Um einen Baum zu Fällen, brauchst du eine scharfe Axt – ***Visual Studio Code im APEX-Umfeld***

Maik Michel, Opitz Consulting Deutschland

Du kannst einen Baum auch mit stumpfer Axt mühsam zu Fall bringen oder du investierst etwas Zeit ins Schärfen deines Tools und erreichst dein Ziel dann einfacher und schneller. Der Editor ist dein tägliches Werkzeug. Er sollte deine Arbeit optimal unterstützen. Auf dem Markt gibt es viele Editoren und IDEs. Aber im Datenbankumfeld wird es rar. Viele nutzen Tools wie den SQL Developer von Oracle, den TOAD von Dell/Quest oder andere Produkte. Dabei bietet der Markt zusätzlich noch viele andere Tools. Gerade im Web-Umfeld entwickelt sich der Trend, sich hier klassischer Editoren zu bedienen, statt auf die oftmals überladenen IDEs zu setzen. Mit Visual Studio Code (Open-Source) hat hier Microsoft der Entwicklergemeinde einen ziemlich guten Editor zur Verfügung gestellt.

Was ist VSCode

Visual Studio Code ist ein Quelltexteditor aus dem Hause Microsoft und steht uns kostenlos zur Verfügung. VSCode ist OpenSource. Der Quellcode ist auf GitHub veröffentlicht und unterliegt der MIT-Lizenz. Microsoft bietet den Editor für alle gängigen Plattformen (Windows, MacOS, Linux) zum Download an. Zusätzlich zur „normalen" Version gibt es eine sogenannte Insider-Build-Version. Diese beinhaltet die bereits geplanten nächsten Features, die so oder in einer ähnlichen Form demnächst in der Standard-Version das Licht der Welt erblicken. VSCode unterliegt einem monatlichen Updatezyklus. Anwender und hierbei vor allem Entwickler können sich somit immer über neue Features und regelmäßige BugFixes freuen. Technisch betrachtet ist VSCode eine Electron App und basiert im Kern auf dem Monaco-Editor, der auch seit der Version 20.1 innerhalb von APEX als Sourcecode-Editor benutzt wird.

Unterschied zu „herkömmlichen" IDEs

Anders als die typischen IDEs, mit denen wir als Datenbankentwickler arbeiten, handelt es sich bei VSCode erst mal nur um einen rein dateibasierten Texteditor.

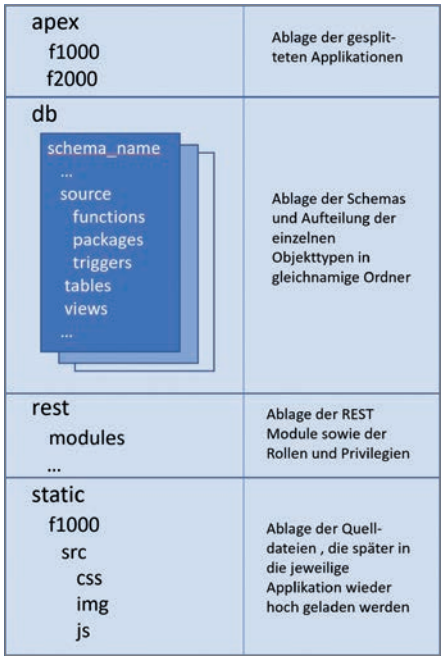

Abbildung 2: Verzeichnisstruktur (Quelle: Maik Michel)

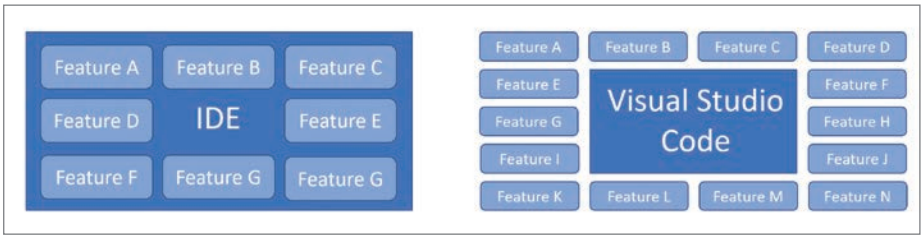

Abbildung 1: Unterschied IDE zu VSCode (Quelle: Maik Michel)

Wir benötigen keine permanente Verbindung zur Datenbank. IDEs wie TOAD, SQL Developer, PL/SQL-Developer oder DataGrip bringen von Hause aus eine Menge an Features mit. Sie lassen sich nur schwer oder gar nicht erweitern oder anpassen *(siehe Abbildung 1)*. Die Hauptdevise von VSCode heißt: Make it your own. Dahinter verbergen sich ein ausgeklügeltes Konfigurationsmanagement und ein Marktplatz voller Extensions, der für jede Anforderung etwas bereithält.

Ein weiterer Unterschied ist das Arbeiten im Dateisystem. Anders als bei den „normalen" Datenbank-IDEs arbeitet VSCode komplett auf dem Dateiverzeichnis des Anwenders. So braucht es auch keine ständige Datenbankverbindung, um zum Beispiel Tabellen anzulegen oder Packages kompilieren zu können. Das verschafft uns enorme Vorteile in verschiedenen Aspekten des Entwicklungszyklus. Zum einen lassen sich Änderungen direkt in einer Versionsverwaltung einchecken, was die Fehleranfälligkeit beim Ausliefern von Updates reduziert. Zum anderen ist der dateibasierte Zugriff über VSCode enorm schnell. Die verschiedenen Komponenten einer Datenbank / APEX-Applikation lassen sich schnell öffnen und Impactanalysen können so schnell durchgeführt werden.

