DATABASE ENGINEERING

SAE - INSTITUTE

by Manuel Loof / Xinq / manu@manuel-loof.de

AGENDA

- Vorstellung
- Erwartungen
- Definition Datenbanken
- Relationale Datenbanken
- Datenbanken anprogrammieren
- OR Mapper
- Game Konzept erstellen
- NoSQL Datenbanken

VORSTELLUNG

Manuel Loof

- seit 16 Jahren Software Entwickler
  - .net / C#
  - Datenbanken / SQL / XML / HTML
  - Regex
  - Visual Basic 6.0 / VBA
- seit 10 Jahren Ausbilder
- Leitung Softwareentwicklung Contentmanagement / NWB Verlag
- Indie Spiele Entwickler {Happy Land}

GAME DEVELOPMENT

- Unity 3D
- Blender
- Gimp
- Inkscape / Microsoft Expression Design
- Audacity

EURE ERWARTUNGEN

Meine Erwartungen ;-)

- Verstehen was Datenbanken sind.
- Wie und wo man sie einsetzen kann.
- Eigenständing Abfragen erstellen.
- Datenbanken im Spielekontext.
- Eigenständig Dantenbanken planen und erstellen.

DEFINITION DATENBANKEN

“Eine Datenbank, auch Datenbanksystem (DBS) genannt, ist ein System zur elektronischen Datenverwaltung. Die wesentliche Aufgabe eines DBS ist es, große Datenmengen effizient, widerspruchsfrei und dauerhaft zu speichern und benötigte Teilmengen in unterschiedlichen, bedarfsgerechten Darstellungsformen für Benutzer und Anwendungsprogramme bereitzustellen.”

Quelle: Wikipedia

EINSATZGEBIETE

- Banksysteme
- ERP-Systeme
- Shops
- Wikipedia
- Spiele
- ...

DATENBANKARTEN

- relational
- No SQL
- objektorientiert
- dokumentenorientiert
- hierarchisch
- netzwerkartig

RELATIONALE DATENBANKEN

- Tabellen
- Normal Formen
- Beziehungen
- SQL
- Indizies
- Datenbanksysteme
- Beispiel Programm C#

TABELLE / RELATION

Players

ID	Player	Score
1	Paul	7320
2	Sophia	12000
3	Joe	8420

NORMALFORMEN

Die Normalisierung dient der Sicherstellung eines konsitenten und widerspruchsfreien Datenbestandes.

BEIPSPIEL

Unnormalisierte Tabelle

name	adresse
Heinz Müller	Auf der Klaue 12, 48161 Münster
Heinz Müller	Auf der Klaue 12, 48161 Münster

DIE 1. NORMALFORM

1. Jede Spalte enthält nur unteilbare (atomare, atomische) Werte.
2. Spalten, die gleiche oder gleichartige Informationen enthalten, sind in eigene Tabellen (Relationen) auszulagern.
3. Jede Tabelle enthält einen Primärschlüssel.

TABELLE 1. NORMALFORM

Benutzer

id	vorname	nachname	strasse	hausnummer	plz
1	Heinz	Müller	Auf der Klaue	12	48161

DIE 2. NORMALFORM

1. Die Tabelle erfüllt die 1. Normalform.
2. Alle Informationen in den Spalten, die nicht Teil des Primärschlüssels sind, müssen sich auf den gesamten Primärschlüssel beziehen.

TABELLE 2. NORMALFORM

Benutzer

ID	vorname	nachname
1	Heinz	Müller

Adresse

ID	benutzerID	strasse	hausnummer	plz	stadt
1	1	Auf der Klaue	12	48161	Münster

DIE 3. NORMALFORM

1. Die Tabelle erfüllt die 2. Normalform.
2. Informationen in den Spalten, die nicht Teil des Primärschlüssels sind, dürfen funktional nicht voneinander abhängen.

