Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Dennis Haseloff

Entwicklung einer Anwendung zur Auswertung von Statistiken im Golf mit Hilfe von 'Online Analytical Processing' und Oracle Spatial

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Dennis Haseloff

Entwicklung einer Anwendung zur Auswertung von Statistiken im Golf mit Hilfe von 'Online Analytical Processing' und Oracle Spatial

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Wolfgang Gerken
Zweitgutachter: Prof. Dr. Olaf Zukunft

Eingereicht am: 11. Juni 2015

Dennis Haseloff

Thema der Arbeit
Entwicklung einer Anwendung zur Auswertung von Statistiken im Golf mit Hilfe von 'Online Analytical Processing' und Oracle Spatial

Stichworte
OLAP, Oracle, Datenbank, Golf, Statistik, Oracle spatial, SDO GEOMETRY

Kurzzusammenfassung
Dieses Dokument beschreibt die Funktionsweise von „Online Analytical Processing"und der Verarbeitung von geometrischen Daten mit Oracle. Als Beispielanwendung wird in dieser Arbeit eine Applikation entwickelt, die einem Golfspieler mit Hilfe von Geodaten auf dem Golfplatz als digitaler Caddy dienen soll
Dennis Haseloff

Title of the paper
Developing an application for the analysis of statistics in golf with the help of 'Online Analytical Processing' and Oracle Spatial

Keywords
OLAP, Oracle, database, golf, statistics, Oracle spatial, SDO GEOMETRY

Abstract
This document describes the operation of „online analytical processing"and the processing of geometric data with Oracle. As a prototype in this paper an application is developed, which uses spatial data to serve a golfer as a digital caddy on the golf course

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1. Einleitung

1.1. Motivation

In vielen Sportarten geht es um Statistiken. Besonders für die Amerikaner ist es besonders wichtig, immer genau zu wissen, welcher Sportler in seiner Sportart weshalb der Beste ist. Sind es im Fußball die Tore und Torvorlagen, die die Qualität eines Spielers ziemlich genau ausmachen, so sind es in anderen Sportarten viel mehr Faktoren, die beziffern, welches Leistungsniveau ein Sportler gerade hat. In besonders komplexen Sportarten, wie Baseball und Golf, spiegeln Statistiken am genauesten den aktuellen Leistungsstand des Aktiven dar. So wird im Baseball nach Auswertung von Statistiken immer genau derjenige Schlagmann eingesetzt, der gegen den aktuellen Pitcher des gegnerischen Teams die beste Trefferquote hat. Anders als beim Mannschaftssport Baseball verhält es sich in der Individualsportart Golf. Dort ist der Gegner nicht das gegnerische Team, sondern der Platz, beziehungsweise das Loch, auf dem der Golfer gerade steht. So kann ein Golfer die Statistiken nutzen, um genau zu wissen, welchen Schläger er zu jedem beliebigen Zeitpunkt benutzen muss. Dieses Wissen kann er dann verwenden, um Hindernissen wie Seen, Bunkern und Ausgrenzen aus dem Weg zu gehen. Genau an dieser Stelle soll die in dieser Arbeit zu entwickelnde Anwendung ansetzen. Sie soll dem Hobbygolfer helfen, mit der Benutzung von Statistiken sein Spiel zu verbessern und unnötige Fehler zu vermeiden.

1.2. Zielsetzung

Im Rahmen dieser Bachelorarbeit soll im Kontext einer prototypischen Applikation aufgezeigt werden, dass die Verwendung von OLAP in Verbindung mit Geodaten sich dazu eignet, Statistiken des Golfspiels auszuwerten. Außerdem sollen die Statistiken dazu genutzt werden, dem Spieler auf seiner Golfrunde eine *just-in-time*-Hilfestellung bei Schlägen zu geben. Diese Funktionalität soll über eine 'Web-App' gelöst werden, auf der dann die von OLAP ausgewerteten Daten grafisch dargestellt werden. Die Anwendung soll dem Benutzer aufgrund der erhobenen Daten einen Vorschlag für den nächsten Schlag machen können, indem die Anwendung die aus dem *Data-Warehouse* durch OLAP verarbeiteten Daten analysiert und dem

Benutzer darstellt. Zur Umsetzung dieser Funktionen reichen der gewählte Schläger und das Handicap eines Spielers nicht aus. An dieser Stelle werden dann weiterführende Informationen über einen Schlag benötigt, wie die genaue Position des Balles, die Ausmaße eines Grüns oder auch die Position von Hindernissen. Besonders die exakte Position des Balles auf einem Loch ist sehr wichtig für die Entwicklung eines Vorschlags des weiteren Vorgehens auf dem Loch. An dieser Stelle wird ein auf einer Datenbank basierendes geografisches Informationssystem benötigt. Durch die Auswertung von Positionen direkt in der Datenbank, mit Hilfe von Aggregatfunktionen, wie der Berechnung der Distanz zwischen zwei Objekten, können diese relevanten Informationen leichter und effizienter dem Benutzer dargestellt werden.

1.3. Gliederung

Um eine klare theoretische Basis für OLAP und ORACLE™ Spatial zu schaffen, wird das Kapitel 2 auf die Grundlagen dieser beiden Einzelthemen eingehen. Dabei wird der Schwerpunkt auf dem *Online Analytical Processing* für Oracle und dem Datentyp $SDO\_GEOMETRY$ liegen.
Das Kapitel 3 beschäftigt sich mit der Gewinnung der Geodaten des Golfplatzes mit Hilfe von Google Earth (Steven J. Whitmeyer und Ornduff, 2012). Hierbei wird großer Wert auf die Struktur der produzierten KML (Google-Maps-Group, 2014) Dateien gelegt, da aus diesen Daten das *data warehouse* als Grundlage der OLAP-Anwendung initialisiert wird.
Die folgenden Abschnitte im Kapitel 4 beschäftigen sich mit dem Entwurf und dem Design der Datenbank sowie mit der Analyse zur Entwicklung der Anwendung, welche die Schnittstelle zur Datenbank, beziehungsweise zum OLAP Prozess, darstellt. Zu diesem Schritt gehört auch die Einführung von Dimensionen in dem gebildeten Kontext, sowie deren Level und Hierarchien. Auch das eigentliche Zusammenstellen von OLAP-Cubes wird in diesem Kapitel abgehandelt, da dies einen konzeptionellen Entwurf der Datenstruktur darstellt.
Das Kapitel der Implementierung 5 schließt dann die Entwicklungsphase der Anwendung mit dem Erstellen der OLAP Schnittstelle zur Web-Applikation ab. In diesem Schritt werden auch die OLAP-Cubes durch SQL-Anfragen ausgewertet. Die entwickelte Logik in der Applikation stellt dann ein *Decision Support System* (Power, 2004) für den Golfspieler auf dem Golfplatz zur Verfügung.
Zum Abschluss fasst das Kapitel 6 die Ergebnisse dieser Arbeit kurz zusammen und gibt einen Ausblick auf mögliche Erweiterungen und Verbesserungen des entstandenen Systems sowie ein abschließendes Fazit der wissenschaftlichen Aspekte dieser Arbeit.

2. Grundlagen

2.1. OLAP

Der Begriff OLAP wurde erstmals von S.B. Codd, E.F. Codd[1] und C.T Salley in einem White Paper (E. Codd, S. Codd und Salley, 1993) erwähnt. Die allgemein gültige Abkürzung steht für *On-Line Analytical Processing*. Das Paper beschäftigt sich mit der damals neuen Frage, ob klassische relationale oder auch objektrelationale Datenbanken, bzw. deren Abfragesprache SQL, eine mehrdimensionale Datenanalyse leisten können. In früheren Herangehensweisen wurde OLAP oft mit der mehrdimensionalen Datenanalyse gleichgesetzt. Die Autoren des Papers sahen die mehrdimensionale Datenanalyse aber vielmehr als ein Feature von OLAP neben vielen weiteren.
Im heutigen Sprachgebrauch wird der Begriff OLAP oft bei sehr technischen Darstellungen einer Lösung verwendet. Analog dazu wird der Begriff der Mehrdimensionalität dafür verwendet, um den Sachverhalt eines Problems zu charakterisieren. Trotz dieser logischen Abgrenzung ist eine trennscharfe begriffliche Abgrenzung der beiden Begriffe nahezu unmöglich.
Mitte der achtziger Jahre wurde das Interesse von Managern nach Werkzeugen zur Analyse von Unternehmensdaten immer stärker und die bis dato eingesetzten MIS oder EIS[2] konnten zwar schnell und einfach Datenanalysen durchführen, doch war es den Managern nicht möglich, durch einfache Anweisungen auf Änderungen in Unternehmensstrukturen zu reagieren.

2.1.1. Möglichkeiten und Grenzen von SQL

Aufgrund der neuen Anforderungen an Messinstrumente in der Datenanalyse legen Codd und Salley besonderen Wert darauf, den Sinn von solchen neuen Herangehensweisen damit zu begründen, dass die für den Einsatz in RDBMS[3] entwickelte Abfragesprache SQL[4] durch sich selbst limitiert ist. SQL wurde entwickelt, um möglichst einfache und natürlichsprachliche

[1] *Erfinder*des Konzepts der relationalen Datenbanken
[2] Management oder Executive Information System
[3] Relationale Datenbank Management Systeme
[4] Structured Query Language

Anfragen an eine Datenbank zu stellen, das heißt also, dass für jede neue Sicht (Dimension) prinzipiell eine neue SQL Anfrage geschrieben werden muss. Das setzt für den Endbenutzer dann einen erheblich höheren Wissensstand der Datenbasis voraus, als es ein Manager in einem Unternehmen haben kann.
Als erklärendes Beispiel wird eine Datenbank mit Geschäftsdaten herangezogen, für die mit Hilfe von SQL Abfragen eine Quasi-Mehrdimensionalität erzeugt werden soll 2.1.

