



PowerBuilder User Group Germany e.V.

PBUGG Meeting 2026 · Berlin · 19.–20. Oktober 2026

Performance Optimierung mit AI

Ein Erfahrungsbericht

Thomas Enzenberger · Power People IT-Services GmbH

Einführung in Claude

Was ist Claude – und wofür lässt es sich einsetzen?

- Claude ist die KI-Assistenten-Familie von Anthropic – zugänglich über `claude.ai`, API, Claude Code und Integrationen (z. B. `kw/OS`).
- Versteht natürliche Sprache, Quellcode, Bilder und lange Dokumente – bis zu 1 Mio. Token Kontext gleichzeitig.
- Kann eigenständig Werkzeuge nutzen, Dateien bearbeiten und mehrstufige Aufgaben planen und ausführen („Agentic AI“).
- Modelle unterscheiden sich in Qualität, Geschwindigkeit und Kosten – die passende Wahl hängt von der Aufgabe ab.

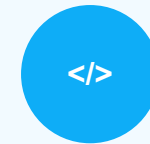
Voraussetzungen für PowerBuilder-Code

- **Quellcode als Text, nicht nur binär.** Erst ab PowerBuilder 2025 liegen `.sr*`-Dateien durchgängig lesbar vor – vorher steckte der Code nur im binären `.pbl`.
- **PowerBuilder-Wissen als Regeln oder Skills.** Syntax-Eigenheiten und Teamkonventionen gehören in `CLAUDE.md`-Regeln oder Skills – generisches Training deckt eine Nischensprache nur oberflächlich ab.
- **Encoding beachten.** `.srw/.srf/.sru` sind UTF-8 mit BOM – naive Edits zerstören Umlaute.
- **Der Mensch kompiliert und testet.** Claude kompiliert nicht selbst – die PB-IDE bleibt die letzte Instanz.



Sprache & Analyse

Texte verstehen, zusammenfassen, übersetzen, bewerten



Code

Schreiben, refactoren, erklären, Fehler analysieren



Agents & Tools

Werkzeuge aufrufen, Dateien bearbeiten, Aufgaben ausführen

Das richtige Modell wählen

Claude-Modelle im Überblick – und kosteneffiziente Alternativen für Alltags-/Volumenlast



Opus 5
Höchste Qualität

Komplexe Analysen, kritische Aufgaben



Sonnet 5
Bestes Gesamtpaket

Alltägliche Entwicklung, ausgewogen



Haiku 4.5
Schnell & günstig

Hohes Volumen, niedrige Latenz

Kosteneffiziente Alternativen (eigenes LLM-Gateway)

Alternativen zu Opus 5

- **GLM 5.3** – Engste Annäherung ans Opus-Niveau, ca. \$1,40/\$4,40 statt \$5/\$25 – Fallback bei Spend-Cap
- **DeepSeek V4 Pro 0813** – Nah am Frontier-Niveau in Einzeltests, aber noch nicht breit bestätigt

Alternativen zu Sonnet 5

- **Qwen 3.8 Flash** – Bei Coding oft besser als Sonnet – False-Positives im Blick behalten
- **DeepSeek V4 Pro 0813** – Zweite Quelle für Coding, diversifiziert das Risiko
- **Qwen 3.8 Max** – Stark bei Reasoning/Wissen, schwächer bei Coding-Volumenlast
- **DeepSeek V4 Flash 0731** – Günstigste/schnellste Option, schwächstes Profil der Gruppe

Prompting-Frameworks für Claude

So haben wir diesen Vortrag selbst mit Claude aufgebaut

```
<persona> Du bist mein Sparringspartner für einen Vortrag vor PowerBuilder-Entwicklern.</persona>
<context> Wir haben den MSB-Monitoringbericht von N+1 (tausende DB-Queries je Lauf) auf einen bewiesenen
Cache umgebaut (ESHDF-8654) - Code und Tickets liegen vor.</context>
<task> Strukturiere den Vortrag entlang drei Fragen: Was haben wir verbessert? Warum ging reines DB-
Tuning nicht? Wie haben wir es gelöst?</task>
<constraints>
- Nur Fakten, die im Code oder in den Tickets verifizierbar sind
- Verständlich ohne WinEV-Fachwissen
</constraints>
```

1

Klare Struktur

XML-Tags trennen Anweisungen sauber voneinander.

2

Konkrete Constraints

Vermeiden Rückfragen und Fehlinterpretationen.