Natürlich kann VSCode nicht alles ab Werk. In diesem Artikel gebe ich einen Einblick, wie man VSCode speziell für die Applikationsentwicklung im APEX-Umfeld einsetzen kann.

Schärfung der Axt

VSCode arbeitet projektbezogen. Ein Projekt ist dabei ein bestimmtes Verzeichnis. Dieses Verzeichnis stellt im Optimalfall ein Repository in einer Versionsverwaltung dar. Hierzu bietet sich die Nutzung von Git an. VSCode unterstützt das Öffnen eines Ordners über das Kontextmenü im Windows-Explorer und auch im Finder unter MacOS. Es empfiehlt sich, eine bestimmte Verzeichnisstruktur für ein APEX- / Datenbankprojekt zu etablieren *(siehe Abbildung 2)*.

Innerhalb von VSCode können viele Funktionen per Hauptmenü oder Kontextmenü aufgerufen werden. Aber das eigentliche Herzstück, die Kommandozentrale, ist die Auswahl eines Kommandos durch *Strg+Shift+P*. Hier wird nun eine durchsuchbare Liste aller verfügbaren Kommandos angezeigt. So kann man zum Beispiel eine Variable markieren und diese mit der Eingabe *Strg+Shift+P +"lowerc*" in Kleinbuchstaben transformieren *(siehe Abbildung 3)*. Immer wenn Sie eine Aktion / ein Kommando ausführen möchten, erreichen Sie dies über Strg+Shift+P. In der Auswahl-

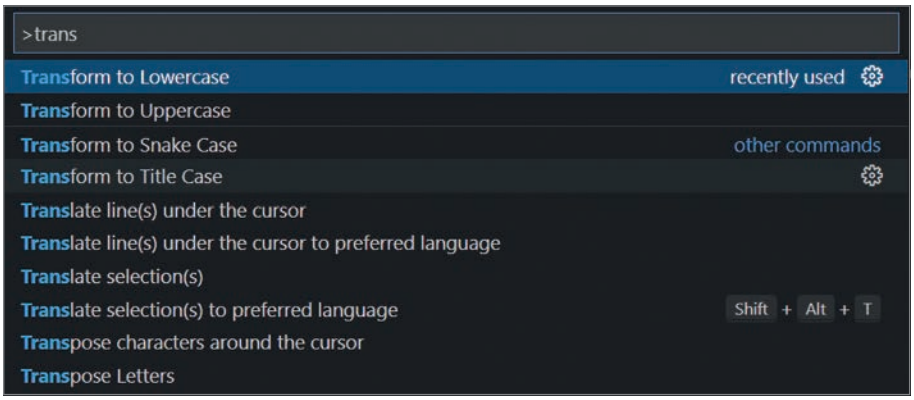

Abbildung 3: List Commands (Quelle: Maik Michel)

liste der Kommandos werden auch die Keyboard-Shortcuts angezeigt, wenn konfiguriert. Mit diesen kann das jeweilige Kommando direkt aufgerufen werden. Die Shortcuts sind dabei frei konfigurierbar (*Strg+K Strg+S*). Des Weiteren können Sie auch die Shortcuts anderer Editoren wie VIM, Atom oder Sublime durch entsprechende Extensions importieren.

Alle Einstellungen, die Sie vornehmen, können pro Benutzer (global) oder pro Workspace gespeichert werden. Das bringt enorme Vorteile mit sich. So können Sie zum Beispiel in einem Projekt Git-bezogene Extensions nutzen und in einem anderen Projekt, falls erforderlich, auf SVN-bezogene Extensions zurückgreifen. Nicht benötigte Extensions müssen somit nicht im RAM vorgehalten werden. Mit *Strg+* können Sie die jeweiligen Einstellungen bearbeiten. Projektbezogene Einstellungen werden dabei im Projektverzeichnis unter .vscode/settings.json abgelegt und können natürlich auch hier bearbeitet werden.

Tipp: Mit der Extension Peacock können Sie sich den äußeren Rand von VSCode unabhängig vom installierten Theme farblich anpassen. So verlieren Sie nie den Überblick, wenn Sie mit mehreren Projekten arbeiten *(siehe Abbildung 4)*.