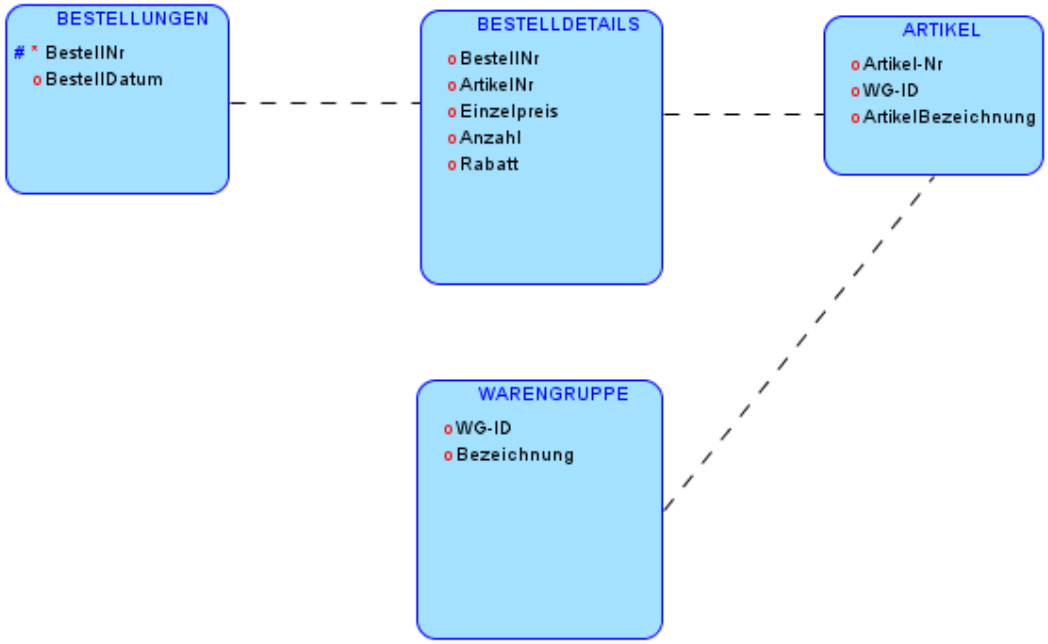

Abbildung 2.1.: Geschäftsdaten ERM

Um die nun entstehende Komplexität zu betrachten, stellt man der Datenbank die Frage, wie sich der Umsatz im Zeitraum 01.01.2012 - 31.12.2012 auf die Dimensionen *Bestelljahr*, *Quartale* und *Warengruppe* verteilt. Aus dieser Fragestellung entsteht, basierend auf Standard-SQL, folgende Anweisung:

```
1 SELECT
2         Bezeichnung as WarengruppenBezeichnung,
3         Year(Bestelldatum) as BestellJahr,
4         'Q' cast(DatePart(q.Bestelldatum) as nvarchar(1)) as Quartale,
5         cast(Sum(Anzahl*Einzelpreis*(1 Rabatt)) as decimal(10,2)) as Umsatz
6 FROM Bestellungen
```

```
7           INNER JOIN Bestelldetails ON
8                   Bestellungen.BestellNr = Bestelldetails.BestellNr
9           INNER JOIN Artikel ON
10                  Bestelldetails.ArtikelNr = Artikel.Artikel Nr
11          INNER JOIN Warenbgruppe ON
12                  Artikel.WG ID = Warengruppe.WG ID
13  GROUP BY
14          Warengruppe.Bezeichnung,
15          Year(Bestelldatum),
16          'Q' cast(DatePart(q.Bestelldatum) as nvarchar(1))
17  ORDER BY
18          Warengruppe.Bezeichnung,
19          Bestelljahr,
20          Quartale
```

Listing 2.1: SQL Anweisung zur Fragestellung

Die Abbildung 2.2 zeigt jetzt nach Anweisung 2.1 einen vollständigen Überblick auf die Umsätze im Jahr 2012. Sie liefert jedoch viele für die Analyse unnötige Daten, weil für jede Kombination der Dimensionen eine neue Zeile angelegt wird. Die von Pito Salas[5] entwickelten Pivot-Tabellen (Jelen und Alexander, 2005) schaffen hierbei Abhilfe. Mittlerweile gehört es zum Standard, diese Art von Daten in diesen speziellen Tabellen darzustellen. In 2.3 wird aufgezeigt, dass eine *PivotTable* die Struktur der Daten soweit vereinfacht, dass ein Analyst sofort einsehen kann, was im Q1/2012 an den Festplatten für ein Umsatz gemacht worden ist. An diesem Beispiel ist leicht zu erkennen, dass *PivotTables* die idealen Browser für mehrdimensionale Darstellungen sind.

An diesen Überlegungen ist zu erkennen, dass es prinzipiell möglich ist, mit SQL-Anweisungen Sichten auf Daten zu erzeugen, die einen mehrdimensionalen Hintergrund haben. Wenn man jetzt aber bedenkt, dass sich die Dimensionen durchaus schnell verändern können, erweist sich die Herangehensweise mit Standard-SQL als nicht praktikabel. Stellt man sich vor, dass der Analyst dieser Daten jetzt noch wissen möchte, wie sich der Verkauf in den einzelnen Monaten dargestellt hat, ist es aus dieser Sicht der Daten nicht zu erkennen und es muss eine komplett neue Anfrage an die Datenbank abgesetzt werden. Dies erzeugt dann einen Mehraufwand des Unternehmens, da hierfür wieder IT-Fachleute mit fundiertem SQL Wissen benötigt werden.

Dennoch ist zu erwähnen, dass auch Standard-SQL mittlerweile so weit entwickelt ist, aus einfachen *SELECT* Ausgaben einen mehrdimensionalen *Cube* zu erstellen. Durch den *Cube* operator, der im *GROUP BY* Segement der SQL Anweisung verwendet wird, kann aus der

[5] Amerikanischer Software Entwickler, Vater der *PivotTables*

	Warengruppe	Bestelljahr	Quartale	Umsatz
1	Festplatte	2012	Q1	20000.34
2	Festplatte	2012	Q2	14753.21
3	Festplatte	2012	Q3	18346.23
4	Festplatte	2012	Q4	35967.12
5	Speicher	2012	Q1	19365.12
6	Speicher	2012	Q2	18456.87
7	Speicher	2012	Q3	20123.45
8	Speicher	2012	Q4	23056.12
9	Kabel	2012	Q1	5333.12
10	Kabel	2012	Q2	3456.56
11	Kabel	2012	Q3	6634.21
12	Kabel	2012	Q4	4562.34
13	Netzteile	2012	Q1	9992.21
14	Netzteile	2012	Q2	8345.21
15	Netzteile	2012	Q3	6194.32
16	Netzteile	2012	Q4	7345.99

Abbildung 2.2.: Übersicht des Artikelumsatz auf die verschiedenen Quartale Warengruppen und Bestelljahre

Bezeichnung	UmsatzSum
Speicher	89099.6
Kabel	19986.23
Netzteile	31877.73
Festplatte	81001.56

Tabelle 2.1.: Ausgabe der SQL-Anweisung 2.2 auf eine Tabelle mit dem Inhalt aus Abbildung 2.2

reinen tabellarischen Sicht eine *View* gemacht werden, die bereits Informationen zusammenfasst. So würde die SQL Anweisung 2.2 aus den Daten aus Abbildung 2.2 die Ausgabe aus Tabelle 2.1.1 machen.

```
SELECT
        CASE
                WHEN
                        (GROUPING(Bezeichnung) = 1) THEN 'ALL'
                ELSE
                        ISNULL(Bezeichnung, 'UNKNOWN')
        END AS Bezeichnung,
        SUM(Umsatz) AS UmsatzSum
FROM
        Verkauefe
GROUP BY
```

```
12          Bezeichnung WITH CUBE
```

Listing 2.2: Beispiel des Cube Operator

	A	B	C	D	E	F
1		Q1	Q2	Q3	Q4	Grand Total
2	Festplatte	20000.34	14753.21	18346.23	35967.12	89066.9
3	Kabel	5333.12	3456.56	6634.21	4562.34	19986.23
4	Netzteile	9992.21	8345.21	6194.32	7345.99	31877.73
5	Speicher	19365.12	18456.87	20123.45	23056.12	81001.56
6	Grand Total	54690.79	45011.85	51298.21	70931.57	221932.42

Abbildung 2.3.: PivotTabelle der SQL Abfrage2.1

An diesen Einschränkungen von SQL erkennt man, dass die von Codd und Salley entwickelten OLAP-Werkzeuge an genau dieser Stellen benötigt werden, um den Benutzer die Daten in einer leicht interpretierbaren Form auszugeben.

2.1.2. Dimensionen

In Kapitel 2.1 wurde der Begriff der Mehrdimensionalität eingeführt. Um diesen zu verstehen, muss man wissen, dass eine Analyse der Unternehmenskenndaten sich nicht oder nicht nur mit Detaildaten beschäftigt, sondern mit sogenannten aggregierten Daten verschiedenster Ebene. Das sind Kennziffern, die vorher vom Unternehmen festgelegt worden sind, die bei Geschäftsvorfällen aufgezeichnet wurden. Diese konsolidierten Daten werden dann z.B. einem Controller ausgegeben, der dann seine Fragestellung auf diese anwenden kann. Eine solche Fragestellung kann die aus 2.1.1 sein. Wie schon in 2.1.1 erwähnt, kann sich die Fragestellung jederzeit ändern, dies setzt eine Konsolidierung der Daten verschiedenster Art und Weise voraus.

> Konsolidierung von Daten ist der Prozess, in dem Teile von Informationen zu einzelnen Blöcken relevanten Wissens zusammengeführt werden. Die höchste Ebene im Pfad einer Datenkonsolidierung wird als Dimension dieser Daten bezeichnet (Brosius et al. 2009, S. 41)

So können die Umsätze nach verschiedensten Kenndaten konsolidiert werden, beispielsweise nach der Zeit, der Warengruppe oder auch dem Lieferziel. Diese Dimensionen weisen dann eine ganze Anzahl von Ebenen (level) auf, so kann der Konsolidierungspfad der Zeit dann Tag - Stunde - Minute, oder der des Lieferziels Kontinent - Land - Stadt an Ebenen aufweisen. Diese Ausprägungen der Dimensionen werden als Elemente (members) bezeichnet.

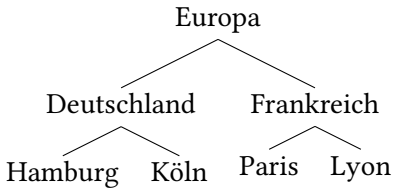

Abbildung 2.4.: Beispiel der Hierarchien

2.1.3. Measures

Als *Measures* werden die Werte bezeichnet, auf die mit Hilfe der Dimensionen zugegriffen werden soll. Im Beispiel 2.2 kann der Umsatz als *Measure* bezeichnet werden. Eine hinreichende Bedingung für eine multidimensionale Datenmenge ist, dass sie mindestens ein *Measure* aufweisen muss.
Da es aber sinnlos erscheint, einem Controller die reinen Detaildaten vorzuhalten, werden vor dem Darstellen in aller Regel sogenannte Aggregatfunktionen auf diese angewendet. Die leistungsfähigste dieser Funktionen ist die Summierung[6], wie sie in Abbildung 2.3 zu sehen ist.

2.1.4. Hierarchien

Die Elemente der Dimensionen können hierarchisch aufgebaut sein. So kann die geografische Dimension die Hierarchieebenen *Kontinent - Land - Stadt* 2.1.4 aufweisen.

2.1.5. Cubes

Dimensionen und *Measures* definieren die Struktur und den Inhalt einer multidimensionalen Datenmenge. Diese Datenmenge wird im Allgemeinen als *Cube*[7] bezeichnet. Im allgemeinen Verständnis ist jetzt anzunehmen, dass ein *Cube* als geometrische Form nur drei Dimensionen enthalten kann und am Anfang der Entwicklung von OLAP Werkzeugen war es auch so gedacht. Doch es ist nicht so, ein *Cube* kann aus beliebig vielen Dimensionen bestehen (Scholl, 2008, S. 30). Diese *Multi-Cubes* können dann allerdings nicht mehr grafisch dargestellt werden, was eigentlich ein großer Vorteil von OLAP ist. Denn das Verständnis einer Abfrage mit 3 Dimensionen kann sich der menschliche Verstand gut visualisieren. So zeigt Abbildung 2.5 (Oracle-OLAP-Group, 2008, S. 42) ein Beispiel eines 3-dimensionalen *Cubes*, welcher die

[6] Bekannt durch das *SUM(x)* im Standard-SQL
[7] Das deutsche Wort Datenwürfel wurde von Microsoft eingeführt, aber nach kurzer Zeit wieder verworfen.

Dimensionen $source$, $route$ und $time$ besitzt. Diese Dimensionen sind wiederum in die Hierarchien aus Abbildung 2.7, Abbildung 2.6 und Abbildung 2.8 aufgeteilt. So kann der Benutzer jetzt auf den ersten Blick sehen, dass der Umsatz im ersten Quartal in Afrika auf dem Luftweg 190 Einheiten betrug und am 17. Februar 1999 gemessen worden ist.

