z/OS-CPU-Zeit, APL auf gewöhnlichen Windows-Servern
   - mainframe primär für Betrieb, Laufzeiten für dynamisches SQL instabil und nicht vorhersehbar

ERGO

Allgemeine Argumente

1. Pro **JOIN**
   - besser verständlich für Außenstehende
   - (sonst) geeignete Hilfsfunktionen in APL benötigt
2. Neutral
   - Zeiten in vielen Fällen vergleichbar ($m \times n$-Verbindung selten)
   - in APL exakt festgelegter Durchführungspfad für Verbindung (aber nicht für **SELECT**)
3. Pro APL
   - für **JOIN** in DB2 teure z/OS-CPU-Zeit, APL auf gewöhnlichen Windows-Servern
   - mainframe primär für Betrieb, Laufzeiten für dynamisches SQL instabil und nicht vorhersehbar

ERGO

Allgemeine Argumente

1. Pro **JOIN**
   - besser verständlich für Außenstehende
   - (sonst) geeignete Hilfsfunktionen in APL benötigt
2. Neutral
   - Zeiten in vielen Fällen vergleichbar ($m \times n$-Verbindung selten)
   - in APL exakt festgelegter Durchführungspfad für Verbindung (aber nicht für **SELECT**)
3. Pro APL
   - für **JOIN** in DB2 teure z/OS-CPU-Zeit, APL auf gewöhnlichen Windows-Servern
   - mainframe primär für Betrieb, Laufzeiten für dynamisches SQL instabil und nicht vorhersehbar

ERGO

- Durch **JOIN** teilweise signifikant mehr I/O
- eventuell ausschlaggebend
- Beispiel Stornodaten

```
SELECT T1.VNR, T1.POS_LNR, T1.TA_ID
FROM &CREATOR.CDTB0104 T1
WHERE ...
AND T1.UNGUELTIG_AB  = DATE(&DATUM) + 1 DAY
AND T1.TA_ABG_SL BETWEEN 500 AND 699
...
```

- Durch **JOIN** teilweise signifikant mehr I/O
- eventuell ausschlaggebend
- Beispiel Stornodaten

```
SELECT T1.VNR, T1.POS_LNR, T1.TA_ID
FROM &CREATOR.CDTB0104 T1
WHERE ...
AND T1.UNGUELTIG_AB  = DATE(&DATUM) + 1 DAY
AND T1.TA_ABG_SL BETWEEN 500 AND 699
...
```

- Durch **JOIN** teilweise signifikant mehr I/O
- eventuell ausschlaggebend
- Beispiel Stornodaten

```
SELECT T1.VNR, T1.POS_LNR, T1.TA_ID
FROM &CREATOR.CDTB0104 T1
WHERE ...
AND T1.UNGUELTIG_AB  = DATE(&DATUM) + 1 DAY
AND T1.TA_ABG_SL BETWEEN 500 AND 699
...
```

Prinzipielle Hindernisse für **JOIN**

- Daten aus unterschiedlichen subsystems (z.B. operativ / dispositiv)
- Daten aus unterschiedlichen Datenbanken (z.B. ergänzende Daten aus SQL Server) — Gefahr unsinniger Ausführung
- Tabellen nicht zur Verknüpfung geeignet (operative Tabellen, Werte nach halboffenen Intervallen bzw. gelten bis zur nächsten `HIST_LNR`)
- ergänzende Informationen aus weiteren Quellen (z.B. Dateien)

Prinzipielle Hindernisse für **JOIN**

- Daten aus unterschiedlichen subsystems (z.B. operativ / dispositiv)
- Daten aus unterschiedlichen Datenbanken (z.B. ergänzende Daten aus SQL Server) — Gefahr unsinniger Ausführung
- Tabellen nicht zur Verknüpfung geeignet (operative Tabellen, Werte nach halboffenen Intervallen bzw. gelten bis zur nächsten `HIST_LNR`)
- ergänzende Informationen aus weiteren Quellen (z.B. Dateien)

Prinzipielle Hindernisse für **JOIN**

- Daten aus unterschiedlichen subsystems (z.B. operativ / dispositiv)
- Daten aus unterschiedlichen Datenbanken (z.B. ergänzende Daten aus SQL Server) — Gefahr unsinniger Ausführung
- Tabellen nicht zur Verknüpfung geeignet (operative Tabellen, Werte nach halboffenen Intervallen bzw. gelten bis zur nächsten `HIST_LNR`)
- ergänzende Informationen aus weiteren Quellen (z.B. Dateien)

ERGO

Prinzipielle Hindernisse für **JOIN**

- Daten aus unterschiedlichen subsystems (z.B. operativ / dispositiv)
- Daten aus unterschiedlichen Datenbanken (z.B. ergänzende Daten aus SQL Server) — Gefahr unsinniger Ausführung
- Tabellen nicht zur Verknüpfung geeignet (operative Tabellen, Werte nach halboffenen Intervallen bzw. gelten bis zur nächsten `HIST_LNR`)
- ergänzende Informationen aus weiteren Quellen (z.B. Dateien)

Prinzipielle Hindernisse für **JOIN**

- Daten aus unterschiedlichen subsystems (z.B. operativ / dispositiv)
- Daten aus unterschiedlichen Datenbanken (z.B. ergänzende Daten aus SQL Server) — Gefahr unsinniger Ausführung
- Tabellen nicht zur Verknüpfung geeignet (operative Tabellen, Werte nach halboffenen Intervallen bzw. gelten bis zur nächsten `HIST_LNR`)
- ergänzende Informationen aus weiteren Quellen (z.B. Dateien)

Persönliches Vorgehen

- Daten in APL verbinden
- nur in Ausnahmen **JOIN**, hauptsächlich kleine „Dimensionen“
- eine Tabelle führend, meistens **CDTB0105**

Persönliches Vorgehen

- Daten in APL verbinden
- nur in Ausnahmen **JOIN**, hauptsächlich kleine „Dimensionen“
- eine Tabelle führend, meistens **CDTB0105**

Persönliches Vorgehen

- Daten in APL verbinden
- nur in Ausnahmen **JOIN**, hauptsächlich kleine „Dimensionen“
- eine Tabelle führend, meistens CDTB0105

Persönliches Vorgehen

- Daten in APL verbinden
- nur in Ausnahmen **JOIN**, hauptsächlich kleine „Dimensionen“
- eine Tabelle führend, meistens **CDTB0105**

- Präferierter Datentyp
  - wo möglich 323 (32-bit integer)
  - wo nötig 645 (64-bit float)
  - vermeide 807 (80-bit heterogeneous)
- Fast immer möglich
  - numerische Schlüssel
  - Schlüssel mit wenigen (≤ 10) Werten über Fallunterscheidung
  - Tarifnamen u.Ä. in Hilfsvariablen auslagern
  - Datum als ganze Zahl, *YYYYMMDD*
  - aktuarielle Werte mit 2 oder 4 Nachkommastellen

ERGO

- Präferierter Datentyp
  - wo möglich 323 (32-bit integer)
  - wo nötig 645 (64-bit float)
  - vermeide 807 (80-bit heterogeneous)
- Fast immer möglich
  - numerische Schlüssel
  - Schlüssel mit wenigen (≤ 10) Werten über Fallunterscheidung
  - Tarifnamen u.Ä. in Hilfsvariablen auslagern
  - Datum als ganze Zahl, *YYYYMMDD*
  - aktuarielle Werte mit 2 oder 4 Nachkommastellen

ERGO

- Präferierter Datentyp
  - wo möglich 323 (32-bit integer)
  - wo nötig 645 (64-bit float)
  - vermeide 807 (80-bit heterogeneous)
- Fast immer möglich
  - numerische Schlüssel
  - Schlüssel mit wenigen (≤ 10) Werten über Fallunterscheidung
  - Tarifnamen u.Ä. in Hilfsvariablen auslagern
  - Datum als ganze Zahl, *YYYYMMDD*
  - aktuarielle Werte mit 2 oder 4 Nachkommastellen

ERGO

- Präferierter Datentyp
  - wo möglich 323 (32-bit integer)
  - wo nötig 645 (64-bit float)
  - vermeide 807 (80-bit heterogeneous)
- Fast immer möglich
  - numerische Schlüssel
  - Schlüssel mit wenigen (≤ 10) Werten über Fallunterscheidung
  - Tarifnamen u.Ä. in Hilfsvariablen auslagern
  - Datum als ganze Zahl, *YYYYMMDD*
  - aktuarielle Werte mit 2 oder 4 Nachkommastellen

- Präferierter Datentyp
  - wo möglich 323 (32-bit integer)
  - wo nötig 645 (64-bit float)
  - vermeide 807 (80-bit heterogeneous)
- Fast immer möglich
  - numerische Schlüssel
  - Schlüssel mit wenigen (≤ 10) Werten über Fallunterscheidung
  - Tarifnamen u.Ä. in Hilfsvariablen auslagern
  - Datum als ganze Zahl, *YYYYMMDD*
  - aktuarielle Werte mit 2 oder 4 Nachkommastellen

ERGO

- Präferierter Datentyp
  - wo möglich 323 (32-bit integer)
  - wo nötig 645 (64-bit float)
  - vermeide 807 (80-bit heterogeneous)
- Fast immer möglich
  - numerische Schlüssel
  - Schlüssel mit wenigen (≤ 10) Werten über Fallunterscheidung
  - Tarifnamen u.Ä. in Hilfsvariablen auslagern
  - Datum als ganze Zahl, *YYYYMMDD*
  - aktuarielle Werte mit 2 oder 4 Nachkommastellen

- Präferierter Datentyp
  - wo möglich 323 (32-bit integer)
  - wo nötig 645 (64-bit float)
  - vermeide 807 (80-bit heterogeneous)
- Fast immer möglich
  - numerische Schlüssel
  - Schlüssel mit wenigen (≤ 10) Werten über Fallunterscheidung
  - Tarifnamen u.Ä. in Hilfsvariablen auslagern
  - Datum als ganze Zahl, *YYYYMMDD*
  - aktuarielle Werte mit 2 oder 4 Nachkommastellen

- Präferierter Datentyp
  - wo möglich 323 (32-bit integer)
  - wo nötig 645 (64-bit float)
  - vermeide 807 (80-bit heterogeneous)
- Fast immer möglich
  - numerische Schlüssel
  - Schlüssel mit wenigen (≤ 10) Werten über Fallunterscheidung
  - Tarifnamen u.Ä. in Hilfsvariablen auslagern
  - Datum als ganze Zahl, *YYYYMMDD*
  - aktuarielle Werte mit 2 oder 4 Nachkommastellen