Das Hauptmotto von VSCode ist, wie bereits erwähnt, die uneingeschränkte Erweiterbarkeit durch Extensions: *Make it your own*. Über den entsprechenden Navigationspunkt am linken Rand oder mit *Strg+X* gelangen Sie zum Verwaltungsbereich der

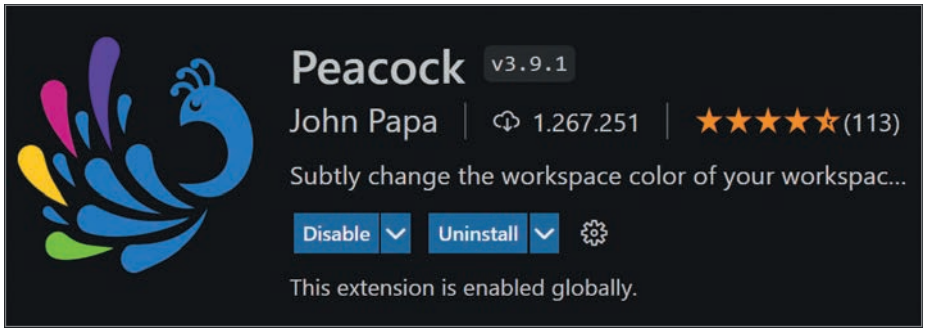

Abbildung 4: Extension Peacock (Quelle: Maik Michel)

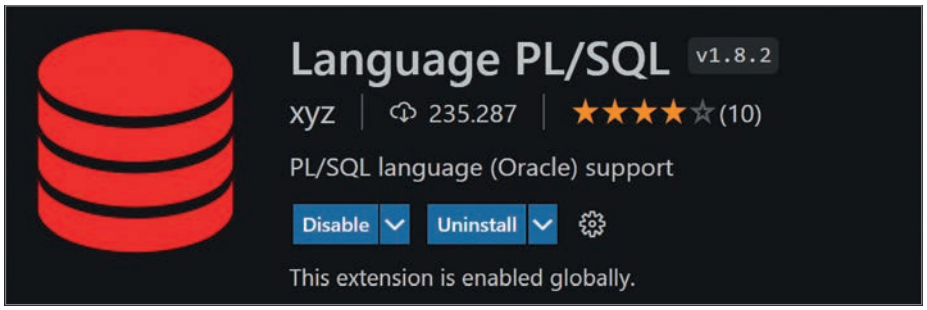

Abbildung 5: Extension Language PL/SQL (Quelle: Maik Michel)

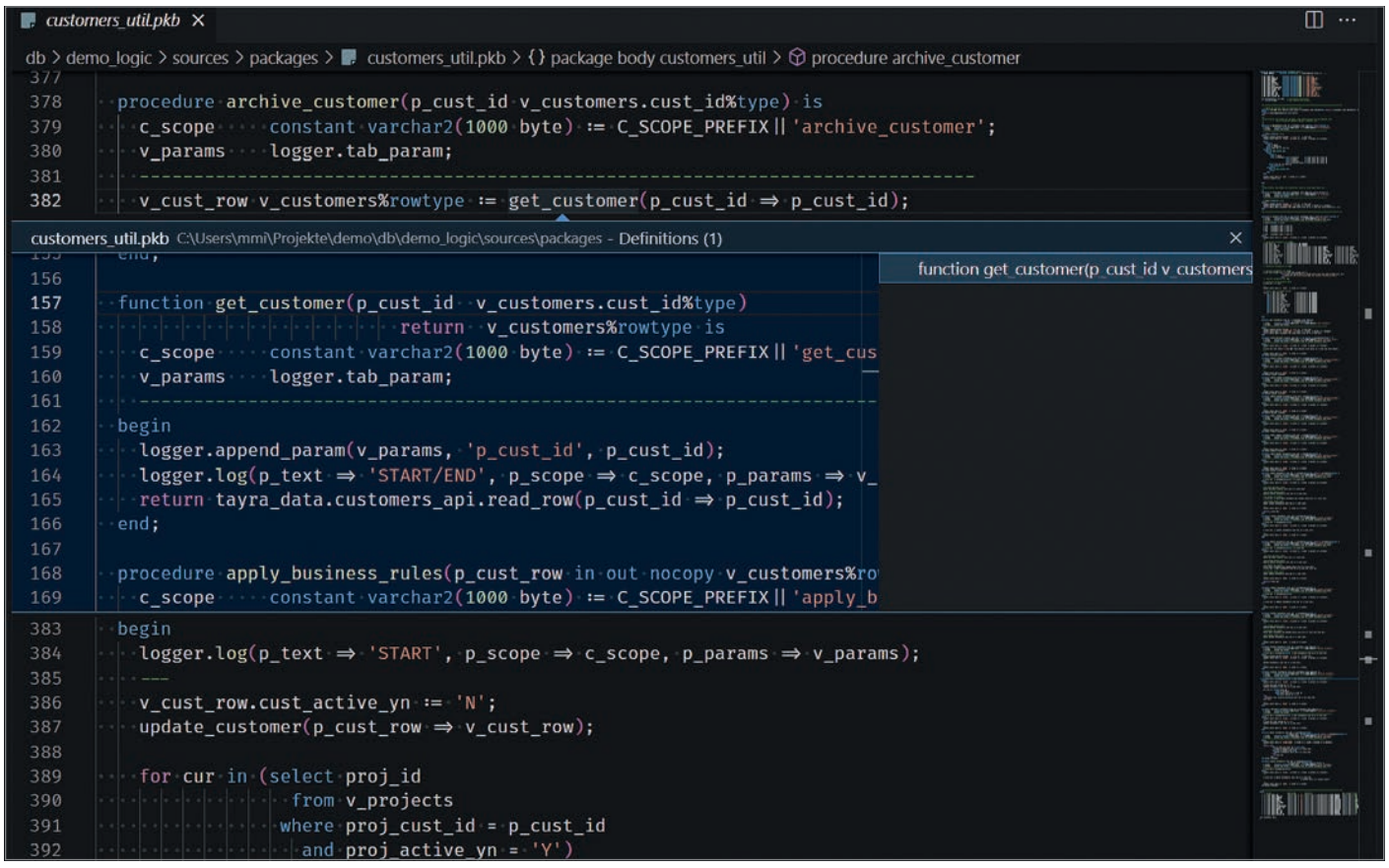

Abbildung 6: Peek-Bereich (Quelle: Maik Michel)

```
{
   "version": "2.0.0",
   "tasks": [{
       "label": "sqlplus",
       "command": "sqlplus",
       "args": ["username/password@sid", "@\"${file}\""]
   }]
}
```