3

Iteratives Verfeinern

Perfektion entsteht im Dialog, nicht im ersten Prompt.

Der Fall: MSB-Monitoringbericht

21.000 Datensätze, tausende Einzel-Queries – und wieso kein Index half

- Ein Bericht mit rund 21.000 Datensätzen je Lauf – pro Zeile mehrere fachliche Einzelermittlungen.
- Jede Ermittlung: ein eigener, bereits gekeyter DB-Zugriff (Primär- oder Fremdschlüssel).
- Kein fehlender Index, kein teurer Join – der Engpass ist die Anzahl der Roundtrips.
- Ein Index macht aus tausend schnellen Queries keine einzige – nur weniger Aufrufe helfen.

21.000

Berichtszeilen, jede mit eigenem Einzelzugriff

~1.500–3.000

DB-Queries je Ermittlung – laut Funktionsheader im Code

Werkzeug: der Trace-Bauplan

Vollständiger Mechanismus: Instanzvariablen, INI-Schalter, CSV-Export, CPU()-Messung

```
// Instanzvariablen des Fensters
boolean      ib_trace = false
string       is_trace_dir = "C:\temp"
integer      li_trace_file
long         ll_t0, ll_t_sec1, ll_tc_sec1

// open-Event: Schalter frisch aus der INI lesen
ib_trace      = ProfileInt(  n_sys_message.IS_APPL_INI_NAME, &
                           'Debug', 'WinEVTrace', 0) = 1
is_trace_dir = ProfileString(n_sys_message.IS_APPL_INI_NAME, &
                           'Debug', 'WinEVTraceDir', 'C:\temp')

if ib_trace then
    li_trace_file = FileOpen(is_trace_dir + "\trace_msb_ims.csv", &
                           LineMode!, Write!, Shared!, Replace!)
    FileWrite(li_trace_file, "row;zaehler_n;zaehler_cpu;vertrag_n;vertrag_cpu")
end if

// retrieveend, je Berichtszeile und Abschnitt:
ll_t0 = CPU()
ls_typ = parent.uf_getzaehlerzusgerforderte_cached(ls_id, ld_stichtag, ...)
ll_t_sec1 += CPU() - ll_t0
ll_tc_sec1 ++

if ib_trace then
    FileWrite(li_trace_file, string(ll_row) + ";" + string(ll_tc_sec1) + ";" + string(ll_t_sec1))
end if
```

Cache, Baustein 1: Bulk-Load

Ein SELECT vor der Schleife ersetzt N Einzelabfragen in der Schleife

```
// ESHDF-8654: ein DataStore je fachlichem Bereich, plus Ladeflag
n_sys_datastore  ids_abrechnungen_cache, ids_zaeher_cache
boolean          ib_abrechnungen_cache_loaded = false

// Bulk-Load: einmal pro Berichtslauf, VOR der Zeilenschleife
if not invalid(ids_abrechnungen_cache) then
    f_init_dataStoreSQLCA(ids_abrechnungen_cache, "d_ds_abrechnungen_cache")
    ids_abrechnungen_cache.allowselmandant = false
end if

ls_select = "SELECT DISTINCT dnetzabrech.int_nr, dnetzabrech.dnetzkunde_id, ... "
ids_abrechnungen_cache.restprogrammfilter = &
    gettripel("dnetzabrech", "", ls_select)

ll_rows = ids_abrechnungen_cache.retrieve()
ib_abrechnungen_cache_loaded = true      // auch 0 Zeilen = geladen
```

Der abgeschaltete Mandantenfilter ist bewusst: der Suchschlüssel ist global eindeutig, nicht mandantengebunden.

Cache, Baustein 2: Sortierung & find()

Sortierung plus sequenzieller Cursor bauen nach, was ein B-Tree-Index leistet

```
// Vorbedingung für find(): Sortierschlüssel = Suchschlüssel
if ll_rows > 0 then
    ids_abrechnungen_cache.SetSort("dnetzkunde_id A, datum_ab A, datum_bis A")
    ids_abrechnungen_cache.Sort()
end if

// Memoization: derselbe Schlüssel wie beim letzten Aufruf?
if as_dnk_id = is_jvb_memo_dnk_id then
    adec_jahresarbeit = idec_jvb_memo_result
    return ib_jvb_memo_success
end if

ll_row = ids_abrechnungen_cache.find("dnetzkunde_id = '" + as_dnk_id + "'", &
                                     1, ids_abrechnungen_cache.rowcount())

do while ll_row > 0 and ll_row <= ids_abrechnungen_cache.rowcount()
    if ids_abrechnungen_cache.getitemstring(ll_row, "dnetzkunde_id") <> as_dnk_id then
        exit                // Cache ist sortiert -> Block endet hier
    end if
    adec_jahresarbeit += ids_abrechnungen_cache.getitemdecimal(ll_row, "jahresarbeit")
    ll_row++
loop

is_jvb_memo_dnk_id    = as_dnk_id
idec_jvb_memo_result = adec_jahresarbeit
```