TABELLEN 3. NORMALFORM

Benutzer

ID	vorname	nachname
1	Heinz	Müller

Adresse

ID	benutzerID	strasse	hausnummer	plz
1	**1**	Auf der Klaue	12	**48161**

PLZ

plz	stadt
48161	Münster

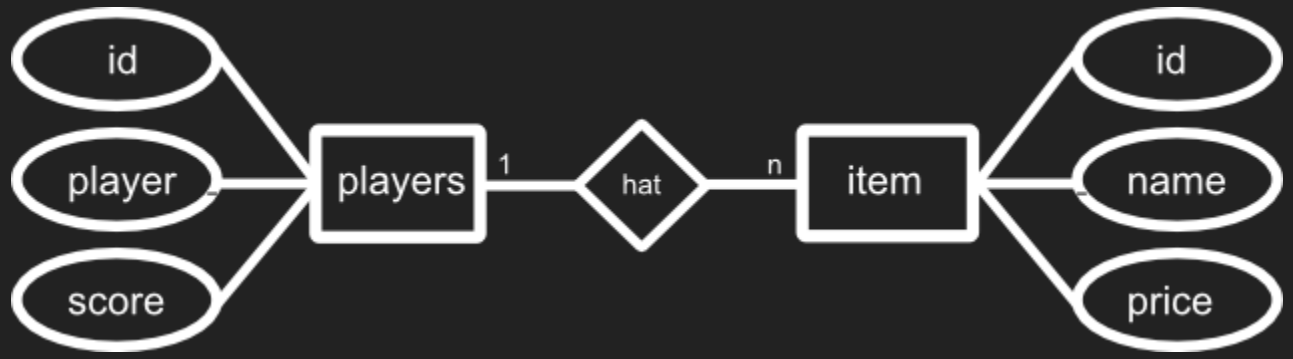
ER-MODELL

„Entity-Relationship-Modell“

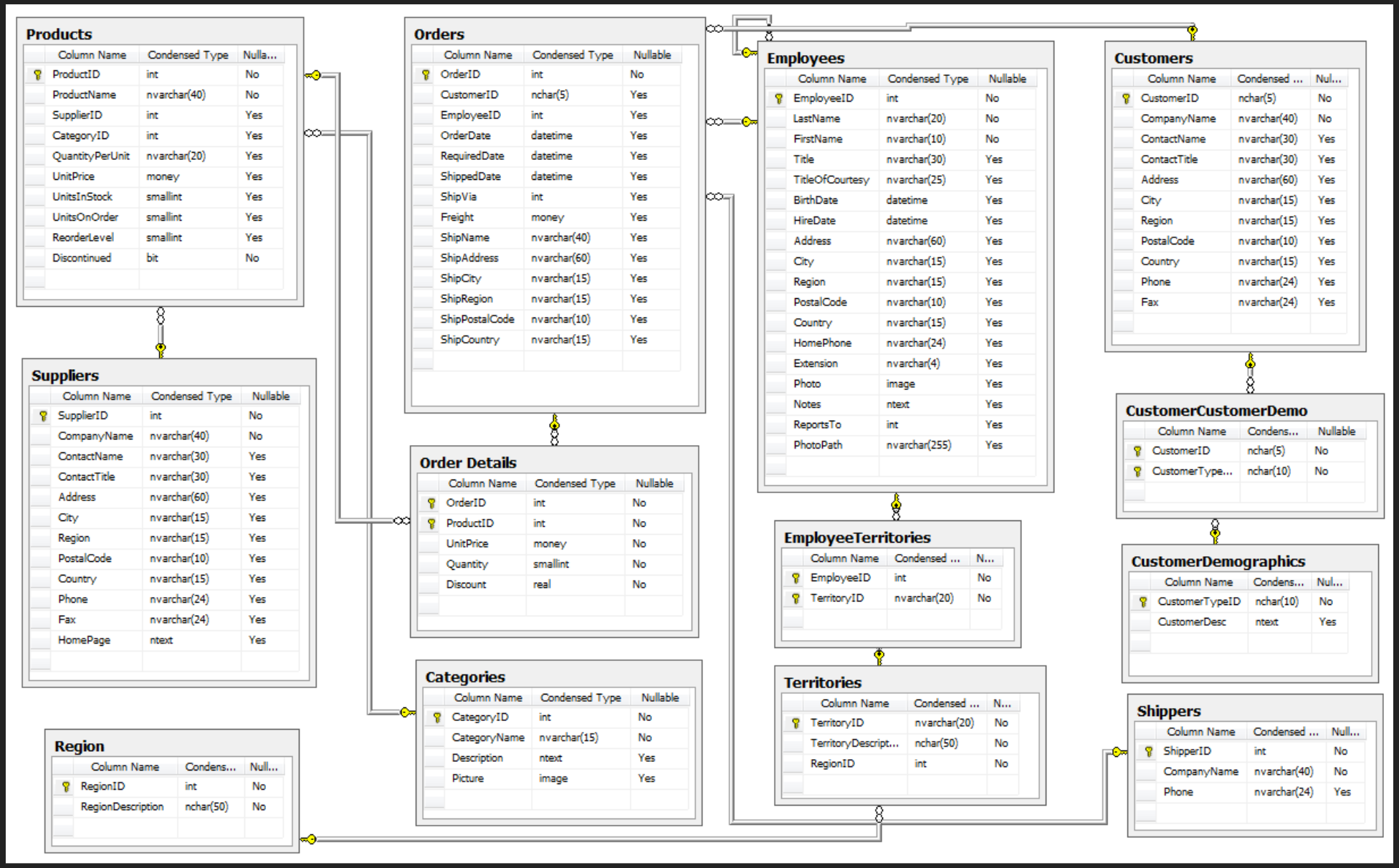
“Ein ER-Modell dient sowohl in der konzeptionellen Phase der Anwendungsentwicklung der Verständigung zwischen Anwendern und Entwicklern (dabei wird nur das Was behandelt, d. h. fachlich-sachliche Gegebenheiten, nicht das Wie, die Technik) als auch in der Implementierungsphase als Grundlage für das Design der – meist relationalen – Datenbank.”