*Listing 1: BuildTask mit SQL*Plus*

Extensions. Hier können Sie Extensions suchen, installieren, deinstallieren oder auch gezielt für einen Workspace an- oder ausschalten. Zusätzlich stellt Microsoft über die URL: *https://marketplace.visualstudio.com/vscode* einen Marktplatz bereit, von dem die bereits genannten Funktionen ebenfalls erreichbar sind. Ähnlich wie in einem App-Store können auch hier die Extensions bewertet werden. Sofern es sich bei einer Extension um Open Source handelt, gelangen Sie auch zum jeweils hinterlegten Repository auf GitHub.

VSCode für APEX-Entwickler

Nach dieser kurzen Einführung in VSCode widmen wir uns nun dem eigentlichen Thema: Wie kann VSCode uns als APEX-Entwickler unterstützen? Die Hauptaufgabe von VSCode bleibt auch für uns das Bearbeiten von Quellcode. VSCode dient hier nicht als vollständiger Ersatz für SQL Developer oder TOAD. VSCode übernimmt hier das Bearbeiten von Skripten, Packages, JavaScript und CSS. Datenanalysen, ER-Diagramme usw. können mit anderen Tools besser bearbeitet werden. VSCode unterstützt von Hause aus kein PL/SQL. Das muss für uns eine Sprach-Erweiterung übernehmen *(siehe Abbildung 5)*.

Nach der Installation von „Language PL/SQL“ haben wir nicht nur Syntax-Highlighting für PL/SQL als Funktionalität hinzubekommen. Der Code wird nun für uns auch geparst und VSCode erlaubt uns mithilfe der OutlineView oder *Strg+Shift+O* das Springen zu den jeweiligen Methoden. Zusätzlich können wir mit *Strg+Klick* auf den Methoden-Namen oder mit *F12* zur Methode selbst navigieren. Durch *Alt+F12* zeigt uns VSCode diese Methode in einem sogenannten Peek-Bereich an *(siehe Abbildung 6)*. Das funktioniert sowohl innerhalb des gerade geöffneten Packages als auch innerhalb des geöffneten Workspace, da jede Datei im Hintergrund indiziert wird. Des Weiteren steht uns nun IntelliSense zur Verfügung. Das heißt, wir bekommen während der Eingabe Vorschläge zur Code-Vervollständigung angezeigt. Die Vorschläge bedienen sich aus den im Workspace vorhandenen Dateien.

Wie kommt nun unser Quellcode in die Datenbank?

In erster Linie bedienen wir uns hier SQL*Plus oder SQLcl. Ich empfehle hier SQL*Plus, da die Startzeit hierfür erheblich schneller ist als die von SQLcl. Beide Tools bringen aber ihre Vorteile mit sich und sollten in keiner Entwicklungsumgebung fehlen. Um nun also eine in VSCode geöffnete Datei mithilfe von SQL*Plus in eine Datenbank einzuspielen, bedienen wir uns der sogenannten BuildTasks. Diese stehen uns via *Strg+Shift+B* oder mit dem Command „Task: Run Build Task“ zur Verfügung. Sollte noch kein BuildTask angelegt sein, haben wir die Möglichkeit, einen einfachen HelloWorld-Task durch VSCode anlegen zu lassen, und erlangen so einen ersten Eindruck, wie die Definition solcher Tasks auszusehen hat. BuildTasks werden im Workspace unter .vscode/tasks.json gespeichert. BuildTasks können sich verschiedener VSCode- und Umgebungsvariablen bedienen. So könnten wir einen simplen BuildTask erzeugen, der mit einer Datenbankverbindung und der aktuell geöffneten Datei SQL*Plus aufruft *(siehe Listing 1)*.

Das funktioniert so weit, hat aber zwei Nachteile. Erstens wird sich leider

```
#!/bin/bash
# call_sqlplus.sh
DB_CONNECTION=$1
SOURCE_FILE=$2

sqlplus -s -l $DB_CONNECTION <<!
set serveroutput on
set scan off
set define off
set pagesize 0
set linesize 2000
set wrap off
set trim on
set sqlblanklines on
set heading off

@"$SOURCE_FILE"

select user_errors
  from (
     select lower(attribute) || ' ' || line || '/' ||
position || ' ' || name
          || case
                when type = 'PACKAGE' then '.pks'
                when type = 'PACKAGE BODY' then '.pkb'
                else '.sql'
             end || ' '
          || replace(text, chr(10), ' ') as user_errors
     from user_errors
     where attribute in ('ERROR', 'WARNING')
     order by attribute desc, type, name, line, position
     ) ;
!
```

