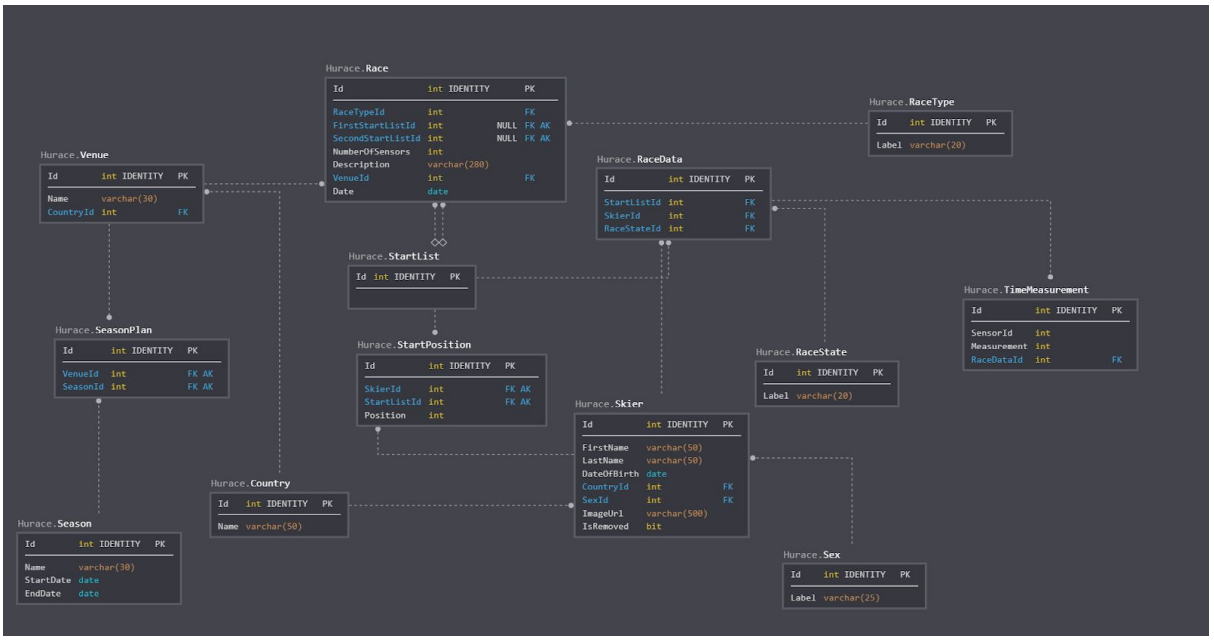
SWK/WEA Dokumentation

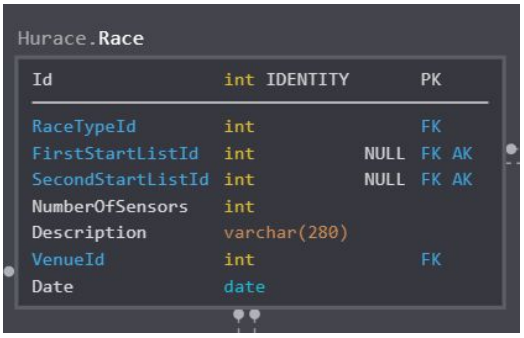
Datenbankmodell



Country

Enumeration für Länderkürzel (z.B.: “AUT”, “GER”, ...)

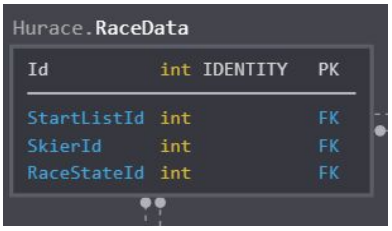
Race



In Race werden alle Relevanten Daten zu einem Rennen gespeichert.