Quelle: Wikipedia

EINFACHE NOTATION



DATENBANKDIAGRAMM NORTHWIND



SQL

„Structured Query Language“

“SQL ist eine Datenbanksprache zur Definition von Datenstrukturen in relationalen Datenbanken sowie zum Bearbeiten (Einfügen, Verändern, Löschen) und Abfragen von darauf basierenden Datenbeständen.”

Quelle: Wikipedia

ABFRAGEN / QUERIES

- Select
- Where
- Order by
- Insert
- Update
- Delete
- Create
- Drop

ABFRAGEN / QUERIES

- Alter
- Beziehungen / Relations
- Joins
- Aggregatfunktionen
- Subqueries
- Group by
- Having
- Union
- Transaktionen

SELECT

Das „select” Statement sagt aus, was für Daten wovon geholt werden sollen.

```
Select * from table;
```

```
Select * from players;
```

ID	Player	Score
1	Paul	7320
2	Sophia	12000
3	Joe	8420

SELECT MIT BESTIMMTEN SPALTEN

```
Select column1, column2 from table;
```

```
Select player, score from players;
```

Player	Score
Paul	7320
Sophia	12000
Joe	8420

SELECT DISTINCT

Durch den Zusatz „distinct” werden identische Werte aus der Spalte zusammen geführt.

```
Select distinct column1 from table;
```

```
Select distinct playerid as PlayerWithItem from relation;
```

Relation

PlayerID	ItemID
1	1
1	2
2	3
3	1

PlayerWithItem
1
2
3

WHERE

Das „where” stellt die Bedingung der Abfrage da.

```
Select * from table where column='value';
```

```
Select * from players where score>10000;
```