- Präferierter Datentyp
  - wo möglich 323 (32-bit integer)
  - wo nötig 645 (64-bit float)
  - vermeide 807 (80-bit heterogeneous)
- Fast immer möglich
  - numerische Schlüssel
  - Schlüssel mit wenigen (≤ 10) Werten über Fallunterscheidung
  - Tarifnamen u.Ä. in Hilfsvariablen auslagern
  - Datum als ganze Zahl, *YYYYMMDD*
  - aktuarielle Werte mit 2 oder 4 Nachkommastellen

- Präferierter Datentyp
  - wo möglich 323 (32-bit integer)
  - wo nötig 645 (64-bit float)
  - vermeide 807 (80-bit heterogeneous)
- Fast immer möglich
  - numerische Schlüssel
  - Schlüssel mit wenigen (≤ 10) Werten über Fallunterscheidung
  - Tarifnamen u.Ä. in Hilfsvariablen auslagern
  - Datum als ganze Zahl, *YYYYMMDD*
  - aktuarielle Werte mit 2 oder 4 Nachkommastellen

ERGO

Beispiele Einlesen als four byte integer

```
T1.TA_ID "TAID"
INT(T1.VNR) "VNR"
POSITION(SUBSTR(T1.POS_LNR, 1, 1),
     'ABCDEFGHIJKLMNOPQRSTUVWXYZ', OCTETS) "POS"
(CASE WHEN T1.GUELTIG_BIS = '9999-12-31' THEN 1
     ELSE 0 END) "LAUF-KZ"
INT(HEX(T1.GUELTIG_AB)) "VON"
(CASE WHEN T1.AW_ART_SL IN ('2', '3', '4', '8') THEN 2
      WHEN T1.AW_ART_SL IN ('1', '6', '9')      THEN 3
      WHEN T1.TARIF_SL_L =   '1'                THEN 4
      ELSE 1 END) "AGGR-ZUST-SL"
CAST (10000 * T1.RBT_BETR AS INTEGER) "RBT"
CAST (100 * T1.RUE_BIL_VJ AS INTEGER) "RUE-BIL-VJ"
```

- Vergleich eindeutig
  - Bearbeitung in DB2 größtenteils vergleichbar mit APL
  - I/O und verbundene Lauzeit sehr stark reduziert
  - Speicherplatz-Verwaltung in APL besser
- Persönliches Vorgehen
  - wenn irgendwie möglich gewünschten Datentyp auf Datenbankseite erzwingen
  - so gut es geht Hauptdaten direkt als four byte integer einlesen

- Vergleich eindeutig
  - Bearbeitung in DB2 größtenteils vergleichbar mit APL
  - I/O und verbundene Lauzeit sehr stark reduziert
  - Speicherplatz-Verwaltung in APL besser
- Persönliches Vorgehen
  - wenn irgendwie möglich gewünschten Datentyp auf Datenbankseite erzwingen
  - so gut es geht Hauptdaten direkt als four byte integer einlesen

- Vergleich eindeutig
  - Bearbeitung in DB2 größtenteils vergleichbar mit APL
  - I/O und verbundene Lauzeit sehr stark reduziert
  - Speicherplatz-Verwaltung in APL besser
- Persönliches Vorgehen
  - wenn irgendwie möglich gewünschten Datentyp auf Datenbankseite erzwingen
  - so gut es geht Hauptdaten direkt als four byte integer einlesen

- Vergleich eindeutig
  - Bearbeitung in DB2 größtenteils vergleichbar mit APL
  - I/O und verbundene Lauzeit sehr stark reduziert
  - Speicherplatz-Verwaltung in APL besser
- Persönliches Vorgehen
  - wenn irgendwie möglich gewünschten Datentyp auf Datenbankseite erzwingen
  - so gut es geht Hauptdaten direkt als four byte integer einlesen

ERGO

- Vergleich eindeutig
  - Bearbeitung in DB2 größtenteils vergleichbar mit APL
  - I/O und verbundene Lauzeit sehr stark reduziert
  - Speicherplatz-Verwaltung in APL besser
- Persönliches Vorgehen
  - wenn irgendwie möglich gewünschten Datentyp auf Datenbankseite erzwingen
  - so gut es geht Hauptdaten direkt als four byte integer einlesen

ERGO

- Vergleich eindeutig
  - Bearbeitung in DB2 größtenteils vergleichbar mit APL
  - I/O und verbundene Lauzeit sehr stark reduziert
  - Speicherplatz-Verwaltung in APL besser
- Persönliches Vorgehen
  - wenn irgendwie möglich gewünschten Datentyp auf Datenbankseite erzwingen
  - so gut es geht Hauptdaten direkt als four byte integer einlesen

ERGO

- Vergleich eindeutig
  - Bearbeitung in DB2 größtenteils vergleichbar mit APL
  - I/O und verbundene Lauzeit sehr stark reduziert
  - Speicherplatz-Verwaltung in APL besser
- Persönliches Vorgehen
  - wenn irgendwie möglich gewünschten Datentyp auf Datenbankseite erzwingen
  - so gut es geht Hauptdaten direkt als four byte integer einlesen

ERGO

Übersicht über Unterteilung in Nummernkreise

Einfache Technik zur Bewältigung großer Datenmengen

Nummernkreise	Bearbeitung großer Datenmengen über Nummernkreise
Abschnitte	Zusammenfassung von Daten zu Bestandsabschnitten
Führende Tabelle	Verwendung offenen Teilabfragen auf die führende Tabelle

ERGO

Übersicht über Unterteilung in Nummernkreise

Einfache Technik zur Bewältigung großer Datenmengen

Nummernkreise	Bearbeitung großer Datenmengen über Nummernkreise
Abschnitte	Zusammenfassung von Daten zu Bestandsabschnitten
Führende Tabelle	Verwendung offenen Teilabfragen auf die führende Tabelle

Übersicht über Unterteilung in Nummernkreise

Einfache Technik zur Bewältigung großer Datenmengen

Nummernkreise Bearbeitung großer Datenmengen über Nummernkreise

Abschnitte Zusammenfassung von Daten zu Bestandsabschnitten

Führende Tabelle Verwendung offenen Teilabfragen auf die führende Tabelle

Übersicht über Unterteilung in Nummernkreise

Einfache Technik zur Bewältigung großer Datenmengen

Nummernkreise	Bearbeitung großer Datenmengen über Nummernkreise
Abschnitte	Zusammenfassung von Daten zu Bestandsabschnitten
Führende Tabelle	Verwendung offenen Teilabfragen auf die führende Tabelle

ERGO

Übersicht über Unterteilung in Nummernkreise

Einfache Technik zur Bewältigung großer Datenmengen

Nummernkreise	Bearbeitung großer Datenmengen über Nummernkreise
Abschnitte	Zusammenfassung von Daten zu Bestandsabschnitten
Führende Tabelle	Verwendung offenen Teilabfragen auf die führende Tabelle

Einlesen großer Datenmengen über eine Reihe parametrisierter Abfragen.

Datenverwaltung

- optimal wenn alle Daten im Hauptspeicher
- sonst suche minimale notwendige Schleifen
- z.B. Hauptschleife (Nummernkreis `NK_NR`)
- nach Hauptschlüssel (Versicherungsnummer `VNR`)
- jeweils so viele Sätze wie möglich (Nummernkreislänge, z.B. `NK_L = 10*9`)
- eingelesene Daten garantiert synchronisiert

Datenverwaltung

- optimal wenn alle Daten im Hauptspeicher
- sonst suche minimale notwendige Schleifen
- z.B. Hauptschleife (Nummernkreis `NK_NR`)
- nach Hauptschlüssel (Versicherungsnummer `VNR`)
- jeweils so viele Sätze wie möglich (Nummernkreislänge, z.B. `NK_L = 10*9`)
- eingelesene Daten garantiert synchronisiert

Datenverwaltung

- optimal wenn alle Daten im Hauptspeicher
- sonst suche minimale notwendige Schleifen
  - z.B. Hauptschleife (Nummernkreis `NK_NR`)
  - nach Hauptschlüssel (Versicherungsnummer `VNR`)
  - jeweils so viele Sätze wie möglich (Nummernkreislänge, z.B. `NK_L = 10*9`)
  - eingelesene Daten garantiert synchronisiert

Datenverwaltung

- optimal wenn alle Daten im Hauptspeicher
- sonst suche minimale notwendige Schleifen
- z.B. Hauptschleife (Nummernkreis `NK_NR`)
- nach Hauptschlüssel (Versicherungsnummer `VNR`)
- jeweils so viele Sätze wie möglich (Nummernkreislänge, z.B. `NK_L = 10*9`)
- eingelesene Daten garantiert synchronisiert

Datenverwaltung

- optimal wenn alle Daten im Hauptspeicher
- sonst suche minimale notwendige Schleifen
- z.B. Hauptschleife (Nummernkreis `NK_NR`)
- nach Hauptschlüssel (Versicherungsnummer `VNR`)
- jeweils so viele Sätze wie möglich (Nummernkreislänge, z.B. `NK_L = 10*9`)
- eingelesene Daten garantiert synchronisiert

Datenverwaltung

- optimal wenn alle Daten im Hauptspeicher
- sonst suche minimale notwendige Schleifen
- z.B. Hauptschleife (Nummernkreis `NK_NR`)
- nach Hauptschlüssel (Versicherungsnummer `VNR`)
- jeweils so viele Sätze wie möglich (Nummernkreislänge, z.B. `NK_L = 10*9`)
- eingelesene Daten garantiert synchronisiert

Datenverwaltung

- optimal wenn alle Daten im Hauptspeicher
- sonst suche minimale notwendige Schleifen
- z.B. Hauptschleife (Nummernkreis `NK_NR`)
- nach Hauptschlüssel (Versicherungsnummer `VNR`)
- jeweils so viele Sätze wie möglich (Nummernkreislänge, z.B. `NK_L = 10*9`)
- eingelesene Daten garantiert synchronisiert