*Listing 2: Shell-Skript zum Aufruf von SQL*Plus*

SQL*Plus nicht von selbst schließen. Man ist gezwungen, in der nun erscheinenden SQL*Plus-Shell *exit* einzugeben. Zweitens nehmen wir uns die Möglichkeit, den Output durch VSCode parsen zu lassen, um uns mögliche Kompilierungsfehler oder Warnungen anzuzeigen. BuildTasks können mit einem sogenannten Problem Matcher konfiguriert werden. VSCode parst mit diesem die Ausgabe via Regex und hebt so die fehlerhafte Zeile entsprechend hervor. Somit kapseln wir den Aufruf durch ein Shell-Skript, das für uns SQL*Plus aufruft, die Tabelle USER_ERRORS nach Fehlern abfragt sowie die Ausgabe passend für unseren Problem Matcher formatiert. Dabei ersetzen wir das Command im BuildTask durch den jeweiligen Skript-Interpreter, beispielsweise Bash *(siehe Listing 2 und 3)*.

Als APEX-Entwickler kompilieren wir aber nicht nur Packages oder Funktionen für die Datenbank. Wir erstellen Tabellen und Views, lassen gegebenenfalls automatisch Table-API-Packages generieren, schreiben JavaScript oder CSS-Code, laden diesen und auch Bilder als Static-Files in Applikationen, exportieren Applikationen und/oderREST-Module und führen Tests aus. All das kann man mit den BuildTasks konfigurieren. Und Sie merken schon, das kann dann doch etwas kompliziert werden. Aus diesem Grund habe ich die Extension dbFlux entwickelt. dbFlux nimmt Ihnen die komplette Konfiguration dieser BuildTasks ab *(siehe Abbildung 7)*.

```
{
  "version": "2.0.0",
  "tasks": [{
      "label": "sqlplus with bash",
      "command": "bash",
          "args": [
              "call_sqlplus.sh",
              "username/password@sid",
              "@\"${file}\""],
          "group": {
              "kind": "build",
              "isDefault": true},
          "problemMatcher": {
              "fileLocation": [
                  "relative",
                  "${fileDirname}"],
              "pattern": [{
                  "regexp": "(.*) (\\d*)/(\\d*) (.*?) (.*)",
                  "severity": 1,
                  "line": 2,
                  "column": 3,
                  "file": 4,
                  "message": 5,
                  "loop": true}]
              }
      }]
}
```

Listing 3: BuildTask mit Bash

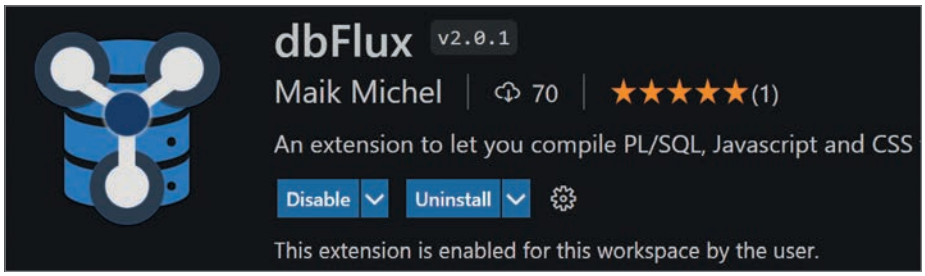

Abbildung 7: Extension dbFlux (Quelle: Maik Michel)

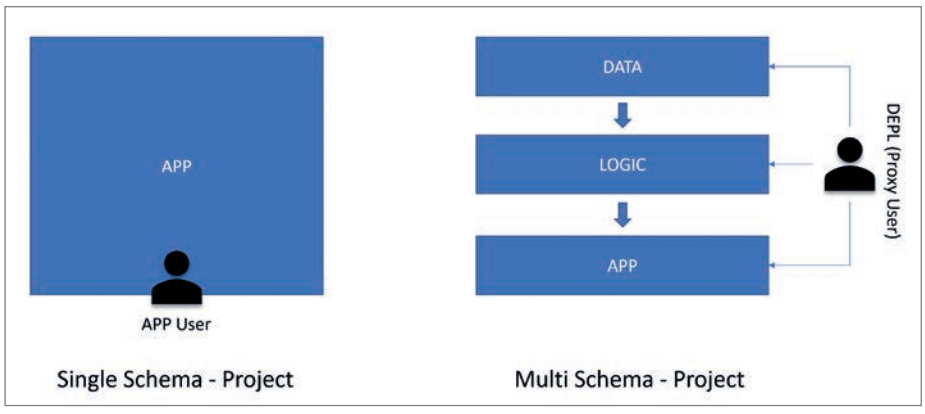

Abbildung 8: Single- versus Multi-Schema-Projekt (Quelle: Maik Michel)

Wie kommt nun unser Quellcode in die Datenbank

Sobald Sie dbFlux installiert haben, können Sie ein bereits konfiguriertes Projektverzeichnis öffnen oder aber sich eins von dbFlux anlegen lassen. Dabei werden die erforderliche Verzeichnisstruktur angelegt und die Projektinformationen gespeichert. Mit dbFlux können Sie entweder ein Multi-Schema- oder Single-Schema-Projekt erstellen. In einem Multi-Schema-Projekt wird ein Proxy-Benutzer für die Verbindung zu den jeweiligen Schemas benutzt. Im Single-Schema-Projekt wird sich direkt mit dem jeweiligen Datenbank-Schema verbunden *(siehe Abbildung 8)*.