ID	Player	Score
2	Sophia	12000

WHERE OPERATORS

=	Equal to	Author = 'Alcott'
<>	Not equal to	Dept <> 'Sales'
>	Greater than	Hire_Date > '2012-01-31'
<	Less than	Bonus < 50000.00
>=	Greater than or equal	Dependents >= 2
<=	Less than or equal	Rate <= 0.05

WHERE OPERATORS

BETWEEN	Between an inclusive range	Cost BETWEEN 100.00 AND 500.00
LIKE	Match a character pattern	First_Name LIKE 'Will%'
IN	Equal to one of multiple possible values	DeptCode IN (101, 103, 209)
IS or IS NOT	Compare to null (missing data)	Address IS NOT NULL

ORDER BY ASC

„Order by" setzt die Reihenfolge des Ergebnisses fest. asc bedeutet aufsteigend.

```
Select * from table order by column;
```

```
Select * from players order by score;
```

ID	Player	Score
1	Paul	7320
3	Joe	8420
2	Sophia	12000

ORDER BY DESC

„Order by" setzt die Reihenfolge des Ergebnisses fest. desc bedeutet absteigend.

```
Select * from table order by column desc;
```

```
Select * from players order by score desc;
```

ID	Player	Score
2	Sophia	12000
3	Joe	8420
1	Paul	7320

INSERT

Mit dem „insert” Befehl werden neue Datensätze angelegt.

```
INSERT INTO table
        (column1, column2, column3)
VALUES
        ('test', 'N', NULL);
```

```
INSERT INTO players
        (player, score)
VALUES
        ('Lilly', '0');
```

ID	Player	Score
1	Paul	7320
2	Sophia	12000
3	Joe	8420
4	Lilly	0

UPDATE

Das „update” Statement ermöglicht die manipulation von Datensätzen. Welche Datensätze verändert werden, wird durch die „where” Klausel bestimmt.
Achtung! Ohne „where” Klausel werden alle Datensätze in der Tabelle manipuliert.

```
UPDATE table
SET column1 = 'updated value'
WHERE column2 = 'N';
```

```
UPDATE players
SET score = '10050'
WHERE id = 1;
```

ID	Player	Score
1	Paul	10050

DELETE

Durch den „delete” Befehl werden Datensätze gelöscht.
***Achtung!** Auch hier wird über die „where” Bedingung wieder festgelegt was gelöscht werden soll. Ohne das Bedingung werden alle Datensätze der Tabelle gelöscht.*

```
DELETE FROM table
WHERE column2 = 'N';
```

```
DELETE FROM players
WHERE id = 3;
```

ID	Player	Score
1	Paul	10050
2	Sophia	12000

CREATE

Der „create” Befehl erzeugt Datenbanken, Tabellen und Indizes.

```
CREATE DATABASE dbname;
```

CREATE

```
CREATE TABLE example(
        column1 INTEGER IDENTITY(1,1),
        column2 VARCHAR(50),
        column3 DATE NOT NULL,
        PRIMARY KEY (column1, column2)
);
```

```
CREATE TABLE players(
        ID INTEGER IDENTITY(1,1),
        player VARCHAR(30),
        score INTEGER,
        PRIMARY KEY (ID)
);
```

DROP

Durch den Befehl „drop” werden Tabellen gelöscht.

```
DROP TABLE table;
```

```
DROP TABLE players;
```

ALTER TABLE

Durch den „alter” Befehl können Tabellen modifiziert werden.

```
ALTER TABLE table_name
ADD column_name datatype
```

```
ALTER TABLE table_name
DROP COLUMN column_name
```

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype
```

BEZIEHUNGEN / RELATIONS

Players

ID	Player
1	Paul
2	Sophia
3	Joe

Item

ID	Name	Price
1	Axt	500
2	Schwert	1200
3	Kettensäge	2400

PlayerItems

PlayerID	ItemI
1	1
1	2
2	3
3	1

VERKNÜPFUNGEN / JOINS

„Joins" erlauben das Verknüpfen von Tabellen.