RaceTypeId: Referenz auf den Renntyp
FirstStartListId: Referenz auf Startliste für ersten Durchgang
FirstStartListId: Referenz auf Startliste für zweiten Durchgang
NumberOfSensors: Anzahl der im Rennen verwendeten Sensoren
Description: Rennbeschreibung
VenueId: Referenz auf einen Veranstaltungsort
Date: Veranstaltungsdatum

RaceData

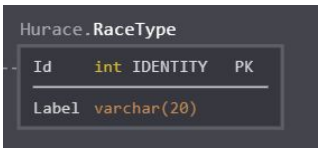


Mit RaceData wird gespeichert, welchen Zustand der Antritt eines Schifahrers bei einem Rennen hat.

RaceState

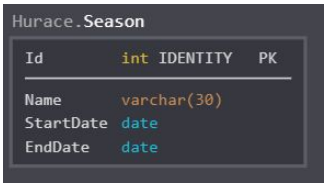
Enumeration für Rennstati (z.B.: Disqualifiziert, Abgeschlossen,...)

RaceType



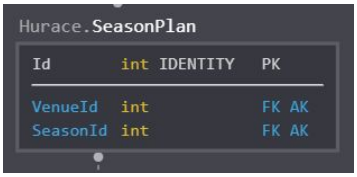
Enumeration für Renntypen (z.B.: Slalom, Riesentorlauf,...).

Season



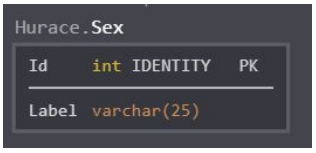
In der Seasontabelle wird Name und Dauer gespeichert, Maßnahmen um zu verhindern dass Saisonen gleichzeitig auftreten können wurden nicht getroffen da dies nicht gefordert wurde. In den Testdaten wurden 2 ganzjährliche Saisonen abgebildet, eine in 2017 und die andere in 2018.

SeasonPlan



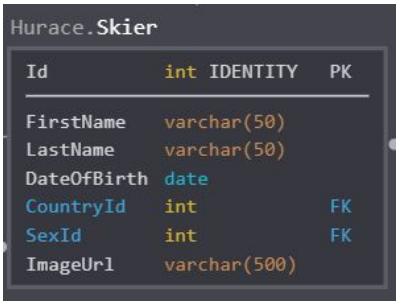
SeasonPlan bildet die Beziehung zwischen Saison (*Season*) und Austragungsort (*Venue*) ab.

Sex



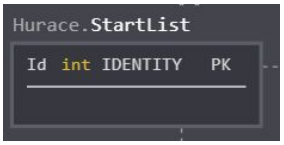
Das Geschlecht einer Person als Enumeration abzubilden, wurde entschieden um auf noch unbekannte und mögliche Änderungen hinsichtlich der gesellschaftlichen Einstellung zum sexuellen Geschlecht, vorbereitet zu sein.

Skier



Enthält alle persönlichen Daten zu den Schifahrern.

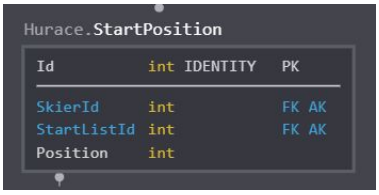
StartList



Fortlaufende Nummer für die Startlisten. Der Grund, warum diese Entität überhaupt existiert, ist die Angabe. Dort ist gefordert, dass eine Entität mit Namen StartList gespeichert werden soll.
Unserer Meinung nach ist es semantisch-gesehen falsch, wenn man die Position direkt im StartList-Tabelle speichert, da ja der Name der Tabelle suggeriert, dass ein Eintrag darin eine eigenständige Startliste darstellt.
In unserem Datenmodell setzt sich somit eine einzelne Startliste nicht durch viele einzelne StartList-Datensätzen zusammen, sondern bildet nur eine einzige StartListe ab.
Die Information über einzelne Startnummern wurde in die StartPosition-Tabelle verschoben.

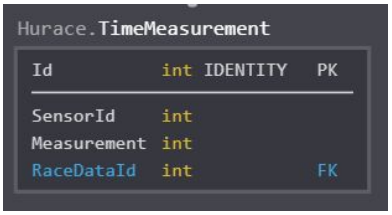
Andererseits lässt sich auch darüber streiten, ob eine Tabelle mit synthetischem Primärschlüssel und keinen weiteren Spalten überhaupt Sinn macht.