- Beispiel Limitierungs-Simulation
  - mit VNR_VON = (¯1 + NK_NR) × NK_L
  - und VNR_BIS = ¯1 + (NK_NR × NK_L)
  - ```
    SELECT ...
    FROM &CREATOR.CDTB0105 T1
    WHERE T1.VNR BETWEEN &VNR_VON AND &VNR_BIS
    ...
    ORDER BY "VNR", ...
    ```
    ```

- Beispiel Limitierungs-Simulation
  - mit `VNR_VON = (¯1 + NK_NR) × NK_L`
  - und `VNR_BIS = ¯1 + (NK_NR × NK_L)`
  - `SELECT ...`
    `FROM &CREATOR.CDTB0105 T1`
    `WHERE T1.VNR BETWEEN &VNR_VON AND &VNR_BIS`
    `...`
    `ORDER BY "VNR", ...`

- Beispiel Limitierungs-Simulation
  - mit VNR_VON = (¯1 + NK_NR) × NK_L
  - und VNR_BIS = ¯1 + (NK_NR × NK_L)
  - ```
    SELECT ...
    FROM &CREATOR.CDTB0105 T1
    WHERE T1.VNR BETWEEN &VNR_VON AND &VNR_BIS
    ...
    ORDER BY "VNR", ...
    ```
    ```

- Beispiel Limitierungs-Simulation
  - mit VNR_VON = (¯1 + NK_NR) × NK_L
  - und VNR_BIS = ¯1 + (NK_NR × NK_L)
  - ```
    SELECT ...
    FROM &CREATOR.CDTB0105 T1
    WHERE T1.VNR BETWEEN &VNR_VON AND &VNR_BIS
    ...
    ORDER BY "VNR", ...
    ```
    ```

- Alternativ Teilbestand
  - mit geeigneter, separat erzeugter Hilfstabelle **HTABN**
  - ```
    SELECT ...
    FROM &CREATOR.CDTB0105 T1, &HTABN T2
    WHERE T1.VNR = T2.VNR
    ...
    ORDER BY "VNR", ...
    ```
    ```

- Alternativ Teilbestand
  - mit geeigneter, separat erzeugter Hilfstabelle **HTABN**
  - ```
    SELECT ...
    FROM &CREATOR.CDTB0105 T1, &HTABN T2
    WHERE T1.VNR = T2.VNR
    ...
    ORDER BY "VNR", ...
    ```
    ```

- Alternativ Teilbestand
  - mit geeigneter, separat erzeugter Hilfstabelle **HTABN**
  - ```
    SELECT ...
    FROM &CREATOR.CDTB0105 T1, &HTABN T2
    WHERE T1.VNR = T2.VNR
    ...
    ORDER BY "VNR", ...
    ```
    ```

- Prinzipielle Probleme der einfachen Unterteilung
  - Bestand vollkommen ungleichmäßig über `VNR` verteilt
  - mehrere Hundert Abfragen mit leerem Ergebnis (reiner Zeitverlust)
  - teilweise Bearbeitung kleiner Ergebnisse in APL
  - kein Optimum erreichbar
- Strebe gleichmäßige Bestandsabschnitte an
  - z.B. mit Abschnittslänge `ABS_L = 10*6`

- Prinzipielle Probleme der einfachen Unterteilung
  - Bestand vollkommen ungleichmäßig über `VNR` verteilt
  - mehrere Hundert Abfragen mit leerem Ergebnis (reiner Zeitverlust)
  - teilweise Bearbeitung kleiner Ergebnisse in APL
  - kein Optimum erreichbar
- Strebe gleichmäßige Bestandsabschnitte an
  - z.B. mit Abschnittslänge `ABS_L = 10*6`

- Prinzipielle Probleme der einfachen Unterteilung
  - Bestand vollkommen ungleichmäßig über `VNR` verteilt
  - mehrere Hundert Abfragen mit leerem Ergebnis (reiner Zeitverlust)
  - teilweise Bearbeitung kleiner Ergebnisse in APL
  - kein Optimum erreichbar
- Strebe gleichmäßige Bestandsabschnitte an
  - z.B. mit Abschnittslänge `ABS_L = 10*6`

- Prinzipielle Probleme der einfachen Unterteilung
  - Bestand vollkommen ungleichmäßig über `VNR` verteilt
  - mehrere Hundert Abfragen mit leerem Ergebnis (reiner Zeitverlust)
  - teilweise Bearbeitung kleiner Ergebnisse in APL
  - kein Optimum erreichbar
- Strebe gleichmäßige Bestandsabschnitte an
  - z.B. mit Abschnittslänge `ABS_L = 10*6`

- Prinzipielle Probleme der einfachen Unterteilung
  - Bestand vollkommen ungleichmäßig über `VNR` verteilt
  - mehrere Hundert Abfragen mit leerem Ergebnis (reiner Zeitverlust)
  - teilweise Bearbeitung kleiner Ergebnisse in APL
  - kein Optimum erreichbar
- Strebe gleichmäßige Bestandsabschnitte an
  - z.B. mit Abschnittslänge `ABS_L = 10*6`

- Prinzipielle Probleme der einfachen Unterteilung
  - Bestand vollkommen ungleichmäßig über `VNR` verteilt
  - mehrere Hundert Abfragen mit leerem Ergebnis (reiner Zeitverlust)
  - teilweise Bearbeitung kleiner Ergebnisse in APL
  - kein Optimum erreichbar
- Strebe gleichmäßige Bestandsabschnitte an
  - z.B. mit Abschnittslänge `ABS_L = 10*6`

- Prinzipielle Probleme der einfachen Unterteilung
  - Bestand vollkommen ungleichmäßig über `VNR` verteilt
  - mehrere Hundert Abfragen mit leerem Ergebnis (reiner Zeitverlust)
  - teilweise Bearbeitung kleiner Ergebnisse in APL
  - kein Optimum erreichbar
- Strebe gleichmäßige Bestandsabschnitte an
  - z.B. mit Abschnittslänge `ABS_L = 10*6`

- Erste Verbesserung (in APL)
  - Bestand vor Bearbeitung sammeln

```
:FOR NK_NR :IN ...
    DATEN ← (...) DATEN_EINLESEN (NK_NR ...)
    BESTAND ← BESTAND ,[1] DATEN
    :WHILE ABS_L ≤ ↑ ρBESTAND
        I ← ...
        DATEN ← I ≠ BESTAND
        BESTAND ← (~ I) ≠ BESTAND
        ...
    :ENDWHILE
:ENDFOR
```

  - damit zumindest für weitere Schritte gleichmäßige Abschnitte
  - aber erste Randfälle (z.B. letzter Nummernkreis) zu beachten

- Erste Verbesserung (in APL)
  - Bestand vor Bearbeitung sammeln

```
:FOR NK_NR :IN ...
    DATEN ← (...) DATEN_EINLESEN (NK_NR ...)
    BESTAND ← BESTAND ,[1] DATEN
    :WHILE ABS_L ≤ ↑ ρBESTAND
        I  ← ...
        DATEN    ←    I  ≠ BESTAND
        BESTAND ← (~ I) ≠ BESTAND
        ...
    :ENDWHILE
:ENDFOR
```

  - damit zumindest für weitere Schritte gleichmäßige Abschnitte
  - aber erste Randfälle (z.B. letzter Nummernkreis) zu beachten

- Erste Verbesserung (in APL)
  - Bestand vor Bearbeitung sammeln

```
:FOR NK_NR :IN ...
    DATEN ← (...) DATEN_EINLESEN (NK_NR ...)
    BESTAND ← BESTAND ,[1] DATEN
    :WHILE ABS_L ≤ ↑ ρBESTAND
        I  ← ...
        DATEN    ←    I  ≠ BESTAND
        BESTAND ← (~ I) ≠ BESTAND
        ...
    :ENDWHILE
:ENDFOR
```

  - damit zumindest für weitere Schritte gleichmäßige Abschnitte
  - aber erste Randfälle (z.B. letzter Nummernkreis) zu beachten

- Erste Verbesserung (in APL)
  - Bestand vor Bearbeitung sammeln

```
:FOR NK_NR :IN ...
    DATEN ← (...) DATEN_EINLESEN (NK_NR ...)
    BESTAND ← BESTAND ,[1] DATEN
    :WHILE ABS_L ≤ ↑ ρBESTAND
        I  ← ...
        DATEN    ←    I  ≠ BESTAND
        BESTAND ← (~ I) ≠ BESTAND
        ...
    :ENDWHILE
:ENDFOR
```

  - damit zumindest für weitere Schritte gleichmäßige Abschnitte
  - aber erste Randfälle (z.B. letzter Nummernkreis) zu beachten

- Praktisches Problem der schlechten Kontrolle
  - große Hilfstabellen können Hauptspeicher füllen
  - ähnlich bei sehr vielen Spalten und/oder Abschnitten mit viel Bestand (z.B. Beitragsanpassungs-Simulation)
- Dazu teilweise offene Abfragen mit

`0 ADO_SELECT SEL`	führt **SELECT** durch und gibt alle Ergebnissätze zurück
`0 ADO_SELECT (0 SEL 0)`	führt **PREPARE** aus, öffnet **CURSOR** und gibt Ergebnisbeschreibung zurück
`ABS_L ADO_SELECT (1 '' 0)`	führt **FETCH** durch, gibt maximal `ABS_L` Ergebnissätze zurück und lässt **CURSOR** offen

- Praktisches Problem der schlechten Kontrolle
  - große Hilfstabellen können Hauptspeicher füllen
  - ähnlich bei sehr vielen Spalten und/oder Abschnitten mit viel Bestand (z.B. Beitragsanpassungs-Simulation)
- Dazu teilweise offene Abfragen mit

`0 ADO_SELECT SEL`	führt **SELECT** durch und gibt alle Ergebnissätze zurück
`0 ADO_SELECT (0 SEL 0)`	führt **PREPARE** aus, öffnet **CURSOR** und gibt Ergebnisbeschreibung zurück
`ABS_L ADO_SELECT (1 '' 0)`	führt **FETCH** durch, gibt maximal `ABS_L` Ergebnissätze zurück und lässt **CURSOR** offen

- Praktisches Problem der schlechten Kontrolle
  - große Hilfstabellen können Hauptspeicher füllen
  - ähnlich bei sehr vielen Spalten und/oder Abschnitten mit viel Bestand (z.B. Beitragsanpassungs-Simulation)
- Dazu teilweise offene Abfragen mit

`0 ADO_SELECT SEL`	führt **SELECT** durch und gibt alle Ergebnissätze zurück
`0 ADO_SELECT (0 SEL 0)`	führt **PREPARE** aus, öffnet **CURSOR** und gibt Ergebnisbeschreibung zurück
`ABS_L ADO_SELECT (1 '' 0)`	führt **FETCH** durch, gibt maximal `ABS_L` Ergebnissätze zurück und lässt **CURSOR** offen