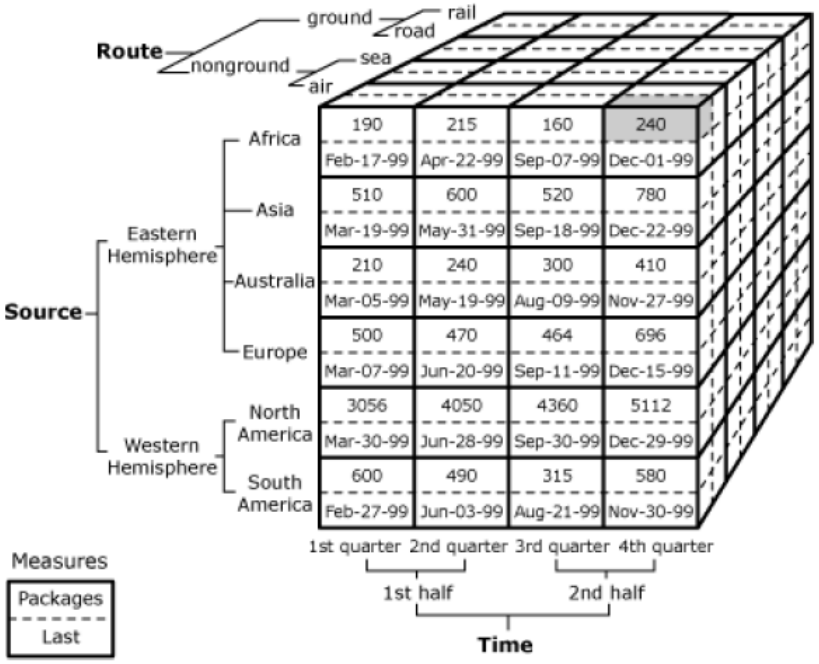

Abbildung 2.5.: Darstellung eines *Cubes*

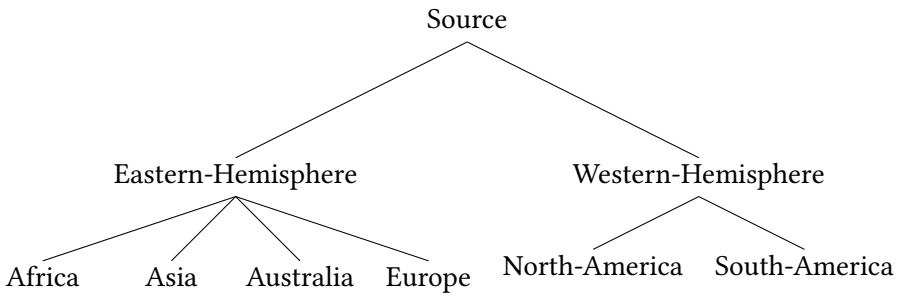

Abbildung 2.6.: Hierarchien der Dimension *Source* aus Abbildung 2.5

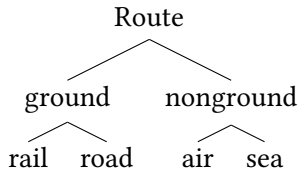

Abbildung 2.7.: Hierarchien der Dimension *Route* aus Abbildung 2.5

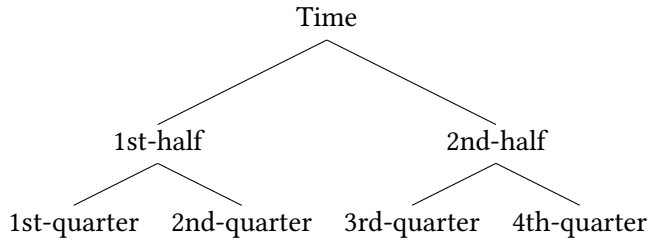

Abbildung 2.8.: Hierarchien der Dimension *Time* aus Abbildung 2.5

2.1.6. Dimensions- und Faktentabellen

Um die *Cubes* mit Daten zu versorgen, müssen jenen jetzt die Dimensionen mit ihren Elementen und die dazugehörigen *Measures* übergeben werden. In kleineren Desktopanwendungen kann es sein, dass eine dieser Eingaben manuell vom Benutzer getätigt wird, jedoch verwenden fast alle leistungsfähigen OLAP-Dienste eine relationale Datenbank als Datenspeicher. Für diese Datenbank gelten drei Regeln (Scholl, 2008, S. 45):

1. die Werte der *Measures* sind in einer Tabelle gespeichert, die als Faktentabelle bezeichnet wird
2. die Elementwerte jeder Dimension werden in einer sogenannten Dimensionstabelle gespeichert
3. der Primärschlüssel jeder Dimensionstabelle erscheint als Fremschlüssel in der Faktentabelle

2.2. ORACLE™ Spatial

ORACLE™ Spatial, kurz Spatial genannt, soll die Verarbeitung und Handhabung räumlicher Daten für den Nutzer und Anwendungen wie Geographische Informationssysteme (GIS) erleichtern (Weigl, 2011). Es ist eine Sammlung an Funktionen und Prozeduren, die es ermöglichen, räumliche Daten zu speichern, zurückzugewinnen, upzudaten und Abfragen darauf

auszuführen.
Es besteht aus 4 Komponenten:

1. Schema der Beschreibung der Speicherung, Syntax und Semantik der unterstützten Datentypen
2. räumliche Indizierungsmechanismen
3. Sammlung von Operatoren und Funktionen, um Spatial-Join und *area-of-interest*-Queries auszuführen
4. administrative Werkzeuge

Um nun geografische Daten in einer Datenbank zu speichern, bietet Oracle zwei Varianten an:

1. Die objekt-relationale Variante mit dem Datentyp *SDO_GEOMETRY*
2. Die relationale Variante mit einer vordefinierten Anzahl von Spalten des Typs NUMBER und einer oder mehr Zeilen pro Objekt

Diese Arbeit wird sich nur mit dem objekt-relationalen Ansatz beschäftigen, da dieser in *Oracle 11g*[8] als Standard empfohlen wird (Murray, 2011, S. 30).

2.2.1. Geometrische Figuren

In ORACLE™ Spatial gelten drei primitive Datentypen (Abbildung 2.9) als Grundlage für den Aufbau von geometrischen Figuren.

1. Point, (x,y)-Koordinaten
2. Line-String, ein oder mehrere zusammengesetzte Paare von Punkten
3. Polygon, zusammengesetzte Linien, die eine durch sie definierte Fläche umschließen

Im Allgemeinen werden diese drei Grundtypen als *Elemente* bezeichnet. Wie in Abbildung 2.9 sehen ist, unterliegen diese Datentypen aber gewissen, durch ihre geometrischen Eigenschaften vorgegebenen, Einschränkungen. So lassen sich mit ihnen keine Kurven und dementsprechend auch keine Kreise und Flächen sinnvoll darstellen. Zu diesem Zweck stellt Oracle die sogenannten *Geometrie* Datentypen (Abbildung 2.10) zur Verfügung.

[8] Aktuelle Version der Oracle Datenbank

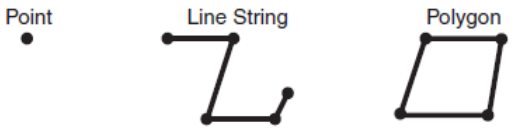

Abbildung 2.9.: Primitive Datentypen in ORACLE™ Spatial

1. Arc-Line-String
2. Arc-Polygon
3. Compound-Polygon
4. Compound-Line-String
5. Circle
6. Rectangle

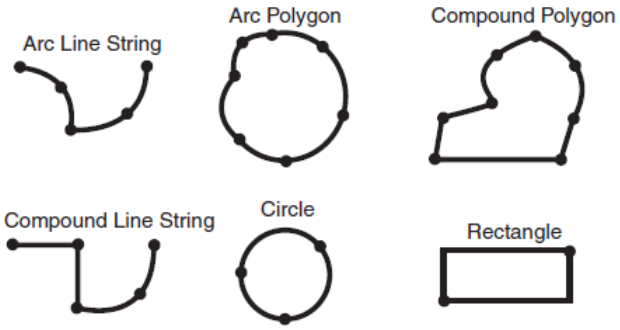

Abbildung 2.10.: Weiterführende Datentypen in ORACLE™ Spatial

Um jetzt ein Bundesland, wie zum Beispiel Niedersachsen, korrekt darzustellen, benötigen wir die in Abbildung 2.9 und Abbildung 2.10 gezeigten Datentypen. Niedersachsen besteht aus einem großen Polygon (Festlandfläche), in der Bremen ausgeschnitten wird, sowie die Polygone für die Inseln. Mehrere Geometrien, die fachlich zusammengehören, bezeichnen wir als Schicht (Layer). Die Geometrien der Bundesländer bilden also eine Schicht (Abbildung 2.11).
Auch ist es möglich, Geometrien für den $\mathbb{R}^3$, beziehungsweise $\mathbb{R}^4$, Raum zu bilden. Da sich diese Arbeit aber nur mit der horizontalen Ebene beschäftigt, wird auf 3D oder auch 4D (lineares Referenzieren) Geometrien verzichtet.

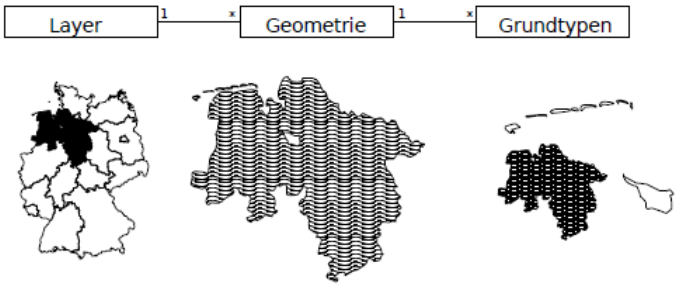

Abbildung 2.11.: Jede Schicht (Bundesländer) kann aus beliebig vielen Geometrien (Niedersachsen) bestehen, wobei die Geometrien wiederum aus vielen primitiven Datentypen 2.9 zusammengesetzt sind.

2.2.2. Koordinatensystem

Ein Koordinatensystem [9] verbindet angegebene Koordinaten mit ihrem Standort und stellt die Beziehung zwischen mehreren Geometrien her. Außerdem erlaubt das Koordinatensystem eine Repräsentation seiner Position im realen Raum.
Jeder *Spatial* hat ein zu sich selbst verbundenes Koordinatensystem. Dieses kann entweder georeferenziert (also verbunden zur Repräsentation auf der Erde) oder nicht georeferenziert (z.B. das kartesische Koordinatensystem, welches keine direkte Verbindung zur Repräsentation auf der Erde hat) sein. Der Vorteil an georeferenzierten Koordinatensystemen ist, dass jede Angabe eine Standard-Einheit hat, in der sie gemessen wird, wie z.B. Meter oder Meilen.
Spatial Daten können mit dem kartesischen, geodetischen (geografisch), abgebildeten oder lokalen Koordinatensystem verbunden sein.

1. **Kartesische Koordinaten** sind Koordinaten, die, gemessen von einem lokalen Ursprung, einen bestimmten Abstand zu beliebig vielen Achsen haben.

2. **Geodetische Koordinaten** sind Koordinaten, die, angegeben in Winkelgraden, die Position relative zu den Pol-Koordinaten der Erde angeben.

3. **Abgebildete Koordinaten** sind Koordinaten, die, basierend auf dem kartesischen Koordinatensystem, nach einem mathematischem *mapping* auf genau einen Punkt auf der Erdoberfläche zeigen.

4. **Lokale Koordinaten** sind kartesische Koordinaten in einem nicht zur Erde referenzierendem Koordinatensystem. Diese werden oft in CAD-Applikationen benutzt.