```
SELECT table1.column1, table1.column2, table2.column1, table2.column2
FROM table1, table2
WHERE table1.column1 = table2.column1;
```

```
SELECT table1.column1, table1.column2, table2.column1, table2.column2
FROM table1 INNER JOIN table2
ON table1.column1 = table2.column1;
```

```
SELECT players.player as playername, playeritems.itemid as itemid
FROM players INNER JOIN playeritems
ON players.id = playeritems.playerid;
```

playername	itemid
Paul	1
Paul	2
Sophia	3
Joe	1

VERKNÜPFUNGEN / JOINS

```
SELECT table1.column1, table1.column2, table2.column1, table2.column2
FROM table1
        INNER JOIN table2
                ON table1.column1 = table2.column1
        INNER JOIN table3
                ON table2.column1 = table3.column1;
```

```
SELECT players.player as playername, item.name as weapon, CONCAT(item
FROM players
        INNER JOIN playeritems
                ON players.id = playeritems.playerid
        INNER JOIN item
                ON item.id = playeritems.playerid;
```

playername	weapon	price
Paul	Axt	500 $
Paul	Schwert	1200 $
Sophia	Kettensäge	2400 $
Joe	Axt	500 $

LEFT OUTER JOIN

Der „left outer join” liefert alle Datensätze von der „linken” Tabelle, inclusive der Datensätze welche in der rechten Tabelle matchen.

```
SELECT table1.column, table2.column
FROM table1 LEFT OUTER JOIN table2
ON table1.column = column2.column;
```

```
SELECT players.player as playername, playeritems.itemid as itemid
FROM players LEFT OUTER JOIN playeritems
ON players.id = playeritems.playerid;
```

playername	itemid
Paul	1
Paul	2
Sophia	3
Joe	1
Lilly	

AGGREGATFUNKTIONEN

SUM, MIN, MAX, COUNT ...

Aggregatfunktionen führen Berechnungen für eine Wertemenge durch und geben einen einzelnen Wert zurück.

```
SELECT Count(table1.column) as NumberOf
FROM table1 LEFT OUTER JOIN table2
ON table1.column = column2.column;
```

```
SELECT Count(players.player) as AnzahlPlayer
FROM players;
```

AnzahlPlayer

3

SUBQUERIES

```
SELECT t.column1, t.column2
FROM table1 as t
WHERE t.column1 in (SELECT table2.column1 FROM table2);
```

```
SELECT *
FROM players
WHERE players.id in (SELECT ID FROM Item);
```

```
SELECT *
FROM players
WHERE players.id in (1,3,4,9);
```

ID	Player	Score
1	Paul	7320
2	Sophia	12000
3	Joe	8420

GROUP BY

Durch „group by” können Datensätze gruppert werden.

```
SELECT column1, aggregate_function(column1)
FROM table1
GROUP BY column1;
```

```
SELECT PlayerID, Count(PlayerID) as AnzahlWaffen
FROM PlayerItems
GROUP BY PlayerID
```

PlayerID	AnzahlWaffen
1	2
2	1
3	1

GROUP BY

```
SELECT PlayerID, Count(PlayerID) as AnzahlWaffen
FROM Players INNER JOIN PlayerItems
ON Players.ID = PlayerItems.PlayerID;
GROUP BY PlayerID
```

PlayerID	AnzahlWaffen
1	2
2	1
3	1

HAVING

„Having" stellt eine Bedingung beim gruppieren da.