Wenn Sie nun zum Beispiel ein Package öffnen und mit *Strg+Alt+B* oder dem Command „dbFlux:Compile current file“ kompilieren, wird das Package mit der dem jeweiligen Schemaverzeichnis entsprechenden Datenbankverbindung kompiliert. Im Hintergrund passiert natürlich genau das, was wir bei den BuildTasks schon gesehen haben. SQL*Plus wird per Shell gestartet, die Ausgabe wird über einen Problem Matcher gelesen und für VSCode vorbereitet. Fehler werden im Problems Panel, im Dateiexplorer sowie in der jeweiligen Zeile im Editor angezeigt *(siehe Abbildung 9)*.

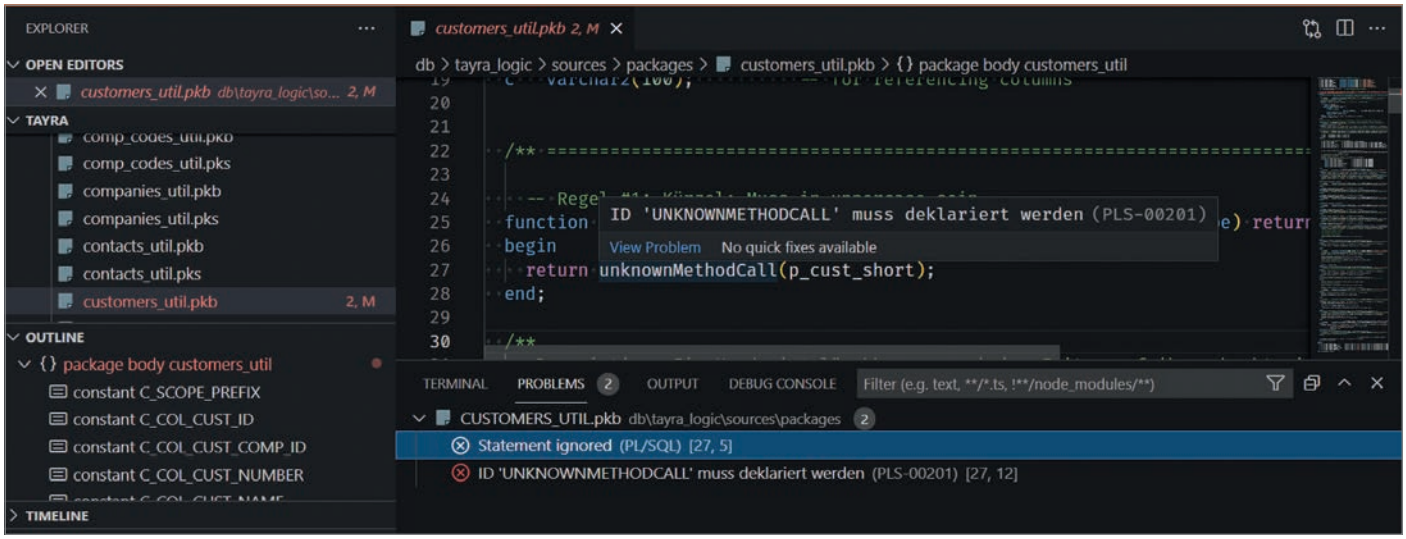

Abbildung 9: Build Package mit dbFlux (Quelle: Maik Michel)

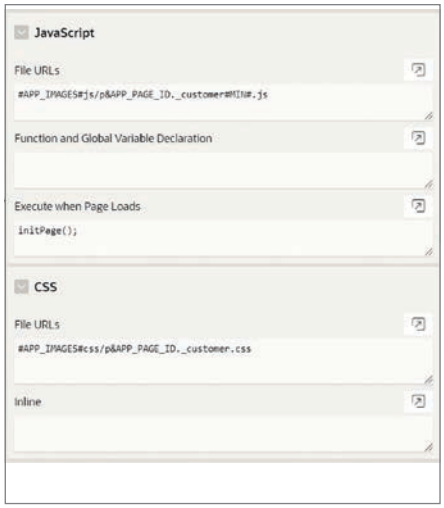

Abbildung 10: Einbinden der StaticFiles (Quelle: Maik Michel)

Eine APEX-Applikation besteht jedoch nicht nur aus PL/SQL oder SQL-Code. Vieles wird auch per JavaScript oder CSS implementiert. Es empfiehlt sich, diesen Code nicht in APEX über den Page-Builder zu implementieren, sondern per StaticFile in die Applikation hochzuladen *(siehe Abbildung 10)*. dbFlux bietet hierzu das Verzeichnis static/src/fXXX an, wobei fXXX für die jeweilige Applikation steht. Dateien, die Sie im Unterordner /js über den BuildTask kompilieren, werden nicht nur zur passenden Applikation hochgeladen. dbFlux erstellt auch eine minifizierte Version sowie eine SourceMap zu dieser JavaScript-Datei und lädt auch diese beiden Dateien entsprechend zur Applikation hoch *(siehe Abbildung 11)*.