[9] Im Allgemeinen als *spatial reference system* bezeichnet

Attribute	Erklärung
SDO_GTYPE	Art der gepeicherten Geometrie
SDO_SRID	Koordinatensystem
SDO_POINT	Einzelne Koordinate
SDO_ELEM_INFO	Metadaten für $SDO\_ORDINATES$
$SDO\_ORDINATES$	Liste von Koordinatenwerte

Tabelle 2.2.: Attribute des Datentyps $SDO\_GEOMETRY$

Intern verwendet ORACLE™ Spatial bei Operationen auf Geometrien immer ein Computermodell, welches auf dem kartesischem Koordinatensystem basiert.

2.2.3. Der Datentyp *SDO_GEOMETRY*

Geo-Daten werden in ORACLE™ Spatial mit dem Datentyp $SDO\_GEOMETRY$ repräsentiert (siehe Tabelle 2.2).

SDO_GTYPE

Das SDO_GTYPE zeigt an, um was für einen Typ von geometrischer Form es sich bei dem Objekt handelt.
Der SDO_GTYPE ist ein durch durch Ziffern kodierter Wert in dem Format *dltt*, wobei

- *d* für den Wert der Dimension steht (2, 3 oder 4)
- *l* die Messdimension für die lineare Referenzierung in einem 3D *linear referencing system* ist (Wird hier nicht betrachtet, in dieser Arbeit immer 0)
- *tt* 2.3 die geometrische Form identifiziert (00 bis 07 und 08 bis 99 für die zukünftige Formen)

SDO_SRID

Der Wert SDO_SRID gibt an, welches Koordinatensystem 2.2.2 (*spatial reference system*) mit dem Objekt verbunden ist. Wenn dieser Wert $null$ ist, so ist kein Koordinatensystem mit dem Objekt verbunden. Ansonsten sind Werte erlaubt, die in der Tabelle $SDO\_COORD\_REF\_SYS$ enthalten sind (Murray, 2011, S. 133).

Wert	Geometrie
00	$Unknown$
01	$Point$
02	$Line$ oder $Curve$
03	$Polygon$ oder $Surface$
04	$Collection$
05	$Multipoint$
06	$Multiline$ oder $Multicurve$
07	$Multipolygon$

Tabelle 2.3.: Werte für *tt* des SDO_GTYPE

SDO_POINT

Wenn das Objekt eine Geometrie vom Typ $Point$ hält, also SDO_GTYPE : *dl*01, wird der Punkt direkt als dieses Attribut im Format SDO_POINT_TYPE gespeichert. Für komplexere Geometrien wird dieses Attribut auf *null* gesetzt.

SDO_ELEM_INFO

SDO_ELEM_INFO beschreibt die Interpretation der Koordinaten in $SDO\_ORDINATES$. Wobei jedes Triple

$$[SDO\_STARTING\_OFFSET\ ,\ SDO\_ETYPE\ ,\ SDO\_INTERPRETATION]$$

in der Zahlenfolge ein Element beschreibt, während:

- SDO_ETYPE den Elementtyp festlegt
- $SDO\_INTERPRETATION$ die Auswertung der Koordinaten angibt
- SDO_OFFSET die absolute Position in der Zahlenfolge, ab der das Element beginnt, deklariert

Die Tabelle 2.4 gibt einige mögliche Kombinationen der Zahlenfolgen an. Alle Kombinationen sind in dem *Oracle User's Guide* (Murray, 2011, S. 53) zu finden.

SDO_ORDINATES

Die Koordinaten stehen als Liste von Zahlen in $SDO\_ORDINATES$. Dabei bilden jeweils d Zahlen eine Koordinate. Die Dimension d wurde in SDO_GTYPE festgelegt. Die Reihenfolge der Koordinaten ist entscheidend. Zusätzlich muss darauf geachtet werden, dass bei

SDO_ETYPE	$SDO\_INTERPRETATION$	Beduetung
0	*	nicht unterstütze Geometrie in der Datenbank
1	1	Punkt (Ortsvektor)
1	0	Punkt (Richtungsvektor zum vorherigen)
1	> 1	Mehrere Punkte
2	0	Linienzug
2	1	Bogenlinie
2	0	Linienzug
1003/2003	1	Einfaches Polygon
1003/2003	3	Rechteck

Tabelle 2.4.: Kombinationen von SDO_ETYPE und $SDO\_INTERPRETATION$ mit deren Bedeutung

äußeren Polygonen die Koordinaten gegen den Uhrzeigersinn angegeben werden, bei inneren Polygonen im Uhrzeigersinn.

Beispiel

Das angegebene Beispiel stammt aus dem *Oracle User's Guide* (Murray, 2011, S. 47-51). Anzunehmen ist, dass ein Unternehmen seine Märkte in einer Datenbank abspeichert und die dazugehörige Position eines Marktes als Rechteck mit den geodetischen Koordinaten kodiert.

1. Tabelle erstellen

An Anweisung 2.3 ist zu sehen, dass unser $SDO\_GEOMETRY$ -Objekt in der Spalte *shape* gehalten wird.

```
1 CREATE TABLE market (
2         mkt_id NUMBER PRIMARY KEY,
3         name VARCHAR2(32),
4         shape SDO_GEOMETRY
5 );
```

Listing 2.3: SQL Anweisung zum Erstellen der Markttabelle

2. Mit Daten füllen

Die Spalten mkt und $name$ sind Standard-SQL Datentypen und daher nicht weiter zu erklären, interessant ist das Füllen der Spalte $shape$. Beim $INSERT$ nutzen wir die in diesem Kapitel beschriebenen validen Werte für $SDO\_GEOMETRY$.

```
1 INSERT INTO market VALUES(
```

```
2           1,
3           'markt_a',
4           SDO_GEOMETRY(
5                   2003,       SDO_GTYPE
6                   NULL,       SDO_SRID
7                   NULL,       SDO_POINT
8                   SDO_ELEM_INFO_ARRAY(1,1003,3),      SDO_ELEM_INFO
9                   SDO_ORDINATE_ARRAY(1,1,  5,7)       SDO_ORDINATES
10          )
11  );
```

Listing 2.4: Einfügen eines Markts in die Beispieltabelle

Die SQL-Anweisung 2.4 fügt einen Markt mit der *mkt_id* '1' und dem *name* 'markt_a' ein. Das $SDO\_GEOMETRY$ -Objekt enthält ein zweidimensionales Polygon (SDO_SRID = 2003). Da es sich nicht um ein bestimmtes Koordinatensystem handelt, wird für den Wert SDO_SRID , wie auch für SDO_POINT (kein einfacher Punkt), *null* eingefügt. Komplizierter wird es beim Wert für das Attribut SDO_ELEM_INFO, dort wird ein Objekt vom Typ $SDO\_ELEM\_INFO\_ARRAY$ eingefügt, welches eigentlich nur sicherstellt, dass valide Werte für dieses Attribut verwendet werden (Beispielkombinationen in 2.2.3). In diesem Fall hat das Element einen Offset von 1 und wird durch die Werte 1003 und 3 als einfaches Rechteck erkannt. Die eigentliche Position im vorher beschriebenen Koordinatensystem ist hier als zweidimensionale Matrix zu sehen: $\left(\begin{smallmatrix}1 & 5\\ 1 & 7\end{smallmatrix}\right)$.
Es entsteht also die in Abbildung 2.12 gezeigte geometrische Form.

2.2.4. Geometrie-Tabellen

Im Gegensatz zu gewöhnlichen SQL-Tabellen erfordern Geometrie-Tabellen mehrere Schritte. Grundsätzlich ist folgende Vorgehensweise einhalten:

1. Tabelle anlegen (SQL Anweisung 2.3)
2. Metadaten definieren
3. Index anlegen

Das Anlegen einer Tabelle, die eine geometrische Form hält, wurde bereits im Abschnitt 2.2.3 abgehandelt. Die Metadaten werden jeweils für eine Schicht in das System eingepflegt. Das ist nötig, um den Bereich außerhalb einer Form zu bestimmen, dies wird unter anderem dazu genutzt, um topologische Beziehungen zwischen mindestens 7 Objekten herzustellen. Auch hier wird auf das Beispiel 2.2.3 zurückgegriffen.

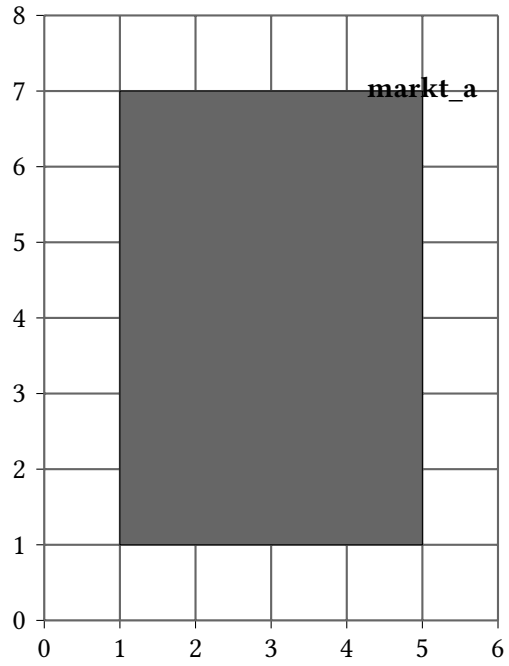

Abbildung 2.12.: Entstehendes Rechteck im angegebenen Koordinatensystem

```
INSERT INTO user_sdo_geom_metadata
        (
                TABLE_NAME,
                COLUMN_NAME,
                DIMINFO,
                SRID
        ) VALUES (
                'market',
                'shape',
                SDO_DIM_ARRAY(
                        SDO_DIM_ELEMENT('X', 0, 20, 0.005),
                        SDO_DIM_ELEMENT('Y', 0, 20, 0.005)
        ),
        NULL
);
```

Listing 2.5: Einfügen der Metadaten für die *market*-Tabelle aus Beispiel 2.2.3

Die Anweisung 2.5 fügt also für die Tabelle *market* mit der Geometriespalte *shape* Metadaten hinzu. Das SDO_DIM_ARRAY legt die Grenzen des Koordinatensystems fest. Da im Beispiel 2.2.3 kein konkretes Koordinatensystem festgelegt worden ist, wird auch hier kein fest

Bezeichnung	Definition
$SDO\_DISTANCE$	Berechnet die Distanz zwischen 2 $SDO\_GEOMETRY$ Objekten
SDO_LENGTH	Berechnet die Länge eines Linienzugs
SDO_AREA	Berechnet den Flächeninhalt einer Geometrie.
SDO_MBR	Berechnet das minimal umgebende Rechteck (MUR) einer Geometrie
SDO_BUFFER	Berechnet die Pufferzone um eine Geometrie
$SDO\_CONVEXHULL$	Erlaubt die Bestimmung der Konvexen Hülle einer Geometrie

Tabelle 2.5.: Aggregatfunktionen von ORACLE™ Spatial

definiertes benötigt.
Anschließend wird ein Index für die Schicht definiert. Dies ist nicht zwingend notwendig. Allerdings ist es empfehlenswert, einen Index zu erstellen, da dieser unter anderem zu einer höheren Performanz bei der Kandidatensuche führt.

```
1 CREATE INDEX
2         market_spatial_idx
3 ON
4         market(shape)
5 INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

Listing 2.6: Einführen eines Index für die Schicht *market.shape*

2.2.5. Aggregatfunktionen

Aus dem Standard-SQL sind bereits einige Aggregatfunktionen wie $SUM(x)$ oder $COUNT(x)$ bekannt. Auch Oracle™ Spatial bietet zur Vereinfachung einige schon fest implementierte Aggregatfunktionen. Tabelle 2.5 zeigt einige der von Oracle bereitgestellten Funktionen. In (Murray, 2011, S. 255) finden sich die restlichen Funktionen.

SDO_DISTANCE

Als mächtigste Funktion gilt $SDO\_DISTANCE$. Da diese in dieser Arbeit am häufigsten verwendet wird, dient sie als einführendes Beispiel für Aggregatfunktionen in Oracle™ Spatial. SQL-Anweisung 2.7 zeigt die Syntax der Funktion, welche nach korrekter Ausführung die Distanz zwischen 2 $SDO\_GEOMETRY$ Objekten ausgibt. Abbildung 2.13 deutet an, dass diese Funktion immer den kürzesten Weg zwischen 2 Objekten wählt, das heißt für Polygone wird immer der Schnittpunkt der Geraden zwischen zwei Formen ausgewählt, der am Kürzesten ist.