```
SELECT column, aggregate_function(column)
FROM table
GROUP BY column
HAVING aggregate_function(column) operator value;
```

```
SELECT PlayerID, Count(PlayerID) as AnzahlWaffen
FROM Players INNER JOIN PlayerItems
ON Players.ID = PlayerItems.PlayerID;
GROUP BY PlayerID
HAVING Count(PlayerID) > 1;
```

PlayerID	AnzahlWaffen
1	2

UNION

Verbindet zwei Select Statements miteinander. ***Achtung!*** *Die Selects müssen die selbe Anzahl an Spalten und diese müssen die identischen Datentypen besitzen.*

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

TRANSACTION

```
BEGIN TRANSACTION name;
```

```
ROLLBACK TRANSACTION name;
```

```
COMMIT TRANSACTION name;
```

DATENTYPEN

integer	Ganze Zahl (positiv oder negativ)
numeric (n, m) oder decimal (n, m)	Festkommazahl (positiv oder negativ) mit insgesamt maximal n Stellen, davon m Nachkommastellen.
float (m)	Gleitkommazahl (positiv oder negativ) mit maximal m Nachkommastellen.
real	Gleitkommazahl (positiv oder negativ).
double oder double precision	Gleitkommazahl (positiv oder negativ).

DATENTYPEN

float und double	sind für technisch-wissenschaftliche Werte geeignet und umfassen auch die Exponentialdarstellung. Wegen der Speicherung im Binärformat sind sie aber für Geldbeträge nicht geeignet, weil sich beispielsweise der Wert 0,10 € (entspricht 10 Cent) nicht exakt abbilden lässt.
character (n) oder char (n)	Zeichenkette Text mit n Zeichen.
varchar (n)	Zeichenkette (also Text) von variabler Länge, aber maximal n druckbaren und/oder nicht druckbaren Zeichen.
text	Zeichenkette (zumindest theoretisch) beliebiger Länge. In manchen Systemen synonym zu clob.
date	Datum (ohne Zeitangabe)

DATENTYPEN

time	Zeitangabe (evtl. inklusive Zeitzone)
timestamp	Zeitstempel (umfasst Datum und Uhrzeit; evtl. inklusive Zeitzone), meistens mit Millisekundenauflösung, teilweise auch mikrosekundengenau
boolean	Boolesche Variable (kann die Werte true(wahr) oder false (falsch) annehmen).
blob (n) oder binary large object (n)	Binärdaten von maximal n Bytes Länge.
clob (n) oder character large object (n)	Zeichenketten mit maximal n Zeichen Länge.

SQL SERVER DATENTYPEN

xml	XML Dokumente. Elemente lassen sich indizieren.
geography	Datentyp für Geographiedaten.

https://msdn.microsoft.com/de-de/library/ms187752%28v=sql.120%29.aspx

LITERATUR EMPFEHLUNGEN

- http://www.w3schools.com/sql/
- Microsoft SQL Server 2012 - Überblick über Konfiguration, Administration, Programmierung

INDIZES

Indizes beschleunigen den Zugriff auf die Daten.

***Achtung!** aber jeder neue Index verlangsamt die Schreibgeschwindigkeit.*

```
CREATE INDEX index_name
ON table_name (column_name)
```

INDIZES

Es gibt **clusterd** und **non custered** Indizes.
Ein **clusterd** Index wird automatisch vom SQL Server bei jeder Tabelle mit Primär Schlüssel angelegt.
Non custered Indizes können auf Spalten angelegt werden.

EIN EIGENES DATENBANK MODELL ENTWICKELN UND UMSETZEN

Hier sind wir alle zusammen gefragt.

Ok, lets do it! ;-)

ADO.NET

ADO.NET ist eine Bibliothek des .NET Framework von Microsoft. Es bietet den Zugriff auf relationale Datenbanken.

ADO.NET

SqlConnection

```
var sqlCon = new SqlConnection(myConnectionString);
```

```
var myConnectionString = "Server=INSPIRON;" +
        "Database=SAE_RolePlayingGame;" +
        "Integrated Security=True";
```

Das SqlConnection Objekt stellt die Verbindung zur Datenbank her.

ADO.NET

SqlCommand

```
var sqlCommand = new SqlCommand("Select * from players;", sqlCon)
```

Mit dem SqlCommand Object führe ich SQL Abfragen aus.

ADO.NET

SqlDataReader