StartPosition



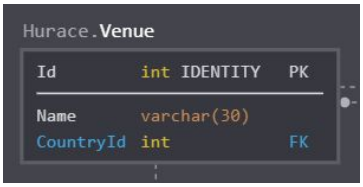
Bildet die Beziehung zwischen Schifahrer und Startliste ab, also an welcher Position ein Schifahrer in einer Startliste aufgelistet ist.

TimeMeasurement



Speichert alle gültigen Zeitmessungen eines Renn-Durchlaufs.

Venue

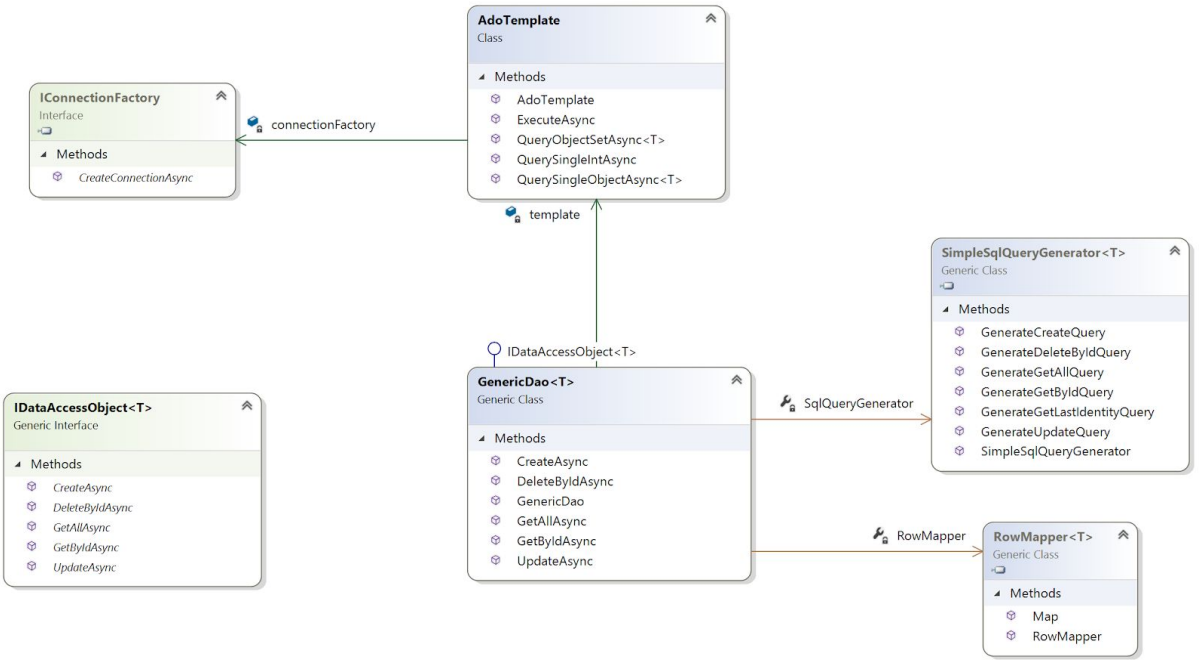


Austragungsorte der Rennen.

Testdatenset

52	Countries
37	Races
3710	RaceData
5	RaceStates
2	RaceTypes
2	Sexes
521	Skiers
74	StartLists
3710	StartPositions
2	Seasons
62	SeasonPlans
18745	TimeMeasurments
31	Venues

Aufbau der Datenzugriffsschicht



SimpleSqlQueryGenerator

Der SimpleSqlQueryGenerator bietet Methoden zur Generierung simpler SQL-Querys abhängig vom Typ der beim Erstellen übergeben wird.
Um SQL-Injection zu verhindern, werden Parameter nicht direkt in die Query eingebettet, sondern über das DbCommand sanitized.

RowMapper

Der RowMapper ist generisch implementiert und kann einzelne Zeilen aus der Datenbank auf Objekte eines Typs mappen, sofern diese kompatibel zueinander sind.