```
1 SDO_GEOM.SDO_DISTANCE (
```

```
    geom1       IN  SDO_GEOMETRY,     1. Geometrie
    geom2       IN  SDO_GEOMETRY,     2. Geometrie
    tolerance   IN  NUMBER            Toleranz
[ , unit VARCHAR2 ]                   optional: Laengeneinheit
) RETURN NUMBER;                      berechneter Abstand
```

Listing 2.7: Syntax der Aggregatfunktion $SDO\_DISTANCE$

Abbildung 2.13.: Ergebnis der $SDO\_DISTANCE$ Funktion bei zwei Polygonen

3. Datengewinnung

Das Gewinnen der Daten für die Applikation beginnt mit dem Ausmessen und Einteilen in Bereiche des Golfplatz. Jedes Loch des hier für die prototypische Applikation benutzten Golfplatzes in Adendorf [1] wird per Google Earth™[2] in Abschlagbox (Abbildung 3.1 weiß), Fairway (Abbildung 3.1 grün), Bunker (Abbildung 3.1 rot) und Grün (Abbildung 3.1 blau) eingeteilt. Für

Abbildung 3.1.: Vermessenes Loch #2 im Golfclub Adendorf

die Vermessung wird das Polygon-Tool von Google Earth™ benutzt. Damit ist es möglich, Polygone über Areale mit seinen geodetischen Informationen zu legen. Außerdem liefert *Google Earth* die Funktionalität des Exportierens von erstellten Polygonen. Damit kann man jegliche Informationen, wie zum Beispiel die zusammenhängenden GPS Koordinaten, die das Polygon beschreiben, erhalten. Diese Informationen werden in der Beschreibungssprache *Keyhole Markup Language* (im Folgenden KML genannt) dargestellt. KML ist eine enge Verwandte der Beschreibungssprache XML (*Extended Markup Language*) und wurde von Google extra zu dem Zweck entwickelt, geodetische Informationen zu halten (Google-Maps-Group, 2014).

[1] http://www.golf-adendorf.de
[2] http://www.google.de/intl/de/

KML wurde im Jahr 2010 als offener Standard von dem Open Geospatial Consortium (OGC)[3] anerkannt. Damit wurden die Ideen von KML eng an das OGC Ziel verknüpft, die Darstellung von geodetischen Informationen zu standardisieren (Reed, 2014).

```
<?xml version="1.0" encoding="UTF 8"?>
<kml
        xmlns="http://www.opengis.net/kml/2.2"
        xmlns:gx="http://www.google.com/kml/ext/2.2"
        xmlns:kml="http://www.opengis.net/kml/2.2"
        xmlns:atom="http://www.w3.org/2005/Atom"
>
<Document>
        <name>Green #2.kml</name>
        <Placemark>
        <name>Green #2</name>
        <description>Green Hole 2</description>
        <styleUrl>#msn_ylw pushpin</styleUrl>
        <Polygon>
                <tessellate>1</tessellate>
                <outerBoundaryIs>
                <LinearRing>
                        <coordinates>
                                10.45920570828532,53.29481621044126,0
                                10.45920248711797,53.294808666673595,0
                                10.45919608748492,53.2948010897883,0
                                10.4591896896328,53.29479351494379,0
                                10.45918647115673,53.29478597598578,0
                                10.45918006600492,53.29477652916211,0
                                10.45917684907856,53.29476899350642,0
                                10.45917363271342,53.29476145899487,0
                                10.45917359267849,53.29475395804754,0
                                10.4591703670302,53.29474455112066,0
                                10.45917032709133,53.29473705172729,0
                                10.45917028719484,53.29472955305715,0
                                10.45917342249997,53.29472208533969,0
                                10.45917655787464,53.29471461775481,0
                                10.45917652776908,53.29470899448663,0
                                10.4591796631132,53.29470152669926,0
                                10.45917961273788,53.29469215546841,0
                        </coordinates>
                </LinearRing>
                </outerBoundaryIs>
```

[3] http://www.opengeospatial.org