- Praktisches Problem der schlechten Kontrolle
  - große Hilfstabellen können Hauptspeicher füllen
  - ähnlich bei sehr vielen Spalten und/oder Abschnitten mit viel Bestand (z.B. Beitragsanpassungs-Simulation)
- Dazu teilweise offene Abfragen mit

`0 ADO_SELECT SEL` führt **SELECT** durch und gibt alle Ergebnissätze zurück

`0 ADO_SELECT (0 SEL 0)` führt **PREPARE** aus, öffnet **CURSOR** und gibt Ergebnisbeschreibung zurück

`ABS_L ADO_SELECT (1 '' 0)` führt **FETCH** durch, gibt maximal `ABS_L` Ergebnissätze zurück und lässt **CURSOR** offen

- Praktisches Problem der schlechten Kontrolle
  - große Hilfstabellen können Hauptspeicher füllen
  - ähnlich bei sehr vielen Spalten und/oder Abschnitten mit viel Bestand (z.B. Beitragsanpassungs-Simulation)
- Dazu teilweise offene Abfragen mit

`0 ADO_SELECT SEL` führt **SELECT** durch und gibt alle Ergebnissätze zurück

`0 ADO_SELECT (0 SEL 0)` führt **PREPARE** aus, öffnet **CURSOR** und gibt Ergebnisbeschreibung zurück

`ABS_L ADO_SELECT (1 '' 0)` führt **FETCH** durch, gibt maximal `ABS_L` Ergebnissätze zurück und lässt **CURSOR** offen

- Praktisches Problem der schlechten Kontrolle
  - große Hilfstabellen können Hauptspeicher füllen
  - ähnlich bei sehr vielen Spalten und/oder Abschnitten mit viel Bestand (z.B. Beitragsanpassungs-Simulation)
- Dazu teilweise offene Abfragen mit

`0 ADO_SELECT SEL`	führt **SELECT** durch und gibt alle Ergebnissätze zurück
`0 ADO_SELECT (0 SEL 0)`	führt **PREPARE** aus, öffnet **CURSOR** und gibt Ergebnisbeschreibung zurück
`ABS_L ADO_SELECT (1 '' 0)`	führt **FETCH** durch, gibt maximal `ABS_L` Ergebnissätze zurück und lässt **CURSOR** offen

- Praktisches Problem der schlechten Kontrolle
  - große Hilfstabellen können Hauptspeicher füllen
  - ähnlich bei sehr vielen Spalten und/oder Abschnitten mit viel Bestand (z.B. Beitragsanpassungs-Simulation)
- Dazu teilweise offene Abfragen mit

`0 ADO_SELECT SEL`	führt **SELECT** durch und gibt alle Ergebnissätze zurück
`0 ADO_SELECT (0 SEL 0)`	führt **PREPARE** aus, öffnet **CURSOR** und gibt Ergebnisbeschreibung zurück
`ABS_L ADO_SELECT (1 '' 0)`	führt **FETCH** durch, gibt maximal `ABS_L` Ergebnissätze zurück und lässt **CURSOR** offen

- Falls führende Tabelle (= die gewünschten Zeilen bestimmend)
  - zusätzliche innere Schleife

```
FERTIG ← 0
DATEN_105_DEF ← 0 ADO_SELECT (0 SEL_105 0)
...
:WHILE ~FERTIG
    DATEN_105 ← ABS_L ADO_SELECT (1 '' 0)
    FERTIG ← ABS_L > ↑ ρDATEN_105
    ...
:ENDWHILE
```

- Falls führende Tabelle (= die gewünschten Zeilen bestimmend)
  - zusätzliche innere Schleife

```
FERTIG ← 0
DATEN_105_DEF ← 0 ADO_SELECT (0 SEL_105 0)
...
:WHILE ~FERTIG
    DATEN_105 ← ABS_L ADO_SELECT (1 '' 0)
    FERTIG ← ABS_L > ↑ ρDATEN_105
    ...
:ENDWHILE
```

- in innerer Schleife mit

```
V  ← 1 , ↑ ρDATEN_105
VV ← DATEN_105_DEF ι ⊂'VNR'
(VNR_VON VNR_BIS) ← DATEN_105[V ; VV]
DATEN_103 ← 0 ADO_SELECT SEL_103
```

- Abfrage starten

```
SELECT ...
FROM &CREATOR.CDTB0103 T1
WHERE T1.VNR BETWEEN &VNR_VON AND &VNR_BIS
...
```

- in innerer Schleife mit

```
V  ← 1 , ↑ ρDATEN_105
VV ← DATEN_105_DEF ι ⊂'VNR'
(VNR_VON VNR_BIS) ← DATEN_105[V ; VV]
DATEN_103 ← 0 ADO_SELECT SEL_103
```

- Abfrage starten

```
SELECT ...
FROM &CREATOR.CDTB0103 T1
WHERE T1.VNR BETWEEN &VNR_VON AND &VNR_BIS
...
```

Übersicht über offene Abfragen (**CURSOR**)

Bessere Technik zur Bewältigung großer Datenmengen

Offene Abfragen	sukzessives Einlesen großer Datenmengen über offene Abfragen
Führende Tabelle	einfache offene Abfragen bei führender Tabelle
Gleichberechtigte Tabellen	Komplikationen bei gleichberechtigte Tabellen
Ungeeigneter Index	Komplikationen bei ungeeignetem Index

Übersicht über offene Abfragen (**CURSOR**)

Bessere Technik zur Bewältigung großer Datenmengen

Offene Abfragen	sukzessives Einlesen großer Datenmengen über offene Abfragen
Führende Tabelle	einfache offene Abfragen bei führender Tabelle
Gleichberechtigte Tabellen	Komplikationen bei gleichberechtigte Tabellen
Ungeeigneter Index	Komplikationen bei ungeeignetem Index

Übersicht über offene Abfragen (**CURSOR**)

Bessere Technik zur Bewältigung großer Datenmengen

Offene Abfragen	sukzessives Einlesen großer Datenmengen über offene Abfragen
Führende Tabelle	einfache offene Abfragen bei führender Tabelle
Gleichberechtigte Tabellen	Komplikationen bei gleichberechtigte Tabellen
Ungeeigneter Index	Komplikationen bei ungeeignetem Index

Übersicht über offene Abfragen (**CURSOR**)

Bessere Technik zur Bewältigung großer Datenmengen

Offene Abfragen	sukzessives Einlesen großer Datenmengen über offene Abfragen
Führende Tabelle	einfache offene Abfragen bei führender Tabelle
Gleichberechtigte Tabellen	Komplikationen bei gleichberechtigte Tabellen
Ungeeigneter Index	Komplikationen bei ungeeignetem Index

Übersicht über offene Abfragen (**CURSOR**)

Bessere Technik zur Bewältigung großer Datenmengen

Offene Abfragen	sukzessives Einlesen großer Datenmengen über offene Abfragen
Führende Tabelle	einfache offene Abfragen bei führender Tabelle
Gleichberechtigte Tabellen	Komplikationen bei gleichberechtigte Tabellen
Ungeeigneter Index	Komplikationen bei ungeeignetem Index

ERGO

Übersicht über offene Abfragen (**CURSOR**)

Bessere Technik zur Bewältigung großer Datenmengen

Offene Abfragen	sukzessives Einlesen großer Datenmengen über offene Abfragen
Führende Tabelle	einfache offene Abfragen bei führender Tabelle
Gleichberechtigte Tabellen	Komplikationen bei gleichberechtigte Tabellen
Ungeeigneter Index	Komplikationen bei ungeeignetem Index

Sukzessives Einlesen großer Datenmenge über offene Abfragen.

ERGO

Bessere Technik offene Abfragen (**CURSOR**)

- Ziel Trennung Vorbereitung (**PREPARE**) und Einlesen der Sätze (**FETCH**)
- quasi keine CPU, fast nur I/O
  - genau dann, wenn keine temporäre Tabelle
  - DB2 gibt direkt Ergebnis vor **PREPARE** zurück
  - Selektion relevanter Sätze beim Einlesen
- nur dann möglich, wenn Abfrage über (Haupt-) Index durchführbar
- fachlich zusammengehörende Sätze müssen zusammen bearbeitet werden
  - setzt Sortierung der Sätze voraus
  - deterministisches Ergebnis nur wenn **ORDER BY** benutzt wird
  - fachlich notwendige Sortierung muss mit (einem) Index kompatibel sein

ERGO

Bessere Technik offene Abfragen (**CURSOR**)

- Ziel Trennung Vorbereitung (**PREPARE**) und Einlesen der Sätze (**FETCH**)
- quasi keine CPU, fast nur I/O
  - genau dann, wenn keine temporäre Tabelle
  - DB2 gibt direkt Ergebnis vor **PREPARE** zurück
  - Selektion relevanter Sätze beim Einlesen
- nur dann möglich, wenn Abfrage über (Haupt-) Index durchführbar
- fachlich zusammengehörende Sätze müssen zusammen bearbeitet werden
  - setzt Sortierung der Sätze voraus
  - deterministisches Ergebnis nur wenn **ORDER BY** benutzt wird
  - fachlich notwendige Sortierung muss mit (einem) Index kompatibel sein

ERGO

Bessere Technik offene Abfragen (**CURSOR**)

- Ziel Trennung Vorbereitung (**PREPARE**) und Einlesen der Sätze (**FETCH**)
- quasi keine CPU, fast nur I/O
  - genau dann, wenn keine temporäre Tabelle
  - DB2 gibt direkt Ergebnis vor **PREPARE** zurück
  - Selektion relevanter Sätze beim Einlesen
- nur dann möglich, wenn Abfrage über (Haupt-) Index durchführbar
- fachlich zusammengehörende Sätze müssen zusammen bearbeitet werden
  - setzt Sortierung der Sätze voraus
  - deterministisches Ergebnis nur wenn **ORDER BY** benutzt wird
  - fachlich notwendige Sortierung muss mit (einem) Index kompatibel sein

ERGO

Bessere Technik offene Abfragen (**CURSOR**)

- Ziel Trennung Vorbereitung (**PREPARE**) und Einlesen der Sätze (**FETCH**)
- quasi keine CPU, fast nur I/O
  - genau dann, wenn keine temporäre Tabelle
  - DB2 gibt direkt Ergebnis vor **PREPARE** zurück
  - Selektion relevanter Sätze beim Einlesen
- nur dann möglich, wenn Abfrage über (Haupt-) Index durchführbar
- fachlich zusammengehörende Sätze müssen zusammen bearbeitet werden
  - setzt Sortierung der Sätze voraus
  - deterministisches Ergebnis nur wenn **ORDER BY** benutzt wird
  - fachlich notwendige Sortierung muss mit (einem) Index kompatibel sein