AdoTemplate

Der beim Ausführen von SQL-Abfragen entstehende Boilerplate-Code wird mittels mehrerer Template-Methods im Falle von ADO.NET mit dem AdoTemplate gekapselt.

GenericDao

Das GenericDao verbindet die Funktionalität von SimpleSqlQueryGenerator und RowMapper und bietet dem verwender simple SQL-Abfragen an.
Diese Klasse ist nicht abstrakt, da die angebotenen Queries so einfach gehalten sind dass keine Anpassung auf spezielle Domaintypen notwendig ist.

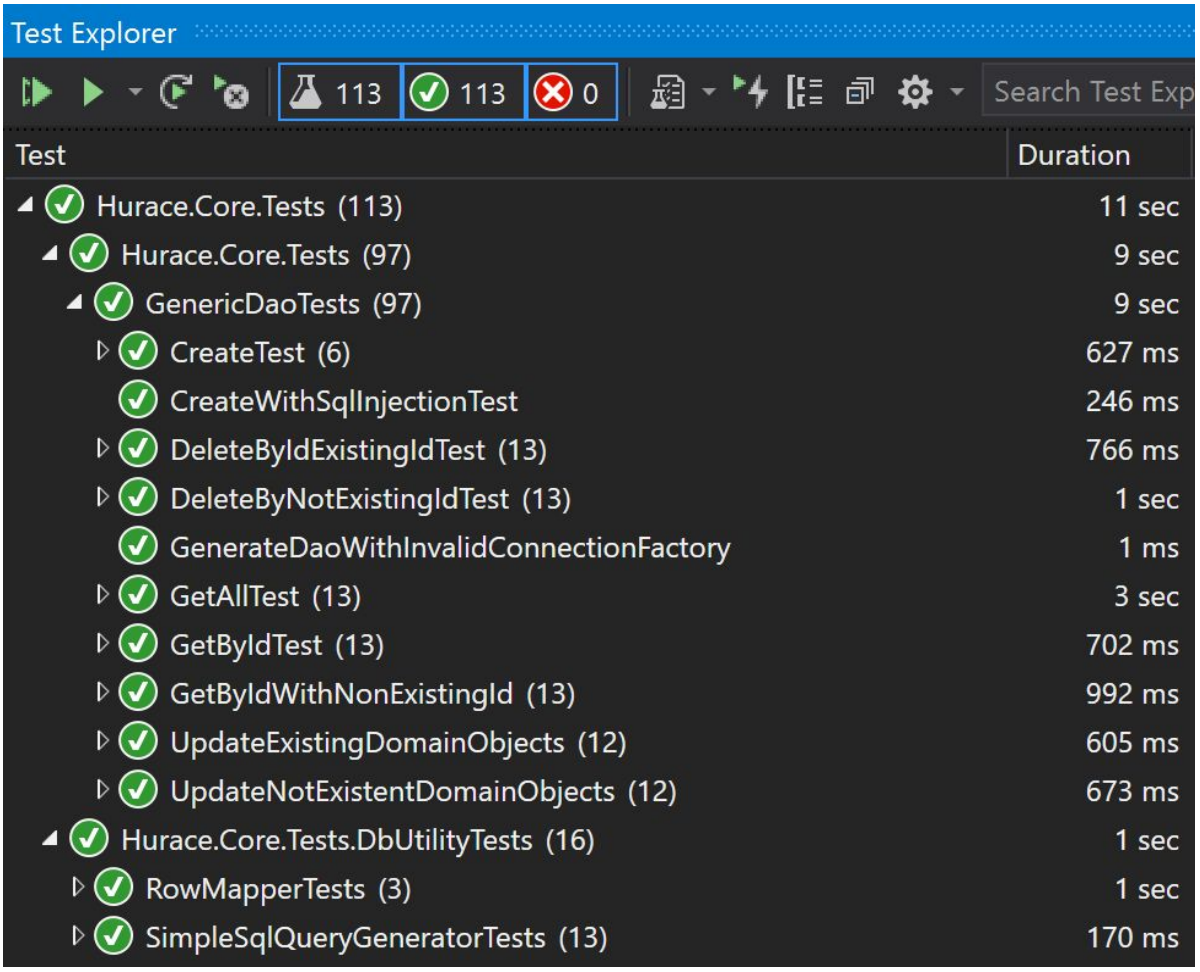
Tests

Für die 3 Hauptkomponenten, das GenericDao, den RowMapper und den SimpleSqlQueryGenerator wurden umfassende UnitTests erstellt.
Es wurden 25 UnitTests mit unterschiedlichen Parametrierungen entwickelt, wodurch insgesamt 113 unterschiedliche Testdurchläufe erzielt werden können.

Ein guter UnitTest ist unabhängig von anderen UnitTests. Dies ist aber bei Tests für die Datenzugriffsschicht nicht selbstverständlich, da eine Datenbank persistent ist.
Es ist auch keine Lösung, die Datenbank transient zu machen, da wir in einer weiteren Abfrage die Ergebnisse auf der Datenbank verifizieren können möchten und bei einer transienten Datenbank die Ergebnisse unmittelbar verloren wären.
Desweiteren ist es auch keine Lösung, das Dao im UnitTest zum Aufsetzen der Testsuite zu verwenden, wenn das Dao selbst getestet werden soll. Das selbe gilt dafür den Boilerplate-Code zum inserten/deleten in die Testsuite zu kopieren, da dieser Code ja auch wieder separat getestet werden müsste, bzw. Angenommen werden würde, dass der Code einfach funktioniert.

Um dieses Problem zu lösen, haben wir uns dazu entschieden, einen TransactionScopes am Beginn eines jeden Unittests der mit der Datenbank zutun hat, anzulegen und nach jedem Unittest diesen TransactionScope wieder zu disposen.