```
39              </Polygon>
40              </Placemark>
41  </Document>
42  </kml>
```

Listing 3.1: Beispiel einer von Google Earth™ erstellten KML-Datei. Diese Datei zeigt das Grün von Loch #2 (Abbildung 3.1 blau) in der verkürzten Fassung. Die Originaldatei hält 98 statt 16 Koordinaten

Die KML-Datei 3.1 enthält die wesentlichen Informationen für das Loch 2 des Golfplatzes in Adendorf. Zum einen die XML Metainformationen, die allerdings für die Verarbeitung nicht relevant sind, sowie zum anderen die Informationen, die von dem Benutzer in Google Earth™ eingetragen worden sind. Die für die Datenbank wichtigen Informationen werden ab Zeile 19 aufgelistet und zwar als Tupel $T = \langle Längengrad, Breitengrad, Elevation \rangle$. Für die Zwecke dieser Arbeit sind aber nur die ersten beiden Elemente des Tupels T relevant, denn die Elevation wird automatisch von dem in ORACLE™ Spatial zu Grunde liegenden Erdmodell zu einer Koordinate angefügt. So können also auch Höhenunterschiede bei Golfschlägen mit einbezogen werden. Dies ist wichtig, da sich diese Unterschiede auf die Schlägerwahl auswirken können.
Insgesamt werden für diese Arbeit 20 dieser, wie in 3.1 beispielhaft angedeuteten KML Dateien für Abschläge, Fairways, Bunker und Grüns erzeugt. Das eigentliche Auslesen (*parsen*) und Eintragen der Daten in die Datenbank wird ausführlich in Kapitel 4 behandelt, da hier auch das Datenbankdesign eine tragende Rolle spielt.

4. Entwurf

4.1. Datenbankdesign

Die Struktur einer relationalen Datenbank wird meistens durch ein ER-Diagramm modelliert. Wenn performancekritische Zugriffe vorauszusehen sind, dann kann noch ein Schritt der Denormalisierung (Totok, 2012, S. 112) folgen. Viele Datenmodellierungs-Tools bieten eine Unterstützung an, sodass aus dem Datenmodell die SQL-DDL-Statements zum Aufbau der Datenbank generiert werden können. Durch das Ausführen der SQL-DDL-Statements werden die Datenbank-Objekte erstellt:

- Schema
- Tablespace
- Tabelle
- Integritätsbedingung
- Index
- View

Nachdem die Datenbank-Objekte erstellt sind, können die Datensätze in die Datenbank eingefügt werden.
Um diese allgemeinen Bedingungen auf das in dieser Arbeit behandelte Szenario anzuwenden, muss geklärt werden, wie die Begriffe nun in unserer geschlossenen Welt übersetzt werden.

4.1.1. Schema

Das Datenbankschema stellt eine formale Beschreibung der Datenbank dar, also in welcher Beziehung Daten, beziehungsweise Objekte, zueinander stehen. Am einfachsten ist das Schema einer Datenbank mithilfe der *ANSI-SPARC-Architektur* (Sauer, 2002, S. 23) zu erläutern. In dieser Herangehensweise gibt es 3 Ebenen, wobei nur die Ebene 2 hier von Relevanz ist. Die

externe Ebene wird dann im Kapitel 4.2 aufgegriffen. Die 3. Ebene wird von dem in dieser Arbeit verwendeten Oracle DBMS verwaltet und optimiert.

1. Die externe Ebene, die den Benutzern und Anwendungen individuelle Benutzersichten bereitstellt. Beispiele: Formulare, Masken-Layouts, Listen, Schnittstellen.

2. Die konzeptionelle Ebene, in der beschrieben wird, welche Daten in der Datenbank gespeichert sind, sowie deren Beziehungen zueinander. Designziel ist hier eine vollständige und redundanzfreie Darstellung aller zu speichernden Informationen. Hier findet die Normalisierung des relationalen Datenbankschemas statt.

3. Die interne Ebene (auch physische Ebene), die die physische Sicht der Datenbank im Computer darstellt. In ihr wird beschrieben, wie und wo die Daten in der Datenbank gespeichert werden. Designziel ist hier ein effizienter Zugriff auf die gespeicherten Informationen. Das wird meistens nur durch eine bewusst in Kauf genommene Redundanz erreicht (z.B. im Index werden die gleichen Daten gespeichert, die auch schon in der Tabelle gespeichert sind).

Konzeptionelle Ebene

Welche Daten in der Datenbank gespeichert werden und welche Beziehung diese Daten zueinander haben, leitet sich aus Kapitel 1.2 ab.
Um einen Golfschlag vorschlagen zu können müssen wesentliche Daten bekannt sein. Dazu zählen vor allem Informationen über den schlagenden Spieler ($GOLFER$), wie das Handicap, und über das bespielte Loch ($HOLE$). Da einem Golfspieler maximal 14 Golfschläger in der Golftasche zur Verfügung stehen, wird auch die Entität des Schlägers ($CLUB$) eingeführt. Durch die Differenzierung des Schlägers kann das Programm im späteren Verlauf mit Hilfe von OLAP genau ermitteln, mit welchem Schläger der Spieler wie weit und vor allem genau schlagen kann. Für die Funktionalität des Vorschlagens eines Schlags, beziehungsweise eines Schlägers, müssen dem Programm weiterführende Informationen zu dem Loch bekannt sein, zusätzich zu der Länge und den Positionen des Fairways und des Grüns. Wichtiger sind vielmehr die genauen Positionen von Hindernissen, wie Bunker oder Wasserhindernisse ($HAZARD$). In der Übersicht ergeben sich also folgende Entitäten:

- Golfer
- Golfstroke
- Hole

- Club
- Hazard
- Golfcourse

Die Beziehungen dieser Entitäten zueinander werden am besten in einem *Entity-Relationship-Modell* (Pernul und Unland, 2003, S. 69) dargestellt. Das zugehörige ERM 4.1 zeigt noch nicht die eigentliche Implementierung der Problemstellung innerhalb des Tabellenschemas, sondern lediglich die Beziehungen zwischen den aus Kapitel 1.2 ausgelesenen Entitäten. Zusätzlich

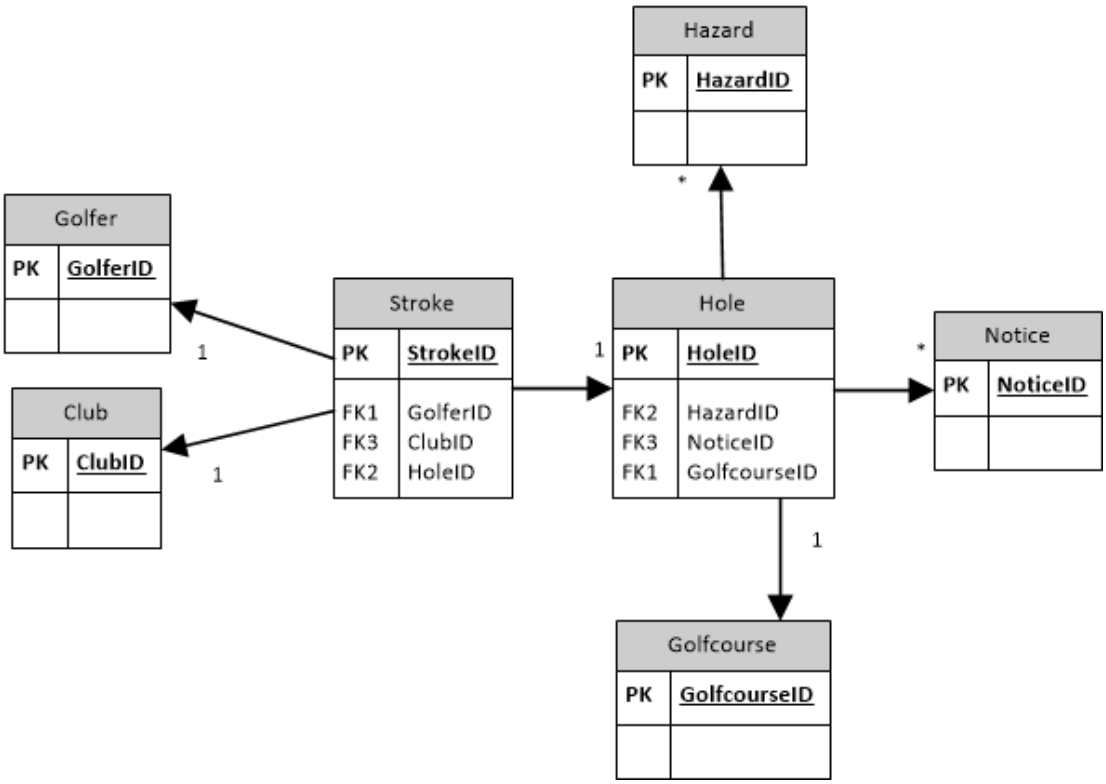

Abbildung 4.1.: Entity-Relationship-Modell mit den Beziehungen der zu erstellenden Datenbank

zu den Beziehungen der Entitäten untereinander beschreibt ein weiterführendes ERM, welche Daten innerhalb einer Tabelle konsistent gehalten werden. Hier muss weit vorausgedacht werden, da sich im späteren Verlauf der Entwicklung alle Änderungen, die dann noch durchgeführt werden müssen, stark auf die Qualität des Designs der Datenbank auswirken können. Das ERM A.4 zeigt außerdem noch, dass einige Spalten einer Tabelle in eigene Typen ausgelagert worden sind. Dieses Ausgliedern macht es möglich, das Design trotz späterer Änderungen

relativ konsistent zu halten. Außerdem wird so eine Typsicherheit geschaffen. Formal dargestellt ergibt sich also folgende Definition der Datenbank:

$$Golfer = \{\underline{GolferID}, forename, surname, handicap\} \tag{4.1a}$$
$$Club = \{\underline{GolferID}, club\_type\}\} \tag{4.1b}$$
$$Hazard = \{\underline{HazardID}, hazard\_type\} \cup \{Hole(HazardID)\} \tag{4.1c}$$
$$Golfcourse = \{\underline{GolfcourseID}, label, address\} \tag{4.1d}$$
$$Notice = \{\underline{NoticeID}, text\} \tag{4.1e}$$
$$\begin{aligned} Hole = {} & \{\underline{HoleID}, length, fairway, tee, green\} \cup \{Golfcourse(GolfcourseID)\} \\ & \cup \{Hazard(HazardID)\} \cup \{Notice(NoticeID)\} \end{aligned} \tag{4.1f}$$
$$\begin{aligned} Stroke = {} & \{\underline{StrokeID}, length, from, to, date, quality\} \\ & \cup \{Golfer(GolferID)\} \cup \{Club(ClubID) \cup \{Hole(HoleID)\}\} \end{aligned} \tag{4.1g}$$

Aufgrund dieser Relationen ist schon zu erkennen, dass sich viele $JOIN$ Anweisungen nicht vermeiden lassen. Daher ist es von Nöten, die Datenbank zu *refactorn*. Dazu bietet Oracle 11g die Möglichkeit, ein sogenanntes hybrides System (Cock, 1997) aufzubauen, das heißt also, dass die Vorteile von relationalen Datenbanken mit denen aus der objekt-relationalen Welt gemischt werden. Dadurch wird erreicht, dass der Benutzer die $JOIN$ Anweisungen nicht mehr von Hand ausführen muss und dass stattdessen das DBMS die Verbindungen zwischen Tabelle-Objekt oder auch Objekt-Objekt über die Objekt-ID eigenständig zuordnet.
In Abbildung A.5 ist zu erkennen, dass aus dem ERM mit 6 Fremdschlüsselbeziehungen ein ERM mit nur einer Fremdschlüsselbeziehung geworden ist. Für die 1-zu-1 Beziehungen wurde ein einfacher Typ als Objekt eingeführt und für die 1-zu-n Beziehungen eine Typliste.
Um die Datenbank für die OLAP Analyse nicht unnötig komplex zu machen, da an dieser Stelle die Spalten in atomaren Datentypen vorliegen müssen, bildet Abbildung 4.2 die endgültige Fassung des Designs der Datenbank ab.

4.1.2. Tablespace

Ein *Tablespace* bezeichnet im Datenbankbereich den Speicherort, in den Tabellen, Indizes und andere Datenobjekte geschrieben werden. Die Aufgabe, den optimalen Speicherplatz für die Daten zu finden, übernimmt das hier eingesetzte Oracle DBMS.

4.1.3. Tabellen

Das eigentliche Erstellen der Tabellen und Typen basiert auf den in Abbildung A.5 angegebenen Entitäten und den dazugehörigen Datentypen der Spalten. Eine Auflistung der SQL

Anweisungen zum Erstellen jeder einzelnen Tabelle und Fremdschlüsselbeziehung erübrigt sich durch die detaillierte Darstellung aus Abbildung A.8.

Indizes

In diesem Abschnitt werden nicht die einfachen Fremdschlüsselbeziehungen der Entitäten zueinander erläutert, sondern vielmehr die interne Indizierung der geometrischen Formen innerhalb der Datenbank.
Als Grundlage für die Indizierung gelten die Metadaten. Diese müssen per Konvention von Oracle in der Tabelle $USER\_SDO\_GEOM\_METADATA$ in jedem Tablespace abgelegt werden, der das Paket $SDO\_GEOMETRY$ verwendet. Die Anweisung 4.1 zeigt die Erstellung dieser Metadatentabelle. In der Anweisung 4.1 sind besonders die Spalten $DIMINFO$ und $SRID$ zu erläutern. Einfach verhält es sich bei der Spalte $SRID$ oder *Spatial Reference System Identifier* — diese gibt das Koordinatensystem an, welches als Grundlage für die Geometrien verwendet werden soll. Da es sich in dieser Arbeit nur um Geometrien aus der realen Welt (also angegeben in Längen- und Breitengrad) handelt, wird diese Spalte immer auf den Wert 4326 gesetzt. Dieser Wert repräsentiert das *World Geodetic System* (aktuell in Version WGS84 (Imagery und Defense, 2000)), also welches mit dem jeder westliche Kartenanbieter arbeitet.
Komplizierter wird es bei der Spalte $DIMINFO$. Der Wert dieses Eintrags hängt direkt mit der Wahl des $SRID$ ab. An dieser Stelle werden weiterführende Informationen abgelegt, die dann dazu führen, dass Aggregatfunktionen durch eine optimierte Indizierung schneller ein Ergebnis liefern. Da das SDO_DIM_ARRAY dieser Spalte die Minimal- und Maximalwerte jeder Achse in diesem Koordinatensystem angibt, muss zum Beispiel die Aggregatfunktion $SDO\_DISTANCE$ (beschrieben in Abschnitt 2.2.5) nicht jedes mögliche Koordinatensystem und deren Grenzen durchlaufen, um ein passendes Ergebnis zu errechnen.

```
1 CREATE TABLE _USER_SDO_GEOM_METADATA
2   (
3     TABLE_NAME   VARCHAR2(30),
4     COLUMN_NAME VARCHAR2(30),
5     DIMINFO  MDSYS.SDO_DIM_ARRAY,
6     SRID  NUMBER
7   );
```

Listing 4.1: SQL Anweisung zum Erstellung des Speichers der Spatial-Metadaten

Nach dem Definieren des Speicherplatzes für die Metadaten müssen alle in Spalten vorkommenden Geometrien in dieser Tabelle hinterlegt werden. So muss zum Beispiel HOLE(FAIRWAY),

in dem Metadaten abgelegt werden (wie es Anweisung 4.2 zeigt), um diese Spalte im Folgenden indizieren zu können.

```
INSERT INTO USER_SDO_GEOM_METADATA
  (
    TABLE_NAME,
    COLUMN_NAME,
    DIMINFO,
    SRID
  )
  VALUES
  (
    'HOLE',
    'FAIRWAY',
    SDO_DIM_ARRAY (
        SDO_DIM_ELEMENT('LONG', 180.0, 180.0, 0.5),
        SDO_DIM_ELEMENT('LAT', 90.0, 90.0, 0.5)
    ),
    4326
  );
```

Listing 4.2: SQL Anweisung zum namensraumweiten Definieren einer Geometrie

Nach dem Eintragen der Metadaten kann dem System jetzt der Index bekannt gemacht werden. Dies wird im weiteren Verlauf nicht nur zur Optimierung der Geschwindigkeiten benutzt, sondern auch dazu, aufbauenden Programmen, wie dem Mapviewer (Abschnitt 5.1.3) die zu Verfügung stehenden Geometrien bekannt zu machen, um diese später auf einer Karte darzustellen. Anweisung 4.3 zeigt die DDL[1] für den Index der Spalte $FAIRWAY$ der Entität $HOLE$.