Ähnlich verhält es sich im Unterordner /css. Hier per *Strg+Alt+B* kompilierte Stylesheets werden in die jeweilige Applikation zuzüglich einer minifizierten (uglified) Version hochgeladen. Die Dateien, die sich in anderen Unterordnern befinden, werden ebenfalls hochgeladen, unterliegen aber keinem weiteren Regelwerk. In erster Linie können das zum Beispiel Bilder oder Templates etwa für APEXOfficePrint (AOP) sein.

VSCode wurde ursprünglich von Entwicklern im Web-Umfeld benutzt und bietet daher enorm viel Unterstützung zur Bearbeitung von JavaScript- und CSS-Dateien. Dabei handelt es sich zum Beispiel um Erweiterungen zur Code-Formatierung (wie Prettier) oder Linter (wie ESLint), die bereits während der Code-Eingabe direkt auf mögliche Syntaxfehler hinweisen. Zudem bietet VSCode durch IntelliSense sinnvolle Optionen zur Vervollständigung des Quellcodes an *(siehe Abbildung 12)*.

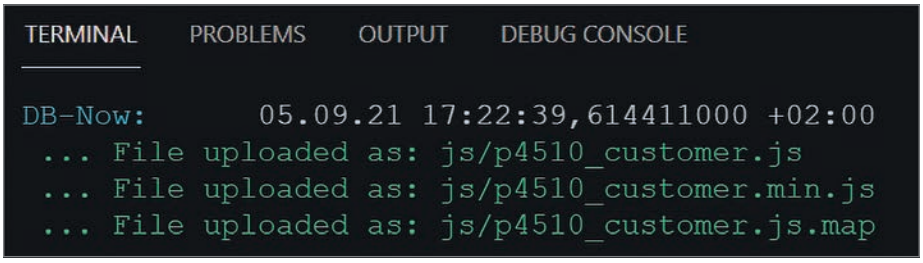

Abbildung 11: dbFlux - Upload der JavaScript-Dateien (Quelle: Maik Michel)

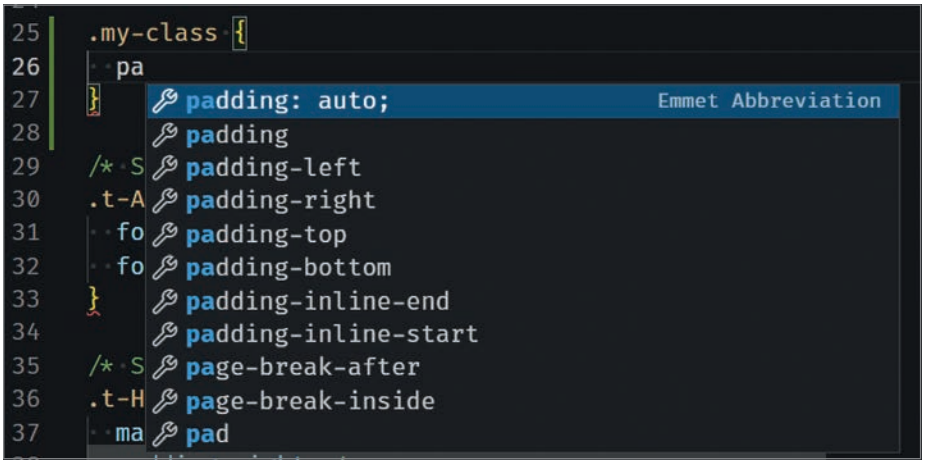

Abbildung 12: CSS IntelliSense (Quelle: Maik Michel)

Tipp: Mit der Extension Wrap Console Log können Sie sich console.log-Statements mit der gerade selektierten Variable erzeugen lassen *(siehe Abbildung 13)*.

Haben wir als Entwickler alle nötigen Änderungen in unserer Applikation implementiert, wollen wir diese natürlich auch exportieren. Mit dbFlux ist auch das möglich. Aufgrund der Benennung der Unterverzeichnisse im Ordner apex (apex/f1000, apex/f2000) werden diese zur Auswahl mit dem ShortCut *Strg+Alt+E* angezeigt. Entscheidet man sich für eine Applikation, wird diese mithilfe von SQLcl exportiert und gesplittet. Eine ähnliche Funktion findet man auch für REST-Module.

Tipp: Probieren Sie die Extension REST Client aus, wenn Sie REST Services durch die Implementierung von PL/SQL Packages entwickeln. Hier können Sie den REST Request als Kommentar an der jeweiligen Methode hinterlegen. Wenn Sie diesen anschließend markieren, können Sie den Response direkt in VSCode betrachten *(siehe Abbildung 14)*.

Die von uns implementierte Applikation wird natürlich auch getestet. Dabei nutzen wir für die UnitTests das Framework utPLSQL. Innerhalb des Verzeichnisbaums finden wir hierfür den Ordner tests/packages. Hier legen wir die Packages ab, die wir zum Testen unserer Business-Logik benötigen. Mit dem Command „dbFlux: Execute Tests" können wir variabel alle Schemas des Projektverzeichnisses auswählen oder auch nur ein bestimmtes. Einmal ausgewählt, werden die entsprechenden Tests ausgeführt und das Ergebnis per Shell-Ausgabe angezeigt *(siehe Abbildung 15)*.