Verify: dem Cache beweisen

Jede Frage zusätzlich an die alte Funktion stellen und beide Werte vergleichen

```
// einmalig, wenn ib_trace: zweite CSV für den Korrektheits-Verify
if ib_trace then
    li_verify_file = FileOpen(is_trace_dir + "\trace_msb_ims_verify.csv", &
                             LineMode!, Write!, Shared!, Replace!)
end if

lb_result = uf_getjahresarbeit_cached(ls_dnk_id, ld_stichtag, ldec_cache)

if ib_trace then
    // dieselbe Frage zusätzlich an die alte DB-Funktion stellen
    in_preiskategoriedaten.uf_getjahresarbeit(ls_dnk_id, ld_stichtag, ldec_db)

    if round(zeroifnull(ldec_cache),0) <> round(zeroifnull(ldec_db),0) then
        ll_verify_nok++
        FileWrite(li_verify_file, string(ll_row) + ";jahresarbeit;" + &
                  string(ldec_cache) + ";" + string(ldec_db) + ";NEIN")
    else
        ll_verify_ok++
    end if
end if
```

Die alte Funktion bleibt erhalten – sie ist die unabhängige Referenz, nicht Altlast, sondern Beweismittel.

Fallback: kontrolliert statt still

Zwei Fallback-Ebenen – eine entfernt, eine gezählt statt verschwiegen

```
// Lade-Fallback ENTFERNT (ESHDF-8654): Cache wird immer vor dem Loop geladen
if not ib_abrechnungen_cache_loaded then
    // kein stiller Rückfall mehr pro Zeile - Ladefehler wird gemeldet:
    getInfos().raiseerror("Abrechnungen-Cache konnte nicht geladen werden")
    return false
end if

// Feldweiser Fallback BLEIBT - aber gezielt und gezählt
ls_zaeher = uf_get_cached_field("zaehler", "int_nr", ls_zaeherId, "nummer")

if emptyifnull(ls_zaeher) = "" then
    ls_zaeher = f_zusatzfeldfeldex("zaehler", "int_nr", ls_zaeherId, "nummer", false)
    ii_trace_fb1 ++           // Miss zählen statt still übergehen
    if ib_trace then
        FileWrite(li_trace_file, string(ll_row) + ";fallback_zaeher;" + ls_zaeherId)
    end if
end if
```

Ohne diese Zähler sieht jeder Cache vollständig aus – auch wenn er es nicht ist.

On the fly: den Trace live mitverfolgen

Shared! statt exklusiv geöffnet – die CSV ist lesbar, während der Bericht sie noch schreibt

```
// FileOpen mit Shared! statt Exclusive! – das ist der ganze Trick
li_trace_file = FileOpen(is_trace_dir + "\trace_msb_ims.csv", &
                        LineMode!, Write!, Shared!, Replace!)

// retrieveend: pro Berichtszeile wird sofort eine Zeile geschrieben
FileWrite(li_trace_file, string(ll_row) + ";" + string(ll_tc_sec1) + ";" + string(ll_t_sec1))

# Extern, WÄHREND der Bericht noch läuft (die ~21.000-Zeilen-Schleife dauert):
Get-Content "C:\temp\trace_msb_ims.csv" -Tail 20 -Wait
```

Kein Sonderwerkzeug nötig – nur die Zugriffsart im FileOpen entscheidet, ob man warten muss oder live mitliest.

Ergebnis

Korrektheit ist belegt – nicht behauptet

7.970

maschinelle Einzelvergleiche Cache gegen alte Funktion

28

Abweichungen gefunden – zu einer Arbeitsliste mit 26
Einträgen destilliert

Ein Cache ohne Korrektheitsbeweis ist kein Fortschritt, sondern ein verschobenes Problem.



Vielen Dank für Ihre Aufmerksamkeit!

Fragen? Diskussion?

Thomas Enzenberger · Power People IT-Services GmbH · thomas.enzenberger@powerpeople.de