```
SqlDataReader reader = sqlCommand.ExecuteReader()
```

Die Methode SqlCommand.ExecuteReader() liefert mir einen SqlDataReader.
Der SqlDataReader stellt ein Verfahren zum Lesen eines Vorwärtsstreams von Zeilen aus einer SQL Server-Datenbank bereit.
Das bedeutet ich kann sequenziell durch mein Abfrage ergebnis laufen.

ADO.NET

SqlDataReader

```
SqlDataReader.Read()
```

Setzt den SqlDataReader auf den nächsten Datensatz.
Die Standardposition des SqlDataReader ist vor dem ersten Datensatz. Daher müssen Sie Read aufrufen, um auf Daten zugreifen zu können.

ADO.NET

SqlDataReader

```
SqlDataReader[int]
```

```
SqlDataReader[string]
```

Die Indexer liefern die entsprechende Spalte entweder nach der entsprechenden Position oder ihrem Spalten Namen.

OR-MAPPER

Ein „OR-Mapper” (Object-relational mapping) ermöglicht das Abbilden von relationen Daten auf ein Objekt einer objektorienten Programmiersprache.

OR-MAPPER

Vorteile eines OR-Mappers.

- *Programmierung gegen echte Objekte*
- *Keine Magic Strings*
- *Eine höhere Abstraktionsebene*

ENTITY FRAMEWORK

In der .NET Welt ist der meist verwendete OR-Mapper das Entiy Framework.

Es ist in seiner ersten finalen Version im .NET Framework 3.5 (Service Pack 1) im Jahr 2008 veröffentlicht worden. Damals gehörte es noch zu ADO.NET. Seit Version Version 4.1 ist das Entity Framework nicht mehr Bestandteil des Basis .NET Framework und steht als Open Source zur Verfügung. Es kann über NuGet bezogen werden.

ENTITY FRAMEWORK

Es gibt zwei verschiedene Ansätze. „Code First” und „Modell First”.
Beim „Code First” Ansatz, werden zuerst die .NET Klassen erzeugt und beim „Modell First” Ansatz kann auf ein bestehendes Datenbank Modell aufgebaut werden.

EDMX-DATEI

Eine EDMX-Datei ist eine XML-Datei, die ein konzeptionelles Modell, ein Speichermodell und das Mapping zwischen diesen Modellen definiert. Eine EDMX-Datei enthält auch Informationen, die vom ADO.NET Entity Data Model Designer (Entity Designer) verwendet werden, um ein Modell grafisch zu rendern. Die empfohlene Vorgehensweise für das Erstellen einer EDMX-Datei ist die Verwendung des Assistenten für Entity Data Model.

NEUE EDMX DATEI ANLEGEN.

- *Hinzufügen*
  - *Neues Element*
- *ADO.NET Entity Data Model*

DBCONTEXT - KLASSE

Eine DbContext-Instanz stellt eine Kombination des Arbeitseinheitsmusters und des Repositorymusters dar, die zum Abfragen einer Datenbank und Gruppieren von Änderungen verwendet werden kann, die dann als Einheit in den Speicher zurückgeschrieben werden. DbContext gleicht konzeptionell ObjectContext.

DBSET<TENTITY>-KLASSE

Eine DbContext-Instanz stellt eine Kombination des Arbeitseinheitsmusters und des Repositorymusters dar, die zum Abfragen einer Datenbank und Gruppieren von Änderungen verwendet werden kann, die dann als Einheit in den Speicher zurückgeschrieben werden. DbContext gleicht konzeptionell ObjectContext.

DATEN LESEN

```
using (var db = new SAE_RolePlayingGameEntities())
{

        var players = from p in db.Players
                      where p.ID == 1
                      select p;

        players.ToList().ForEach(p => Console.WriteLine(p.name));

}
```

DATEN LESEN MIT EINEM JOIN

```
using (var db = new SAE_RolePlayingGameEntities())
{

        var players = from p in db.Players
                      where p.ID == 1
                      select p.Rassen;

        players.ToList().ForEach(r => Console.WriteLine(r.name));