ERGO

Bessere Technik offene Abfragen (**CURSOR**)

- Ziel Trennung Vorbereitung (**PREPARE**) und Einlesen der Sätze (**FETCH**)
- quasi keine CPU, fast nur I/O
  - genau dann, wenn keine temporäre Tabelle
  - DB2 gibt direkt Ergebnis vor **PREPARE** zurück
  - Selektion relevanter Sätze beim Einlesen
- nur dann möglich, wenn Abfrage über (Haupt-) Index durchführbar
- fachlich zusammengehörende Sätze müssen zusammen bearbeitet werden
  - setzt Sortierung der Sätze voraus
  - deterministisches Ergebnis nur wenn **ORDER BY** benutzt wird
  - fachlich notwendige Sortierung muss mit (einem) Index kompatibel sein

ERGO

Bessere Technik offene Abfragen (**CURSOR**)

- Ziel Trennung Vorbereitung (**PREPARE**) und Einlesen der Sätze (**FETCH**)
- quasi keine CPU, fast nur I/O
  - genau dann, wenn keine temporäre Tabelle
  - DB2 gibt direkt Ergebnis vor **PREPARE** zurück
  - Selektion relevanter Sätze beim Einlesen
- nur dann möglich, wenn Abfrage über (Haupt-) Index durchführbar
- fachlich zusammengehörende Sätze müssen zusammen bearbeitet werden
  - setzt Sortierung der Sätze voraus
  - deterministisches Ergebnis nur wenn **ORDER BY** benutzt wird
  - fachlich notwendige Sortierung muss mit (einem) Index kompatibel sein

ERGO

Bessere Technik offene Abfragen (**CURSOR**)

- Ziel Trennung Vorbereitung (**PREPARE**) und Einlesen der Sätze (**FETCH**)
- quasi keine CPU, fast nur I/O
  - genau dann, wenn keine temporäre Tabelle
  - DB2 gibt direkt Ergebnis vor **PREPARE** zurück
  - Selektion relevanter Sätze beim Einlesen
- nur dann möglich, wenn Abfrage über (Haupt-) Index durchführbar
- fachlich zusammengehörende Sätze müssen zusammen bearbeitet werden
  - setzt Sortierung der Sätze voraus
  - deterministisches Ergebnis nur wenn **ORDER BY** benutzt wird
  - fachlich notwendige Sortierung muss mit (einem) Index kompatibel sein

ERGO

Bessere Technik offene Abfragen (**CURSOR**)

- Ziel Trennung Vorbereitung (**PREPARE**) und Einlesen der Sätze (**FETCH**)
- quasi keine CPU, fast nur I/O
  - genau dann, wenn keine temporäre Tabelle
  - DB2 gibt direkt Ergebnis vor **PREPARE** zurück
  - Selektion relevanter Sätze beim Einlesen
- nur dann möglich, wenn Abfrage über (Haupt-) Index durchführbar
- fachlich zusammengehörende Sätze müssen zusammen bearbeitet werden
  - setzt Sortierung der Sätze voraus
  - deterministisches Ergebnis nur wenn **ORDER BY** benutzt wird
  - fachlich notwendige Sortierung muss mit (einem) Index kompatibel sein

ERGO

Bessere Technik offene Abfragen (**CURSOR**)

- Ziel Trennung Vorbereitung (**PREPARE**) und Einlesen der Sätze (**FETCH**)
- quasi keine CPU, fast nur I/O
  - genau dann, wenn keine temporäre Tabelle
  - DB2 gibt direkt Ergebnis vor **PREPARE** zurück
  - Selektion relevanter Sätze beim Einlesen
- nur dann möglich, wenn Abfrage über (Haupt-) Index durchführbar
- fachlich zusammengehörende Sätze müssen zusammen bearbeitet werden
  - setzt Sortierung der Sätze voraus
  - deterministisches Ergebnis nur wenn **ORDER BY** benutzt wird
  - fachlich notwendige Sortierung muss mit (einem) Index kompatibel sein

ERGO

Bessere Technik offene Abfragen (**CURSOR**)

- Ziel Trennung Vorbereitung (**PREPARE**) und Einlesen der Sätze (**FETCH**)
- quasi keine CPU, fast nur I/O
  - genau dann, wenn keine temporäre Tabelle
  - DB2 gibt direkt Ergebnis vor **PREPARE** zurück
  - Selektion relevanter Sätze beim Einlesen
- nur dann möglich, wenn Abfrage über (Haupt-) Index durchführbar
- fachlich zusammengehörende Sätze müssen zusammen bearbeitet werden
  - setzt Sortierung der Sätze voraus
  - deterministisches Ergebnis nur wenn **ORDER BY** benutzt wird
  - fachlich notwendige Sortierung muss mit (einem) Index kompatibel sein

ERGO

Bessere Technik offene Abfragen (**CURSOR**)

- Ziel Trennung Vorbereitung (**PREPARE**) und Einlesen der Sätze (**FETCH**)
- quasi keine CPU, fast nur I/O
  - genau dann, wenn keine temporäre Tabelle
  - DB2 gibt direkt Ergebnis vor **PREPARE** zurück
  - Selektion relevanter Sätze beim Einlesen
- nur dann möglich, wenn Abfrage über (Haupt-) Index durchführbar
- fachlich zusammengehörende Sätze müssen zusammen bearbeitet werden
  - setzt Sortierung der Sätze voraus
  - deterministisches Ergebnis nur wenn **ORDER BY** benutzt wird
  - fachlich notwendige Sortierung muss mit (einem) Index kompatibel sein

ERGO

- Beispiel offene Abfrage
  - Zugriff operative „Tarif-Tabelle“

```
SELECT ...
FROM DB2.KVTB0246 T1
WHERE T1.V_VERW_SYS_SL  = 'KV' -- pro forma
AND   T1.V_ZUSTAND_KZ   = ''   -- pro forma
AND   T1.STATUS_MERKMAL = 0    -- technisch gültig
ORDER BY T1.V_VERW_SYS_SL, T1.VNR, T1.V_ZUSTAND_KZ
       , T1.STATUS_MERKMAL, T1.POS_LNR, T1.TA_ID
       , T1.GUELTIG_AB DESC, T1.HIST_LNR DESC
                             -- exakt nach Index
```

- Beispiel offene Abfrage
  - Zugriff operative „Tarif-Tabelle“

```
SELECT ...
FROM DB2.KVTB0246 T1
WHERE T1.V_VERW_SYS_SL  = 'KV' -- pro forma
AND   T1.V_ZUSTAND_KZ   = ''   -- pro forma
AND   T1.STATUS_MERKMAL = 0    -- technisch gültig
ORDER BY T1.V_VERW_SYS_SL, T1.VNR, T1.V_ZUSTAND_KZ
       , T1.STATUS_MERKMAL, T1.POS_LNR, T1.TA_ID
       , T1.GUELTIG_AB DESC, T1.HIST_LNR DESC
                             -- exakt nach Index
```

Beispiel technische Hindernisse offene Abfrage

- **RUNSTATS** müssen stimmen / angepasst werden
- aus inhaltlichen Gründen bekannt

$$\texttt{STATUS\_MERKMAL} = 0 \quad \gtrapprox \quad 90\% \text{ Bestand}$$

- DB2 geht von Gleichverteilung auf $[0, \approx 100]$ aus
- dies führt zu **TABLE SCAN**
- Sonder-**RUNSTATS** (durch DBA) notwendig

Beispiel technische Hindernisse offene Abfrage

- **RUNSTATS** müssen stimmen / angepasst werden
- aus inhaltlichen Gründen bekannt

$$\texttt{STATUS\_MERKMAL} = 0 \quad \gtrapprox \quad 90\% \text{ Bestand}$$

- DB2 geht von Gleichverteilung auf $[0, \approx 100]$ aus
- dies führt zu **TABLE SCAN**
- Sonder-**RUNSTATS** (durch DBA) notwendig

Beispiel technische Hindernisse offene Abfrage

- **RUNSTATS** müssen stimmen / angepasst werden
- aus inhaltlichen Gründen bekannt

$$\texttt{STATUS\_MERKMAL} = 0 \quad \gtrapprox \quad 90\%\ \text{Bestand}$$

- DB2 geht von Gleichverteilung auf $[0, \approx 100]$ aus
- dies führt zu **TABLE SCAN**
- Sonder-**RUNSTATS** (durch DBA) notwendig

Beispiel technische Hindernisse offene Abfrage

- **RUNSTATS** müssen stimmen / angepasst werden
- aus inhaltlichen Gründen bekannt

$$\texttt{STATUS\_MERKMAL} = 0 \qquad \gtrapprox \qquad 90\%\ \text{Bestand}$$

- DB2 geht von Gleichverteilung auf $[0, \approx 100]$ aus
- dies führt zu **TABLE SCAN**
- Sonder-**RUNSTATS** (durch DBA) notwendig

Beispiel technische Hindernisse offene Abfrage

- **RUNSTATS** müssen stimmen / angepasst werden
- aus inhaltlichen Gründen bekannt

$$\texttt{STATUS\_MERKMAL} = 0 \qquad \gtrapprox \qquad 90\%\ \text{Bestand}$$

- DB2 geht von Gleichverteilung auf $[0, \approx 100]$ aus
- dies führt zu **TABLE SCAN**
- Sonder-**RUNSTATS** (durch DBA) notwendig

Beispiel technische Hindernisse offene Abfrage

- **RUNSTATS** müssen stimmen / angepasst werden
- aus inhaltlichen Gründen bekannt

$$\texttt{STATUS\_MERKMAL} = 0 \qquad \gtrapprox \qquad 90\%\ \text{Bestand}$$

- DB2 geht von Gleichverteilung auf $[0, \approx 100]$ aus
- dies führt zu **TABLE SCAN**
- Sonder-**RUNSTATS** (durch DBA) notwendig

- Beispiel Bestandsmigration
  - bestimme (fachlich) führende Tabelle
  - wähle maximale Anzahl zu bearbeitenden Sätze ABS_L aus
  - Verwaltung etwas komplexer
  - Initialisierung nach Schema

```
FERTIG_346 ← FERTIG_342 ← FERTIG_348 ← ... ← 0
DATEN_346_DEF ←
 (0 RCSNR_346 CONNR) ADO_SELECT (0 SEL_346 0)
DATEN_242_DEF ←
 (0 RCSNR_242 CONNR) ADO_SELECT (0 SEL_242 0)
DATEN_348_DEF ←
 (0 RCSNR_348 CONNR) ADO_SELECT (0 SEL_348 0)
...
```