```
CREATE INDEX "FAIRWAY_SPATIAL_INDEX" ON "HOLE" ("FAIRWAY")
   INDEXTYPE IS "MDSYS"."SPATIAL_INDEX" PARAMETERS ('layer_gtype=POLYGON');
```

Listing 4.3: DDL zum Erstellen des Fairway-Index

4.2. OLAP Entwurf

Um jetzt einen in sich konsistenten und fachlich logischen Entwurf für eine OLAP-Anwendung zu erstellen, werden die Grundlagen aus Abschnitt 2.1 genutzt, um die Punkte Dimensionen 2.1.2 bis hin zu den Faktentabellen 2.1.6 auf das Szenario aus Abschnitt 1.2 anwenden.

[1] Data Definition Language

4.2.1. Dimensionen und Cubes

Um jetzt die Kenndaten aus dem in dieser Arbeit bearbeitenden Szenario zu ziehen, muss erkannt werden, welche Daten hier die wichtigsten zu messenden Kennziffern sind. Im Falle eines Golfschlags ist es zum einen die Länge und zum anderen die Genauigkeit. Die Länge eines Golfschlags wird durch das Messen der Distanz zwischen `Stroke(`$From$`)` nach `Stroke (`to`)` berechnet. Es wäre sich weiterhin vorzustellen, dass die Zeit ($TIME$) genau so von Belangen ist, wie auf welchem Loch ($HOLE$) und mit welchem Schläger ($CLUB$) dieser Schlag ausgeführt worden ist. Abbildung 4.3 zeigt eine grafische Darstellung der Dimensionen, um ein visuelles Verständnis von den definierten Dimensionen zu erhalten. So ergibt die Auswertung der Koordinate k im dreidimensionalen Raum, der durch $hole$, $club$ und $time$ aufgespannt wird, die Länge des Schlags.

$$length = (time, club, hole)$$

Ein anderes Merkmal, welches für die Entscheidung des richtigen Schlägers von Belangen ist, ist die Qualität des Ergebnisses des Schlags. Da dies ein rein subjektiver Wert ist und dieser direkt vom schlagenden Spieler eingetragen wird, kommt es bei der Qualität darauf an, aus welcher Distanz , mit welchem Schläger, auf welchem Loch und zu welcher Zeit der Schlag ausgeführt worden ist. Da ein vier Dimensionen Würfel nicht mehr visuell erfassbar ist, wird dieser rein durch

$$quality = (time, club, hole, distance)$$

beschrieben. An dem Beispiel dieses Datenwürfels ist ein Nachteil der OLAP Architektur zu erkennen. So ist es rein technisch möglich, einen $n - dimensionalen$ Cube zu realisieren. Dennoch bleibt das menschliche Verständnis auf der Strecke, da es nicht möglich ist, wie in Abschnitt 2.1.5 erläutert, mehr als 3 Dimensionen visuell zu erfassen. Dennoch führen die zwei vorangegangen $cubes$ nicht zum dem Ziel, dem Benutzer der Applikation einen adäquaten Schläger für die aktuelle Distanz zum Grün vorzuschlagen. An dieser Stelle kommt ein Nachteil von OLAP zum Tragen: es ist für die reine numerische Auswertung gemacht und unterstützt, dadurch das Aggregatfunktionen verwendet werden, keine Zeichenfolgen als $measures$. Dennoch ist es jetzt von Nöten, dass man über den die Dimensionen des Golfers, der Distanz und der Zeit einen Schläger erhält. Da dieses Mapping über die Dimensions- und relationalen Tabellen realisiert werden kann , wird dieser Fakt jetzt hinten angestellt werden und dann im Kapitel der Implementierung wieder aufgegriffen. So bietet sich für den Würfel der als Grundlage für das DSS gilt folgende formale Beschreibung:

$$clubpredition = (time, golfer, distance)$$

4.2.2. Measures

Wie in Abschnitt 4.2.1 angedeutet, ist für den Würfel aus Abbildung 4.3 die Distanz zwischen 2 Schlägen, also der Wert, der sich aus der Differenz der geodetischen Werte vom Start- und Endpunkt des Balles ergibt, ein $measure$. In Abschnitt 2.1.3 wurde erwähnt, dass vor dem Darstellen der Daten sogenannte Aggregatfunktionen (Abschnitt 2.2.5) auf die reinen Detaildaten angewendet werden. In diesem Fall wird die mächtigste der Aggregatfunktionen von ORACLE™ Spatial angewendet, um die Distanz zwischen 2 Punkten zu berechnen, $SDO\_DISTANCE$ aus Abschnitt 2.2.5. Dementsprechend ist es möglich, die Länge eines Schlags mit folgender SQL-Anweisung auszurechnen:

```
SDO_GEOM.SDO_DISTANCE(
        this.from,
        this.to,
        0.1,
        'unit=m'
);
```

Listing 4.4: SQL Anweisung zum Berechnen der Länge eins Schlags

Um die Messung direkt anzeigen zu lassen, wird für die Tabelle $STROKE$ ein Trigger geschrieben, der beim Einfügen in die Tabelle die Länge eines Schlags ausrechnet und direkt in der Spalte `Stroke(length)` speichert.