}
```

DATEN MANIPULIEREN

```
using (var db = new SAE_RolePlayingGameEntities())
{

        var players = from p in db.Players
                      where p.ID == 1
                      select p;

        var player = players.FirstOrDefault();

        player.name = "MANU";

        Console.WriteLine(db.SaveChanges());

}
```

DATEN LÖSCHEN

```
using (var db = new SAE_RolePlayingGameEntities())
{

        var players = from p in db.Players
                      where p.ID == 1009
                      select p;

        var player = players.FirstOrDefault();

        db.Players.Remove(player);

        Console.WriteLine(db.SaveChanges());

}
```

DATEN HINZUFÜGEN

```
using (var db = new SAE_RolePlayingGameEntities())
{

        var player = new Players();

        player.energie = 100;
        player.ID = 1009;
        player.klasseID = 1;
        player.level = 80;
        player.muenzen = 0;
        player.name = "Manuel";
        player.password = "cbb786c";
        player.rasseID = 1;

        db.Players.Add(player);
```

RELATIONALE DATENBANKSYSTEME

- SQL Server
- Oracle
- MySQL
- ...

MICROSOFT SQL SERVER

Download

NOSQL

NoSQL *(englisch für Not only SQL) bezeichnet Datenbanken, die einen nicht-relationalen Ansatz verfolgen und damit mit der langen Geschichte relationaler Datenbanken brechen. Diese Datenspeicher benötigen keine festgelegten Tabellenschemata und versuchen, Joins zu vermeiden, sie skalieren dabei horizontal. Im akademischen Umfeld werden sie häufig als „strukturierte Datenspeicher“ (engl. structured storage) bezeichnet.*

Quelle: Wikipedia

RAVENDB

- *in .net gemacht*
- *Schema free*
- *free to use*
- *High Performance*
- *Safe by default*

DOCUMENTSTORE

```
string documentStoreLocation = @"http://localhost:8080";
string testDatabase = @"ravenTest";

var documentStore = new DocumentStore { Url = documentStoreLocation,
documentStore.Initialize();
```

IDOCUMENTSESSION

```
using (var documentStore = new DocumentStore { Url = documentStoreLoc
{
        documentStore.Initialize();

        using (IDocumentSession session = documentStore.OpenSession()
        {
                        var userI = new User() { FirstName = "Daniel"
                        var userII = new User() { FirstName = "Manuel

                        session.Store(userI);
                        session.Store(userII);

                session.SaveChanges();
        }
}
```

DATEN LADEN

```
using (var documentStore = new DocumentStore { Url = documentStoreLoc
{
        documentStore.Initialize();

        IList<User> blogs = new List<User>();
        using (IDocumentSession session = documentStore.OpenSession()
        {
                blogs = session.Query<User>().Where(u => u.FirstName
        }

        return blogs;

}
```

DATEN LÖSCHEN

```
using (var documentStore = new DocumentStore { Url = documentStoreLoc
{

        documentStore.Initialize();

        IList<User> blogs = new List<User>();
        using (IDocumentSession session = documentStore.OpenSession()
        {
                blogs = session.Query<User>().Where(u => u.FirstName

                foreach(var blog in blogs)
                {
                        session.Delete(blog);
                }
```

DATEN MANIPULIEREN

```
using (var documentStore = new DocumentStore { Url = documentStoreLoc
{

        documentStore.Initialize();


        IList<User> blogs = new List<User>();
        using (IDocumentSession session = documentStore.OpenSession()
        {
                blogs = session.Query<User>().Where(u => u.FirstName


                int i = 1;
                foreach (var blog in blogs)
                {
```

TEST DATA

- https://www.mockaroo.com/
- http://www.generatedata.com/
- http://www.databasetestdata.com/
- AdventureWorks

DOWNLOADS

- Visual Studio Express
- SQL Server Express

EINSTELLUNGEN

Themes umstellen:

Black (default) - White - League - Sky - Beige - Simple Serif - Blood - Night - Moon - Solarized