  - Randfälle (jeweiliges Ende Tabelle) zu beachten

- Beispiel Bestandsmigration
  - bestimme (fachlich) führende Tabelle
  - wähle maximale Anzahl zu bearbeitenden Sätze ABS_L aus
  - Verwaltung etwas komplexer
  - Initialisierung nach Schema

```
FERTIG_346 ← FERTIG_342 ← FERTIG_348 ← ... ← 0
DATEN_346_DEF ←
 (0 RCSNR_346 CONNR) ADO_SELECT (0 SEL_346 0)
DATEN_242_DEF ←
 (0 RCSNR_242 CONNR) ADO_SELECT (0 SEL_242 0)
DATEN_348_DEF ←
 (0 RCSNR_348 CONNR) ADO_SELECT (0 SEL_348 0)
...
```

  - Randfälle (jeweiliges Ende Tabelle) zu beachten

- Beispiel Bestandsmigration
  - bestimme (fachlich) führende Tabelle
  - wähle maximale Anzahl zu bearbeitenden Sätze `ABS_L` aus
  - Verwaltung etwas komplexer
  - Initialisierung nach Schema

```
FERTIG_346 ← FERTIG_342 ← FERTIG_348 ← ... ← 0
DATEN_346_DEF ←
 (0 RCSNR_346 CONNR) ADO_SELECT (0 SEL_346 0)
DATEN_242_DEF ←
 (0 RCSNR_242 CONNR) ADO_SELECT (0 SEL_242 0)
DATEN_348_DEF ←
 (0 RCSNR_348 CONNR) ADO_SELECT (0 SEL_348 0)
...
```

  - Randfälle (jeweiliges Ende Tabelle) zu beachten

- Beispiel Bestandsmigration
  - bestimme (fachlich) führende Tabelle
  - wähle maximale Anzahl zu bearbeitenden Sätze `ABS_L` aus
  - Verwaltung etwas komplexer
  - Initialisierung nach Schema

```
FERTIG_346 ← FERTIG_342 ← FERTIG_348 ← ... ← 0
DATEN_346_DEF ←
 (0 RCSNR_346 CONNR) ADO_SELECT (0 SEL_346 0)
DATEN_242_DEF ←
 (0 RCSNR_242 CONNR) ADO_SELECT (0 SEL_242 0)
DATEN_348_DEF ←
 (0 RCSNR_348 CONNR) ADO_SELECT (0 SEL_348 0)
...
```

  - Randfälle (jeweiliges Ende Tabelle) zu beachten

- Beispiel Bestandsmigration
  - bestimme (fachlich) führende Tabelle
  - wähle maximale Anzahl zu bearbeitenden Sätze `ABS_L` aus
  - Verwaltung etwas komplexer
  - Initialisierung nach Schema

```
FERTIG_346 ← FERTIG_342 ← FERTIG_348 ← ... ← 0
DATEN_346_DEF ←
 (0 RCSNR_346 CONNR) ADO_SELECT (0 SEL_346 0)
DATEN_242_DEF ←
 (0 RCSNR_242 CONNR) ADO_SELECT (0 SEL_242 0)
DATEN_348_DEF ←
 (0 RCSNR_348 CONNR) ADO_SELECT (0 SEL_348 0)
...
```

  - Randfälle (jeweiliges Ende Tabelle) zu beachten

- Beispiel Bestandsmigration
  - bestimme (fachlich) führende Tabelle
  - wähle maximale Anzahl zu bearbeitenden Sätze `ABS_L` aus
  - Verwaltung etwas komplexer
  - Initialisierung nach Schema

```
FERTIG_346 ← FERTIG_342 ← FERTIG_348 ← ... ← 0
DATEN_346_DEF ←
 (0 RCSNR_346 CONNR) ADO_SELECT (0 SEL_346 0)
DATEN_242_DEF ←
 (0 RCSNR_242 CONNR) ADO_SELECT (0 SEL_242 0)
DATEN_348_DEF ←
 (0 RCSNR_348 CONNR) ADO_SELECT (0 SEL_348 0)
...
```

  - Randfälle (jeweiliges Ende Tabelle) zu beachten

Schema Einlesen führende Tabelle

```
DATEN_346 ←
 (ABS_L RCSNR_346 CONNR) ADO_SELECT (1 '' 0)
FERTIG_346 ← ABS_L > ↑ ρDATEN_346
DATEN_REST_346 ← DATEN_REST_346 ,[1] DATEN_346
V ← DATEN_REST_346[;DATEN_346_DEF ⍳ ⊂'VNR']
VNR_MAX ← V[↑ ρDATEN_REST_346]
I ← VNR_MAX > V
DATEN_346      ←   I  ≠ DATEN_REST_346
DATEN_REST_346 ← (~I) ≠ DATEN_REST_346
...
```

Schema Einlesen restliche Tabellen

```
:WHILE ~ FERTIG_242
:ANDIF VNR_MAX ≥ DATEN_242_REST[↑ ⍴DATEN_242_REST ;
 DATEN_242_DEF ⍳ ⊂'VNR']
    DATEN_242 ←
 (ABS_L RCSNR_242 CONNR) ADO_SELECT (1 '' 0)
    FERTIG_242 ← ABS_L > ↑ ⍴DATEN_242
    DATEN_REST_242 ← DATEN_REST_242 ,[1] DATEN_242
:ENDWHILE
V ← DATEN_REST_242[;DATEN_242_DEF ⍳ ⊂'VNR']
I ← VNR_MAX > V
DATEN_242      ←   I  ≠ DATEN_REST_242
DATEN_REST_242 ← (~I) ≠ DATEN_REST_242
...
```

- Beispiel Rechnungsgrundlagen-Migration
  - Komplikation da keine führende Tabelle
  - zwei gleichberechtigte, stark unterschiedlich besetzte Tabellen
  - Einlese-Schema für jede Tabelle

```
DATEN_x ←
 (ABS_L RCSNR_x CONNR) ADO_SELECT (0 SEL_x 0)
...
:IF    ~ FERTIG_x
:ANDIF ABSL > ↑ ρDATEN_x_REST
    DATEN_x ←
 (ABS_L RCSNR_x CONNR) ADO_SELECT (1 '' 0)
    ...
:ENDIF
```

- Beispiel Rechnungsgrundlagen-Migration
  - Komplikation da keine führende Tabelle
  - zwei gleichberechtigte, stark unterschiedlich besetzte Tabellen
  - Einlese-Schema für jede Tabelle

```
DATEN_x ←
 (ABS_L RCSNR_x CONNR) ADO_SELECT (0 SEL_x 0)
...
:IF    ~ FERTIG_x
:ANDIF ABSL > ↑ ρDATEN_x_REST
    DATEN_x ←
 (ABS_L RCSNR_x CONNR) ADO_SELECT (1 '' 0)
    ...
:ENDIF
```

- Beispiel Rechnungsgrundlagen-Migration
  - Komplikation da keine führende Tabelle
  - zwei gleichberechtigte, stark unterschiedlich besetzte Tabellen
  - Einlese-Schema für jede Tabelle

```
DATEN_x ←
 (ABS_L RCSNR_x CONNR) ADO_SELECT (0 SEL_x 0)
...
:IF    ~ FERTIG_x
:ANDIF ABSL > ↑ ρDATEN_x_REST
    DATEN_x ←
 (ABS_L RCSNR_x CONNR) ADO_SELECT (1 '' 0)
    ...
:ENDIF
```

- Beispiel Rechnungsgrundlagen-Migration
  - Komplikation da keine führende Tabelle
  - zwei gleichberechtigte, stark unterschiedlich besetzte Tabellen
  - Einlese-Schema für jede Tabelle

```
DATEN_x ←
 (ABS_L RCSNR_x CONNR) ADO_SELECT (0 SEL_x 0)
...
:IF     ~ FERTIG_x
:ANDIF ABSL > ↑ ρDATEN_x_REST
    DATEN_x ←
 (ABS_L RCSNR_x CONNR) ADO_SELECT (1 '' 0)
    ...
:ENDIF
```

Synchronisierung

```
:IF ~ ^ / FERTIG_CORE FERTIG_DART
    V  ← (1 + 0 = ↑ ρDATEN_CORE_REST)  ⊃
 (DATEN_CORE_REST[;DATEN_CORE_DEF ⍳ ⊂'TA_ID'])  (,0)
    VV ← (1 + 0 = ↑ ρDATEN_DART_REST)  ⊃
 (DATEN_DART_REST[;DATEN_DART_DEF ⍳ ⊂'TA_ID'])  (,0)
    MAX_TAID ← V[↑ ρV] , VV[↑ ρVV] ⍝ maximale TAIDs
    MAX_TAID ← (1 + ∨ / FERTIG_CORE FERTIG_DART)  ⊃
 (⌊ / MAX_TAID)
 (⌈  /  (~ FERTIG_CORE FERTIG_DART)  ×  MAX_TAID)
 ⍝ (minimale oder efektiv maximale) maximale TAID
...
:ELSE
...
:ENDIF
```

- Probleme mit offenem cursor bei nicht geeigneter Indizierung
  - dispositiv meistens Hauptindex `GES_SL`, `VNR`, teilweise weitere
  - `GES_SL` technisches Merkmal, keine fachliche Bedeutung
  - Sortierung nach `VNR` problematisch
- z.B. alle Kombinationen bestimmen
  - gesamte technische und fachliche Historie
  - wenige Felder aus **CDTB0105**, sonst reine Kombinatorik in APL
  - brauche (die) drei Werte von `GES_SL`
  - Kontrolle über Anzahl Sätze wichtig, sehr große Zwischenergebnisse
  - Indizes

    ```
    GES_SL, VNR, POS_LNR, ...
    VNR, POS_LNR, ...
    ...
    ```

- Probleme mit offenem cursor bei nicht geeigneter Indizierung
  - dispositiv meistens Hauptindex `GES_SL`, `VNR`, teilweise weitere
  - `GES_SL` technisches Merkmal, keine fachliche Bedeutung
  - Sortierung nach `VNR` problematisch
- z.B. alle Kombinationen bestimmen
  - gesamte technische und fachliche Historie
  - wenige Felder aus **CDTB0105**, sonst reine Kombinatorik in APL
  - brauche (die) drei Werte von `GES_SL`
  - Kontrolle über Anzahl Sätze wichtig, sehr große Zwischenergebnisse
  - Indizes