Ein TransactionScope kann mehrere unterschiedliche Transaktionen zu einer großen Transaktion zusammenfassen und wenn eine der einzelnen Transaktionen fehlschlägt, wird für alle zuvor durchgeführten Transaktionen ein *rollback* durchgeführt.
Wenn wir nicht explizit beim TransactionScope angeben, dass diese Transaktion erfolgreich, werden alle darin geschehenen Änderungen zurückgenommen, was uns genau das gewünschte Verhalten bei UnitTests liefert - Persistenz während des UnitTests und transientes Verhalten zwischen verschiedenen UnitTests.

Damit dieses Konzept sich nicht auf das lokale Entwickeln auswirkt (bei dem Persistenz in jeglicher Hinsicht erwünscht ist), werden zwei separate Datenbanken verwendet.
Eine Datenbank ist zum lokalen Entwickeln, dort werden alle Änderungen persistiert. Dies ist auch gleichzeitig eine lokale Replikation des späteren Produktivsystems.
Gleichzeitig gibt es auch eine andere Datenbank, auf der nur UnitTests durchgeführt werden sollen und bei der Änderungen nie länger als die Lebensdauer eines TransactionScopes existieren. Dort existiert nur der Datenzustand, der mit dem Insert-Script beschrieben ist.


Test Explorer
113
113
0
Search Test Exp
Test
Duration
Hurace.Core.Tests (113)
11 sec
Hurace.Core.Tests (97)
9 sec
GenericDaoTests (97)
9 sec
CreateTest (6)
627 ms
CreateWithSqlInjectionTest
246 ms
DeleteByIdExistingIdTest (13)
766 ms
DeleteByNotExistingIdTest (13)
1 sec
GenerateDaoWithInvalidConnectionFactory
1 ms
GetAllTest (13)
3 sec
GetByIdTest (13)
702 ms
GetByIdWithNonExistingId (13)
992 ms
UpdateExistingDomainObjects (12)
605 ms
UpdateNotExistentDomainObjects (12)
673 ms
Hurace.Core.Tests.DbUtilityTests (16)
1 sec
RowMapperTests (3)
1 sec
SimpleSqlQueryGeneratorTests (13)
170 ms