Ich könnte an dieser Stelle noch viel mehr über die Benutzung von VSCode und den Einsatz verschiedener Extensions schreiben, doch das würde den Rahmen dieses Artikels sprengen. Für viele Themen, mit denen wir als Entwickler Tag für Tag konfrontiert sind, gibt es bereits eine oder mehrere Extensions. Probieren Sie diese aus und passen Sie sie an Ihren Workflow an. Da viele Extensions auf GitHub gehostet sind, haben Sie auch immer die Möglichkeit, mit den Entwicklern direkt in Kontakt zu treten.

Nehmen Sie Folgendes mit: Nutzen Sie VSCode und passen Sie es an. Make it your own! Viele tägliche Aufgaben können durch die Konfiguration von VSCode um ein Vielfaches beschleunigt und vereinfacht werden. Für fast jede Herausforderung gibt es eine passende Extension. Das Schärfen der Axt, die optimale Benutzung von VSCode, führt auf jeden Fall dazu, dass wir weniger Unterbrechungen durch das Wechseln zu verschiedenen Tools in unserem Workflow haben. Wir bleiben dadurch länger im Flow.

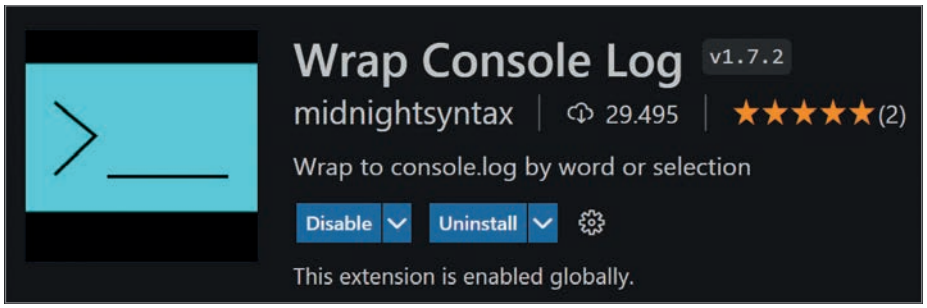

Abbildung 13: Extension Wrap Console Log (Quelle: Maik Michel)

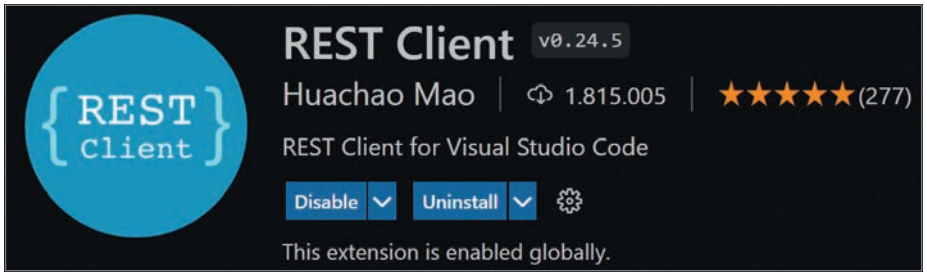

Abbildung 14: Extension REST Client (Quelle: Maik Michel)

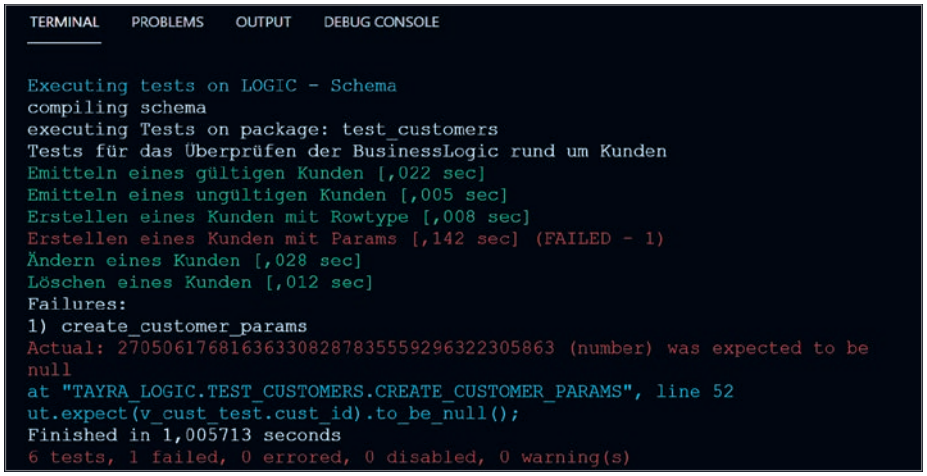

Abbildung 15: Mit dbFlux ausgeführter utPLSQL-UnitTest (Quelle: Maik Michel)

Über den Autor

Maik Michel ist LEAD Developer und Consultant bei der Opitz Consulting Deutschland GmbH. Hier verantwortet er verschiedene Projekte und Teams rund um das Thema Digitale Transformation mit der LowCode-Plattform APEX. Daneben ist er Autor des Blogs micodify.de und als Sprecher auf verschiedenen Konferenzen zu finden.

Maik Michel
maik.michel@opitz-consulting.com