    ```
    GES_SL, VNR, POS_LNR, ...
    VNR, POS_LNR, ...
    ...
    ```

- Probleme mit offenem cursor bei nicht geeigneter Indizierung
  - dispositiv meistens Hauptindex `GES_SL`, `VNR`, teilweise weitere
  - `GES_SL` technisches Merkmal, keine fachliche Bedeutung
  - Sortierung nach `VNR` problematisch
- z.B. alle Kombinationen bestimmen
  - gesamte technische und fachliche Historie
  - wenige Felder aus **CDTB0105**, sonst reine Kombinatorik in APL
  - brauche (die) drei Werte von `GES_SL`
  - Kontrolle über Anzahl Sätze wichtig, sehr große Zwischenergebnisse
  - Indizes

    ```
    GES_SL, VNR, POS_LNR, ...
    VNR, POS_LNR, ...
    ...
    ```

- Probleme mit offenem cursor bei nicht geeigneter Indizierung
  - dispositiv meistens Hauptindex `GES_SL`, `VNR`, teilweise weitere
  - `GES_SL` technisches Merkmal, keine fachliche Bedeutung
  - Sortierung nach `VNR` problematisch
- z.B. alle Kombinationen bestimmen
  - gesamte technische und fachliche Historie
  - wenige Felder aus **CDTB0105**, sonst reine Kombinatorik in APL
  - brauche (die) drei Werte von `GES_SL`
  - Kontrolle über Anzahl Sätze wichtig, sehr große Zwischenergebnisse
  - Indizes

    ```
    GES_SL, VNR, POS_LNR, ...
    VNR, POS_LNR, ...
    ...
    ```

- Probleme mit offenem cursor bei nicht geeigneter Indizierung
  - dispositiv meistens Hauptindex `GES_SL`, `VNR`, teilweise weitere
  - `GES_SL` technisches Merkmal, keine fachliche Bedeutung
  - Sortierung nach `VNR` problematisch
- z.B. alle Kombinationen bestimmen
  - gesamte technische und fachliche Historie
  - wenige Felder aus **CDTB0105**, sonst reine Kombinatorik in APL
  - brauche (die) drei Werte von `GES_SL`
  - Kontrolle über Anzahl Sätze wichtig, sehr große Zwischenergebnisse
  - Indizes

    ```
    GES_SL, VNR, POS_LNR, ...
    VNR, POS_LNR, ...
    ...
    ```

- Probleme mit offenem cursor bei nicht geeigneter Indizierung
  - dispositiv meistens Hauptindex `GES_SL`, `VNR`, teilweise weitere
  - `GES_SL` technisches Merkmal, keine fachliche Bedeutung
  - Sortierung nach `VNR` problematisch
- z.B. alle Kombinationen bestimmen
  - gesamte technische und fachliche Historie
  - wenige Felder aus **CDTB0105**, sonst reine Kombinatorik in APL
  - brauche (die) drei Werte von `GES_SL`
  - Kontrolle über Anzahl Sätze wichtig, sehr große Zwischenergebnisse
  - Indizes

    ```
    GES_SL, VNR, POS_LNR, ...
    VNR, POS_LNR, ...
    ...
    ```

- Probleme mit offenem cursor bei nicht geeigneter Indizierung
  - dispositiv meistens Hauptindex `GES_SL`, `VNR`, teilweise weitere
  - `GES_SL` technisches Merkmal, keine fachliche Bedeutung
  - Sortierung nach `VNR` problematisch
- z.B. alle Kombinationen bestimmen
  - gesamte technische und fachliche Historie
  - wenige Felder aus **CDTB0105**, sonst reine Kombinatorik in APL
  - brauche (die) drei Werte von `GES_SL`
  - Kontrolle über Anzahl Sätze wichtig, sehr große Zwischenergebnisse
  - Indizes

```
GES_SL, VNR, POS_LNR, ...
VNR, POS_LNR, ...
...
```

- Probleme mit offenem cursor bei nicht geeigneter Indizierung
  - dispositiv meistens Hauptindex `GES_SL`, `VNR`, teilweise weitere
  - `GES_SL` technisches Merkmal, keine fachliche Bedeutung
  - Sortierung nach `VNR` problematisch
- z.B. alle Kombinationen bestimmen
  - gesamte technische und fachliche Historie
  - wenige Felder aus **CDTB0105**, sonst reine Kombinatorik in APL
  - brauche (die) drei Werte von `GES_SL`
  - Kontrolle über Anzahl Sätze wichtig, sehr große Zwischenergebnisse
  - Indizes

    ```
    GES_SL, VNR, POS_LNR, ...
    VNR, POS_LNR, ...
    ...
    ```

- Probleme mit offenem cursor bei nicht geeigneter Indizierung
  - dispositiv meistens Hauptindex `GES_SL`, `VNR`, teilweise weitere
  - `GES_SL` technisches Merkmal, keine fachliche Bedeutung
  - Sortierung nach `VNR` problematisch
- z.B. alle Kombinationen bestimmen
  - gesamte technische und fachliche Historie
  - wenige Felder aus **CDTB0105**, sonst reine Kombinatorik in APL
  - brauche (die) drei Werte von `GES_SL`
  - Kontrolle über Anzahl Sätze wichtig, sehr große Zwischenergebnisse
  - Indizes

    ```
    GES_SL, VNR, POS_LNR, ...
    VNR, POS_LNR, ...
    ...
    ```

- Probleme mit offenem cursor bei nicht geeigneter Indizierung
  - dispositiv meistens Hauptindex `GES_SL`, `VNR`, teilweise weitere
  - `GES_SL` technisches Merkmal, keine fachliche Bedeutung
  - Sortierung nach `VNR` problematisch
- z.B. alle Kombinationen bestimmen
  - gesamte technische und fachliche Historie
  - wenige Felder aus **CDTB0105**, sonst reine Kombinatorik in APL
  - brauche (die) drei Werte von `GES_SL`
  - Kontrolle über Anzahl Sätze wichtig, sehr große Zwischenergebnisse
  - Indizes

```
GES_SL, VNR, POS_LNR, ...
VNR, POS_LNR, ...
...
```

- Erster Versuch

```
SELECT INTEGER(T1.VNR) "VNR", ...
FROM K$DB2FTB.CDTB0105 T1
WHERE T1.GUELTIG_AB < T1.UNGUELTIG_AB
ORDER BY T1.VNR, T1.POS_LNR, ...
```

- Index greift, aber
  - **PREPARE** 15min
  - dann Absturz wegen **WORK TABLE**
- also temporäre Tabelle im Hintergrund

- Erster Versuch

```
SELECT INTEGER(T1.VNR) "VNR", ...
FROM K$DB2FTB.CDTB0105 T1
WHERE T1.GUELTIG_AB < T1.UNGUELTIG_AB
ORDER BY T1.VNR, T1.POS_LNR, ...
```

- Index greift, aber
  - **PREPARE** 15min
  - dann Absturz wegen **WORK TABLE**
- also temporäre Tabelle im Hintergrund

- Erster Versuch

```
SELECT INTEGER(T1.VNR) "VNR", ...
FROM K$DB2FTB.CDTB0105 T1
WHERE T1.GUELTIG_AB < T1.UNGUELTIG_AB
ORDER BY T1.VNR, T1.POS_LNR, ...
```

- Index greift, aber
  - **PREPARE** 15min
  - dann Absturz wegen **WORK TABLE**
- also temporäre Tabelle im Hintergrund

- Erster Versuch

```
SELECT INTEGER(T1.VNR) "VNR", ...
FROM K$DB2FTB.CDTB0105 T1
WHERE T1.GUELTIG_AB < T1.UNGUELTIG_AB
ORDER BY T1.VNR, T1.POS_LNR, ...
```

- Index greift, aber
  - **PREPARE** 15min
  - dann Absturz wegen **WORK TABLE**
- also temporäre Tabelle im Hintergrund

- Erster Versuch

```
SELECT INTEGER(T1.VNR) "VNR", ...
FROM K$DB2FTB.CDTB0105 T1
WHERE T1.GUELTIG_AB < T1.UNGUELTIG_AB
ORDER BY T1.VNR, T1.POS_LNR, ...
```

- Index greift, aber
  - **PREPARE** 15min
  - dann Absturz wegen **WORK TABLE**
- also temporäre Tabelle im Hintergrund

- Zweiter Versuch

```
SELECT INTEGER(T1.VNR) "VNR", ...
FROM K$DB2FTB.CDTB0105 T1
WHERE GES_SL = '0501'
AND T1.GUELTIG_AB < T1.UNGUELTIG_AB
UNION ALL
...
WHERE GES_SL = '0015'
...
UNION ALL
...
WHERE GES_SL = '0520'
...
ORDER BY T1.VNR, T1.POS_LNR, ...
```

- Index greift nicht, TABLE SCAN

- Zweiter Versuch

```
SELECT INTEGER(T1.VNR) "VNR", ...
FROM K$DB2FTB.CDTB0105 T1
WHERE GES_SL = '0501'
AND T1.GUELTIG_AB < T1.UNGUELTIG_AB
UNION ALL
...
WHERE GES_SL = '0015'
...
UNION ALL
...
WHERE GES_SL = '0520'
...
ORDER BY T1.VNR, T1.POS_LNR, ...
```

- Index greift nicht, **TABLE SCAN**

- Dritter Versuch, zwei Abfragen

```
SELECT INTEGER(T1.VNR) "VNR", ...
FROM K$DB2FTB.CDTB0105 T1
WHERE GES_SL = '0501'
AND T1.GUELTIG_AB < T1.UNGUELTIG_AB
ORDER BY T1.VNR, T1.POS_LNR, ...
```

- Index greift nicht

- Dritter Versuch, zwei Abfragen

```
SELECT INTEGER(T1.VNR) "VNR", ...
FROM K$DB2FTB.CDTB0105 T1
WHERE GES_SL = '0501'
AND T1.GUELTIG_AB < T1.UNGUELTIG_AB
ORDER BY T1.VNR, T1.POS_LNR, ...
```

- Index greift nicht

- Vierter Versuch, zwei Abfragen

```
SELECT INTEGER(T1.VNR) "VNR", ...
FROM K$DB2FTB.CDTB0105 T1
WHERE GES_SL = '0501'
AND T1.GUELTIG_AB < T1.UNGUELTIG_AB
ORDER BY T1.GES_SL, "VNR", ...
```

