SQL Select Begin day and End day

[crayon-67679ba093aff114222497/]
Result:

(No column name) (No column name)
2016-12-01 00:00:00.000 2016-12-31 23:59:59.900

Zuletzt ausgeführte Querys anzeigen lassen

ab SQL Server 2005
[crayon-67679ba093eca721939597/]
SQL Server 2005
[crayon-67679ba093ece895108870/]

SQL Select nur Datum von Datetime

[crayon-67679ba09404a805114775/]

T-Sql Datenbankverbindung schnell testen

Um mal eben schnell irgendwo zu testen, ob die Datenbankverbindung in Ordnung ist, gibt es einen kleinen Trick.

1. Man erstellt eine Textdatei und benennt die Dateiendung zu .udl um
2. rechte Maustaste, Eigenschaften
3. und dann auf Verbindung. Dort hat man nun alles nötige um eine Datenbankverbindung zu testen

---

mit Dapper ein SQL join mappen

Da ich von Entity Framework 6 aus Perfomancegründen überhaupt nicht zufrieden bin, frage ich mich ernsthaft, wie einige das produktiv einsetzen.

Daraufhin habe ich nach einer Alternative gesucht und bin bei Dapper, einem Projekt von Source stehen geblieben. Hier möchte ich einmal kurz zeigen, wie man mit Dapper eine Liste mit dem erstellten SQL Join erstellt.

Beispiel:

Wir haben in der SQL Datenbank eine Tabelle Automarken und eine andere mit Modelle.

Automarke: Id | Bezeichnung |

Modelle: Id | Bezeichnung | AutomarkeId

Es ist nicht schwer zu erkennen, dass hier eine 1:n Beziehung existiert.

Jetzt wollen wir das ganze in C# abbilden. Dazu bauen wir uns 2 Klassen
[crayon-67679ba0941e8366668962/]
Wir wollen nun eine List<Automarken> gefüllt mit allen Modellen der Automarken haben. Wie machen wir das?

Vielleicht hat der eine oder andere bereits eine Idee, aber denkt mal an die Performance.

Hier kommt Dapper ins Spiel. Dapper mappt für mich mit wenigen Zeilen Code eine Collection ganz nach meinem Geschmack.

Wie das geht, zeige ich hier….

1. Am besten und schnellsten ist es, wenn man Dapper von Nuget herunterlädt: **Install-Package Dapper**
2. Wir brauchen ein Objekt der Klasse SqlConnection, welches natürlich erfolgreich zur Datenbank connected ist

Im Grunde war dies auch schon. Ich habe Bemerkungen im Programmcode angefügt, damit es direkt verständlicher ist.
[crayon-67679ba0941ec260502532/]

SQL With

Kennt Ihr SQL Abfragen wie

[crayon-67679ba09438b337489647/]
jetzt könnte man hingehen und die case when Anweisung in die Where Klausel bringen. Dies würde aber den Code unnötig aufblähen und Fehler sind einprogrammiert. Jetzt bietet SQL die With Anweisung an. Diese kann man sich wie eine ausgelagerte View vorstellen, auf die man zugreifen kann.
[crayon-67679ba09438f161347003/]
Wir sehen , wir können auf die augelagerte Anweisung mit einem neuen Select zugreifen

---

Bulkupsert oder einfach Update, wenn bereits vorhanden ansonsten Insert in MSSQL

Lange habe ich nach einer vernünftigen Lösung gesucht, um riesige Datenmengen schnell in die Datenbank zu schreiben.

Als Quelle steht uns eine List<Class> mit der Klasse zur Verfügung, die der Datenbanktabelle gleicht und jede Menge Inhalte enthält. Versucht man diese Liste nun über das Entity Framework in die Datenbank zu jagen, merkt man schnell, dass das Entity Framework an seine Grenzen stößt. Alternativ haben wir das von der SQLCopy Klasse die BulkInsert() Methode, die aber nur Inserten kann. Findet das BulkInsert einen Index/Primärschlüssel, der bereits vorhanden ist, wird die ganze Show abgebrochen. Wie toll wäre es nun, wenn es nicht abbrechen würde, sondern diese Zeile einfach updated. Und das ganze Superschnell. Ich habe Testweise 10.000 Einträge in 400ms und 1.000.000 in 1 min in die Datenbank bekommen.

Methode Upsert:
[crayon-67679ba094569966844985/]
Methode BulkInsert:
[crayon-67679ba09456d581086118/]
und die beiden Extension Methods:
[crayon-67679ba09456f719091654/]

Angewendet wird das ganze ganz einfach so:

1. Parameter beinhaltet die Liste mit allen Inhalten
2. Parameter gibt an, welche Spalten geprüft werden sollen, ob es zu einem Update kommen soll oder Insert. In meinem Fall, wenn PersNr und CardId bereits in der Datenbank vorhanden sind, so mache ein Update, sonst Insert
3. Parameter gibt an, was inserted werden soll.
4. Parameter gibt an, was geupdated werden soll, wenn Parameter 2 zutrifft.

Bitte auf die tatsächlichen Primärschlüssel, Indizes und NOT NULL Schlüssel/Attribute achten. Ansonsten kommt man schnell zu einem Fehler
[crayon-67679ba094570895561380/]

MS-SQL Trigger

Trigger werden ausgeführt, wenn in einer Tabelle ein Ereignis ausgelöst wird.

Konkret heißt das, dass bei einem INSERT,UPDATE oder DELETE wir eingreifen können und einen oder mehrere Befehle „danach“

oder „anstelle“ von ausführen können.
[crayon-67679ba094786022751960/]
Wählen wir AFTER, dann wird der Datensatz erstellt/geupdated oder gelöscht und **erst dann** greift der Trigger ein.

Wählen wir stattdessen Instead Of, wird **anstelle von** dem Insert, Update oder Delete der Trigger ausgeführt.

Beim Trigger stehen uns 2 weitere Temporäre Tabellen zur Verfügung, nämlich inserted und deleted.

wenn wir z.B. einen neuen Datensatz id, Vorname, Nachname in die Datenbank geschrieben haben,

so können wir aus dem Trigger mit
[crayon-67679ba09478b022596317/]
die Id, die gerade eben eingetragen wurde abfragen.

Machen wir ein Update, so wird der alte Datensatz erst einmal erst in die temporäre Tabelle deleted gesteckt, welchen

wir ebenso mit
[crayon-67679ba09478c805427099/]
abfragen könnten.

SQL Server – Spalteninformationen einer

Tabelle selektieren

Manchmal möchte man Infos über die Spaltern einer bestimmte Tabelle

oder einer ganze Datenbank abrufen. Ich habe 2 Queries geschrieben mit den man diverse Informationen bekommt.

Schemainformationen für 1 Tabelle:
[crayon-67679ba09493e289397222/]
Schemainformationen für ganze Datenbank:
[crayon-67679ba094942802275549/]

SQL Server Querys

Duplicate löschen, welche sich aus col1 und col2 zusammen setzen. Alternativ kann man natürlich auch nur mit col1 arbeiten
[crayon-67679ba094af0619482053/]

Update Spalte, wenn Datum Heute > dann 1 sonst 0:
[crayon-67679ba094af5187072584/]
Zeilennummerierung erzeugen:
[crayon-67679ba094af6368823612/]