- Index greift nicht, weil DB2-Funktion auf `T1.VNR`

- Vierter Versuch, zwei Abfragen

```
SELECT INTEGER(T1.VNR) "VNR", ...
FROM K$DB2FTB.CDTB0105 T1
WHERE GES_SL = '0501'
AND T1.GUELTIG_AB < T1.UNGUELTIG_AB
ORDER BY T1.GES_SL, "VNR", ...
```

- Index greift nicht, weil DB2-Funktion auf `T1.VNR`

- Fünfter Versuch, zwei Abfragen

```
SELECT INTEGER(T1.VNR) "VNR", ...
FROM K$DB2FTB.CDTB0105 T1
WHERE GES_SL = '0501'
AND T1.GUELTIG_AB < T1.UNGUELTIG_AB
ORDER BY T1.GES_SL, T1.VNR, ...
```

- Index greift, **CURSOR** öffnet direkt

- Fünfter Versuch, zwei Abfragen

```
SELECT INTEGER(T1.VNR) "VNR", ...
FROM K$DB2FTB.CDTB0105 T1
WHERE GES_SL = '0501'
AND T1.GUELTIG_AB < T1.UNGUELTIG_AB
ORDER BY T1.GES_SL, T1.VNR, ...
```

- Index greift, **CURSOR** öffnet direkt

Effektiv *n* gleichberechtigte, zusammenzufügende Tabellen

- Einlesen

```
:FOR x :IN ⍳n
    :IF    ~ FERTIG_x
    :ANDIF ABSL > ↑ ⍴DATEN_x_REST
        DATEN_x ←
 (ABS_L RCSNR_x CONNR) ADO_SELECT (1 '' 0)
        ...
        MAX_x ← ...
    :ENDIF
:ENDFOR
```

- Synchronisierung
  - mit `MAX ← ⌈ / MAX_1 MAX_2 ...` inhaltlich falsch
  - mit `MAX ← ⌊ / MAX_1 MAX_2 ...` korrekt, aber
  - mit (nur) einem `DATEN_REST`
  - Kontrolle über Anzahl Sätze nicht gut genug (`WS FULL`!)
- zusätzlich
  - heuristische Methoden (`c × ABS_L` für jeweilige Daten) oder
  - weiteres `DATEN_REST` sowie `DATEN` und Vergleich

```
V ← REST[;DATEN_DEF ⍳ ⊂'VNR']
I ← V[ABS_BEARB_L ⌊ ↑ ⍴REST] ≥ V
DATEN ← ( I) ≠ REST
REST  ← (~I) ≠ REST
```

- Synchronisierung
  - mit `MAX ← ⌈ / MAX_1 MAX_2 ...` inhaltlich falsch
  - mit `MAX ← ⌊ / MAX_1 MAX_2 ...` korrekt, aber
  - mit (nur) einem `DATEN_REST`
  - Kontrolle über Anzahl Sätze nicht gut genug (`WS FULL`!)
- zusätzlich
  - heuristische Methoden (`c × ABS_L` für jeweilige Daten) oder
  - weiteres `DATEN_REST` sowie `DATEN` und Vergleich

```
V ← REST[;DATEN_DEF ⍳ ⊂'VNR']
I ← V[ABS_BEARB_L ⌊ ↑ ⍴REST] ≥ V
DATEN ← ( I) ≠ REST
REST  ← (~I) ≠ REST
```

- Synchronisierung
  - mit `MAX ← ⌈ / MAX_1 MAX_2 ...` inhaltlich falsch
  - mit `MAX ← ⌊ / MAX_1 MAX_2 ...` korrekt, aber
  - mit (nur) einem `DATEN_REST`
  - Kontrolle über Anzahl Sätze nicht gut genug (`WS FULL`!)
- zusätzlich
  - heuristische Methoden (`c × ABS_L` für jeweilige Daten) oder
  - weiteres `DATEN_REST` sowie `DATEN` und Vergleich

```
V ← REST[;DATEN_DEF ⍳ ⊂'VNR']
I ← V[ABS_BEARB_L ⌊ ↑ ⍴REST] ≥ V
DATEN ← ( I) ≠ REST
REST  ← (~I) ≠ REST
```

- Synchronisierung
  - mit `MAX ← ⌈ / MAX_1 MAX_2 ...` inhaltlich falsch
  - mit `MAX ← ⌊ / MAX_1 MAX_2 ...` korrekt, aber
  - mit (nur) einem `DATEN_REST`
  - Kontrolle über Anzahl Sätze nicht gut genug (`WS FULL`!)
- zusätzlich
  - heuristische Methoden (`c × ABS_L` für jeweilige Daten) oder
  - weiteres `DATEN_REST` sowie `DATEN` und Vergleich

```
V ← REST[;DATEN_DEF ⍳ ⊂'VNR']
I ← V[ABS_BEARB_L ⌊ ↑ ⍴REST] ≥ V
DATEN ← ( I) ≠ REST
REST  ← (~I) ≠ REST
```

- Synchronisierung
  - mit `MAX ← ⌈ / MAX_1 MAX_2 ...` inhaltlich falsch
  - mit `MAX ← ⌊ / MAX_1 MAX_2 ...` korrekt, aber
  - mit (nur) einem `DATEN_REST`
  - Kontrolle über Anzahl Sätze nicht gut genug (`WS FULL`!)
- zusätzlich
  - heuristische Methoden (`c × ABS_L` für jeweilige Daten) oder
  - weiteres `DATEN_REST` sowie `DATEN` und Vergleich

```
V ← REST[;DATEN_DEF ⍳ ⊂'VNR']
I ← V[ABS_BEARB_L ⌊ ↑ ⍴REST] ≥ V
DATEN ← ( I) ≠ REST
REST  ← (~I) ≠ REST
```

- Synchronisierung
  - mit `MAX ← ⌈ / MAX_1 MAX_2 ...` inhaltlich falsch
  - mit `MAX ← ⌊ / MAX_1 MAX_2 ...` korrekt, aber
  - mit (nur) einem `DATEN_REST`
  - Kontrolle über Anzahl Sätze nicht gut genug (`WS FULL`!)
- zusätzlich
  - heuristische Methoden (`c × ABS_L` für jeweilige Daten) oder
  - weiteres `DATEN_REST` sowie `DATEN` und Vergleich

```
V ← REST[;DATEN_DEF ⍳ ⊂'VNR']
I ← V[ABS_BEARB_L ⌊ ↑ ⍴REST] ≥ V
DATEN ← ( I) ≠ REST
REST  ← (~I) ≠ REST
```

- Synchronisierung
  - mit `MAX ← ⌈ / MAX_1 MAX_2 ...` inhaltlich falsch
  - mit `MAX ← ⌊ / MAX_1 MAX_2 ...` korrekt, aber
  - mit (nur) einem `DATEN_REST`
  - Kontrolle über Anzahl Sätze nicht gut genug (`WS FULL`!)
- zusätzlich
  - heuristische Methoden (`c × ABS_L` für jeweilige Daten) oder
  - weiteres `DATEN_REST` sowie `DATEN` und Vergleich

```
V ← REST[;DATEN_DEF ⍳ ⊂'VNR']
I ← V[ABS_BEARB_L ⌊ ↑ ⍴REST] ≥ V
DATEN ← ( I) ≠ REST
REST  ← (~I) ≠ REST
```

- Synchronisierung
  - mit `MAX ← ⌈ / MAX_1 MAX_2 ...` inhaltlich falsch
  - mit `MAX ← ⌊ / MAX_1 MAX_2 ...` korrekt, aber
  - mit (nur) einem `DATEN_REST`
  - Kontrolle über Anzahl Sätze nicht gut genug (`WS FULL`!)
- zusätzlich
  - heuristische Methoden (`c × ABS_L` für jeweilige Daten) oder
  - weiteres `DATEN_REST` sowie `DATEN` und Vergleich

```
V ← REST[;DATEN_DEF ⍳ ⊂'VNR']
I ← V[ABS_BEARB_L ⌊ ↑ ⍴REST] ≥ V
DATEN ← ( I) ⌿ REST
REST  ← (~I) ⌿ REST
```

Offene Frage / nächstes to do

- Nummernkreis-Verfahren benutzt solche Tabellen
- Verfahren durch offene Abfragen ersetzen?
- nur führende Tabelle über **CURSOR** oder alle?

Offene Frage / nächstes to do

- Nummernkreis-Verfahren benutzt solche Tabellen
- Verfahren durch offene Abfragen ersetzen?
- nur führende Tabelle über CURSOR oder alle?

Offene Frage / nächstes to do

- Nummernkreis-Verfahren benutzt solche Tabellen
- Verfahren durch offene Abfragen ersetzen?
- nur führende Tabelle über **CURSOR** oder alle?

Schlußwort

Optimierung von lesenden Zugriffen:

- egal falls Datenmenge klein oder Speicherplatz (beliebig) groß
- egal falls Simulation sehr einfach oder Laufzeit egal
- trotzdem in vielen Fälle praktisch sehr wichtig
- damit Basis für die meisten komplexeren Simulationen

ERGO

Schlußwort

Optimierung von lesenden Zugriffen:

- egal falls Datenmenge klein oder Speicherplatz (beliebig) groß
- egal falls Simulation sehr einfach oder Laufzeit egal
- trotzdem in vielen Fälle praktisch sehr wichtig
- damit Basis für die meisten komplexeren Simulationen

ERGO

Schlußwort

Optimierung von lesenden Zugriffen:

- egal falls Datenmenge klein oder Speicherplatz (beliebig) groß
- egal falls Simulation sehr einfach oder Laufzeit egal
- trotzdem in vielen Fälle praktisch sehr wichtig
- damit Basis für die meisten komplexeren Simulationen

ERGO

Schlußwort

Optimierung von lesenden Zugriffen:

- egal falls Datenmenge klein oder Speicherplatz (beliebig) groß
- egal falls Simulation sehr einfach oder Laufzeit egal
- trotzdem in vielen Fälle praktisch sehr wichtig
- damit Basis für die meisten komplexeren Simulationen

ERGO

Schlußwort

Optimierung von lesenden Zugriffen:

- egal falls Datenmenge klein oder Speicherplatz (beliebig) groß
- egal falls Simulation sehr einfach oder Laufzeit egal
- trotzdem in vielen Fälle praktisch sehr wichtig
- damit Basis für die meisten komplexeren Simulationen

ERGO

Schlußwort

Optimierung von lesenden Zugriffen:

- egal falls Datenmenge klein oder Speicherplatz (beliebig) groß
- egal falls Simulation sehr einfach oder Laufzeit egal
- trotzdem in vielen Fälle praktisch sehr wichtig
- damit Basis für die meisten komplexeren Simulationen

◂ begin

ERGO