```
CREATE OR REPLACE
TRIGGER calculate_stroke_length
BEFORE INSERT ON stroke
REFERENCING NEW AS NEW
FOR EACH ROW
BEGIN
  SELECT
        sdo_geom.sdo_distance(:NEW.sFrom, :NEW.sTo, 0.1, 'unit=M')
  INTO
        :NEW.LENGTH
  FROM
        dual;
END;
```

Listing 4.5: SQL Anweisung zum Berechnen der Länge eins Schlags

4.2.3. Hierarchien

Um jetzt eine bessere Übersicht für den Anwender zu schaffen, werden die in Abschnitt 4.2.1 definierten Variablen in Ebenen unterteilt.

Dimension $HOLE$

Die Dimension $HOLE$ wird in der ersten Ebene auf die beiden Abschnitte eines Golfplatzes *1st-nine* und *2nd-nine* aufgeteilt. Die Aufteilung der zweiten Ebene stellt dann die atomare Einheit dieser Dimension dar, das Loch an sich. Abbildung 4.4 zeigt die Aufteilung in einer Baumstruktur.

Dimension $CLUB$

Die Dimension $CLUB$ ist in erster Ebene in die Schlägerarten Holz, Eisen und Wedge, sowie in zweiter Ebene in die einzelnen Schläger aufgeteilt (Abbildung 4.5).

Dimension $TIME$

Die Dimension $TIME$ wird in erster Ebene in die Halbjahre, in zweiter Ebene in die Monate und in dritter Ebene in die Tage aufgeteilt (Abbildung 4.6).

Dimension $DISTANCE$

Die Dimension $DISTANCE$ wird für die Auswertung der Qualität eines Golfschlags herangezogen. Sie wird in erster Ebene in 100m und in zweiter Ebene in 10m Schritte aufgeteilt. Da es hier nicht darauf ankommt, dass der Golfball direkt neben dem Loch landet, sondern, dass der Golfer sieht, ob sein Spiel bei einer Entfernung von 120-130m ausbaufähig ist, wird auf die Einteilung in 1 Meter Schritten verzichtet (Abbildung 4.7).

4.2.4. Dimensions- und Faktentabellen

Nach Abschnitt 2.1.6 ist für jede Messung, also jeden Wert, der aus einem $cube$ gelesen werden kann, eine Dimensions- beziehungsweise Faktentabelle zu erstellen. Dementsprechend müssen, nach Abschnitt 4.2.1, Faktentabellen für die $measures$ *quality* , *length* und *clubprediction* entworfen werden.
Welche Tabellen und/oder Views genau erstellt werden müssen, ergibt sich aus der Wahl der Dimensionen der Datenwürfel. Demnach müssen für die in Abschnitt 4.2.1 entworfenen Dimensionen eines Würfels die zugehörigen Dimensionstabellen angelegt werden.

Aus dem Entwurf der Hierarchien (Abschnitt 4.2.3) lassen sich dann die Dimensionstabellen $HOLE\_DIMESIONS$, $CLUB\_DIMENSIONS$, $TIME\_DIMENSIONS$ und $DISTANCE\_DIMENSIONS$ ableiten. Diese Dimensionen werden dann jeweils mit den Ebenen aus deren beschreibenden Baumdiagrammen befüllt.
Des weiteren wird die Dimension $GOLFER\_DIMENSION$ eingeführt, welche als materialisierte View direkt auf die Daten innerhalb der relationalen Tabelle $GOLFER$ verweist. Dies ist nötig, da jeder Schlag genau einem vorher definierten Golfer zugewiesen werden muss um aussagekräftige Statistiken zu liefern.
Aus den Dimensionen setzen sich jetzt über Fremdschlüsselbeziehungen und den gewählten Kennziffern die Faktentabellen zusammen. Aus den in Abschnitt 4.2.2 entwickelten Kennziffern ergeben sich dann die folgenden formalen Abbildungen der Faktentabellen:

$$QUALITY\_FACTS = \{QUALITY, \underline{HOLE\_ID}, \underline{TIME\_ID}, \underline{CLUB\_ID}, \underline{GOLFER\_ID}, \underline{DISTANCE\_ID}\} \tag{4.2a}$$

$$LENGTH\_FACTS = \{\underline{LENGTH}, \underline{HOLE\_ID}, \underline{TIME\_ID}, \underline{CLUB\_ID}, \underline{GOLFER\_ID}\} \tag{4.2b}$$

$$CLUBPREDICTION\_FACTS = \{\underline{CLUB}, \underline{TIME\_ID}, \underline{DISTANCE\_ID}, \underline{GOLFER\_ID}\} \tag{4.2c}$$

Aufgrund des doppelten Vorkommens der Dimensionen in den Faktentabellen wäre es durchaus denkbar diese zusammenfassen. Allerdings wird von dieser Maßnahme Abstand genommen, da das Zusammenfassen eine unnötige Komplexität im späteren Verlauf der Datenauswertung bringen würde.

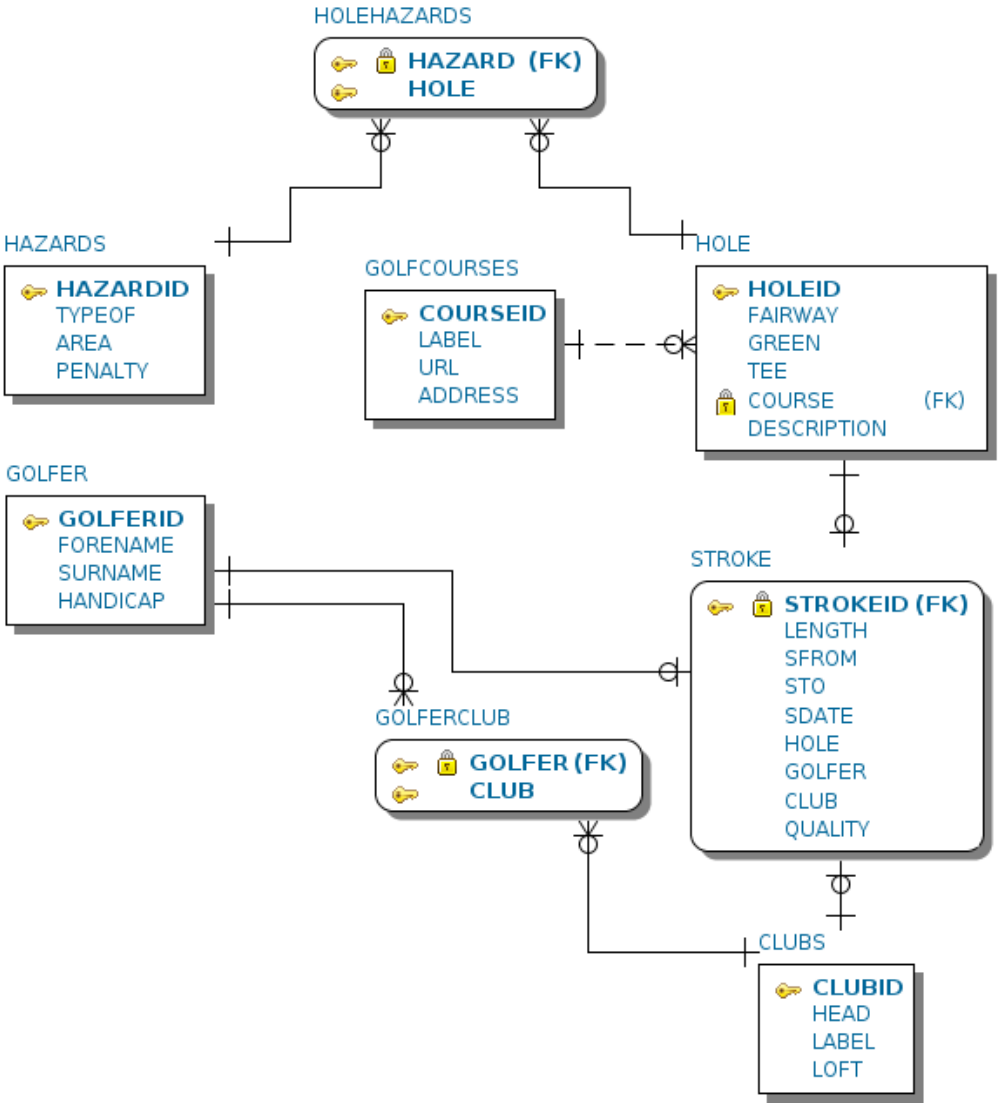

Abbildung 4.2.: Finales ERM für die OLAP Analyse

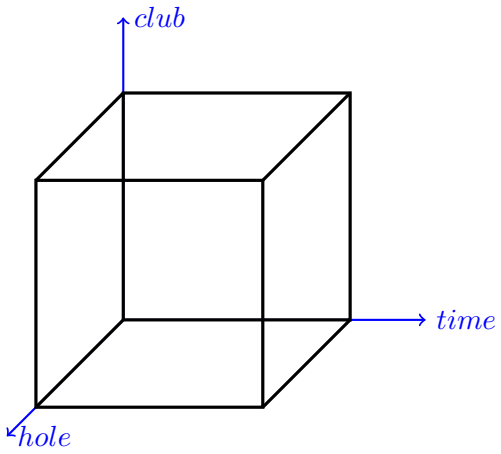

Abbildung 4.3.: *Cube* nach den Dimensionen [time,club,hole], der als Messwert die Distanz eines Schlages enthält

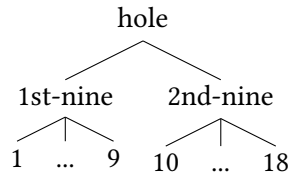

Abbildung 4.4.: Hierarchien der Dimension $HOLE$

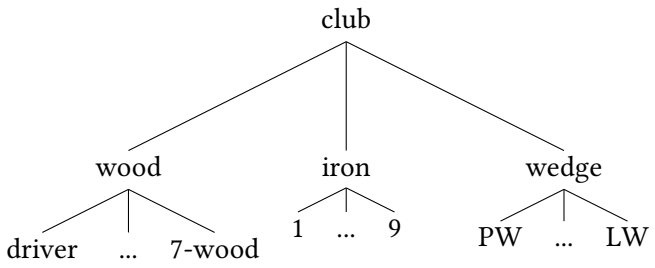

Abbildung 4.5.: Hierarchien der Dimension $CLUB$

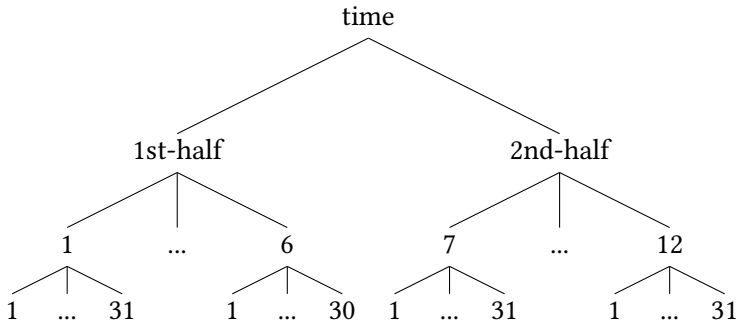

Abbildung 4.6.: Hierarchien der Dimension $TIME$

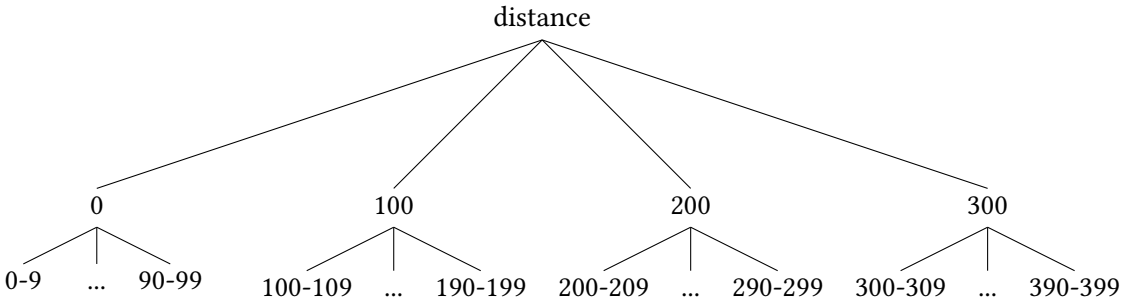

Abbildung 4.7.: Hierarchien der Dimension $DISTANCE$

5. Implementierung

Das Design der Applikation, die das Ziel dieser Arbeit ist, wird sich stark an dem im Kapitel 4.1 beschriebenen Design orientieren. Da es sich bei dem eingesetzten DBMS um ein Produkt von ORACLE™ handelt, werden wegen Kompatibilitätsgründen auch weiterführende ORACLE™ Produkte eingesetzt. Im Folgenden wird dargelegt, welche Trennlinien als Grundlage für die Auswahl zwischen den Schichten (Präsentationsschicht, Logikschicht, Datenhaltungsschicht) gewählt worden sind. Außerdem wird ein Überblick gegeben, welche Konfigurationen notwendig sind, um lauffähige Instanzen der eingesetzten Servertechnologien bereitzustellen.

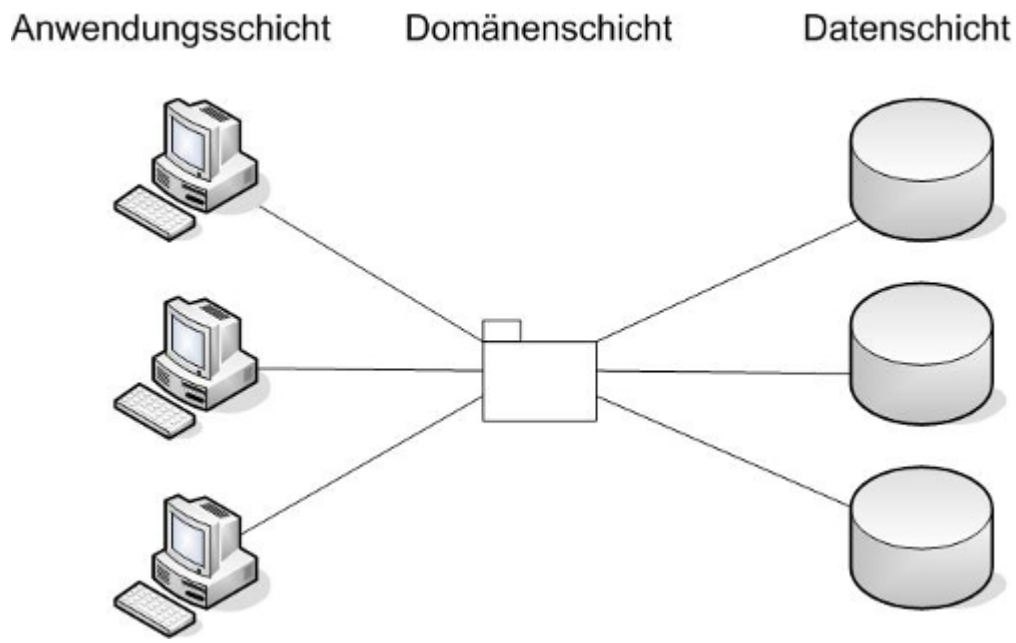

Abbildung 5.1.: 3-Schichten Modell

5.1. Eingesetzte Technologien und deren Nutzen

Wie in Kapitel 5 dargelegt finden sich in den eingesetzten Technologien und Produkten viele ORACLE™ Applikationen. Die folgende Liste gibt einen Überblick über die für den privaten Gebrauch frei verfügbaren Produkte:

- ORACLE™ Application Express 4.2
- ORACLE™ Mapviewer 11.1.1.7.B140717
- ORACLE™ Mapbuilder 11.1.1.7.B140717
- ORACLE™ SQL Developer 4.1.0.17
- ORACLE™ Analytical Workspace Manager 12.1.0.2.0
- Google Maps™ API V3

Im folgenden Kapitel wird im Groben darauf eingegangen, wie und warum die eingesetzte Technologie installiert und konfiguriert worden ist. Da es sich bei den eingesetzten Technologien um sehr spezialisierte und mächtige Software handelt und diese auf eine Art und Weise benutzt worden ist, die dafür gar nicht oder nur teilweise entwickelt wurde, ist es nicht Teil dieser Arbeit, jeden atomaren Installations- oder Konfigurationsschritt zu dokumentieren.

5.1.1. KML Parser

Für die in Kapitel 3 erstellten Lochbeschreibungen in KML (Abschnitt 3) Format wurde die ebenfalls von ORACLE™ entwickelte Programmiersprache JAVA™ (Version 8 Update 25) als Entwicklungssprache zum Schreiben eines gültigen Parsers verwendet. Der Parser nimmt als Argument einen gültigen Pfad zu einer KML Datei oder einen gültigen Pfad zu einem Ordner, der nach der Definition von Google Earth™ gültige KML Dateien enthält. Ein Beispielaufruf ist in Listing 5.1 abgebildet.

```
1 java jar kmlparser.jar ./kml/hole_2.kml
```

Listing 5.1: Beispielaufruf des KML Parser, in dem der KML Parser für Loch 2 ausgeführt wird

Die Abbildung A.6 zeigt das Klassendiagramm, welches den Aufbau des KML Parsers abbildet. In Abbildung A.6 ist außerdem zu erkennen, dass eine natürliche Abbildung der SQL Entitäten (beschrieben in Abbildung A.5) in JAVA vorgenommen worden ist. Dadurch konnte leichter eine Beziehung zwischen der Datenbasis und der Logik des Parser hergestellt werden. Das Sequenzdiagramm in Abbildung A.7 zeigt den Ablauf des Parsingalgorithmus anhand des

aufrufenden Beispiels 5.1. Nach dem Parsen ist das Loch direkt in der Datenbank verfügbar. Außerdem steht aufgrund des in 4.5 eingeführten Trigger zusätzlich die Länge des Loches direkt zur Verfügung. Abbildung A.1 zeigt die Auflistung der Löcher direkt nach dem Parsen der aus Google Earth™ erstellten KML Beschreibungen.

5.1.2. APEX

ORACLE™ APEX ist eine web-basierte Softwareentwicklungsumgebung, die direkt auf der ORACLE™ Datenbank läuft. In dieser Arbeit wird nicht weiter auf die Vor- und Nachteile von ORACLE™ APEX eingegangen. Teil dieser Arbeit ist der Nutzen von APEX als Entwicklungshilfe der Präsentationsschicht.
ORACLE™ bietet 3 Varianten, um die APEX Schnittstelle bereitzustellen:

- ORACLE™ HTTP Server mit mod_plsql
- Embedded PL/SQL Gateway
- ORACLE™ APEX Listener

Für die in dieser Arbeit entwickelte Applikation ist eine Instanz des ORACLE™ HTTP Servers eine notwendige Bedingung, da dieser Konfigurationsmöglichkeiten bietet, um die ganze Funktionalität der mitarbeitenden Programme voll auszunutzen.
Nachdem der OHS installiert ist, muss das Modul mod_plsql eingebunden werden Dieses stellt durch seine Konfiguration (Abgebildet in Listing 5.2) automatisch eine Verbindung zu der Datenbank her. Listing 5.2 zeigt einen DAD[1] (Darshane, 2003), der für die Umgebung */apex* gilt. Über die TNS[2] Zeichenfolge (Zeile 14) ist dem Modul mod_plsql und somit auch APEX bekannt, welcher Server mit welcher SID[3] als Datenbasis heranzuziehen ist.
Des weiteren stechen die Zeilen 5-7 in Listing 5.2 heraus — diese Proxy Einstellungen sind eigens dafür da, dass der ORACLE™ Mapviewer (beschreiben in Abschnitt 5.1.3) für den OHS erreichbar ist.
Nachdem der OHS erfolgreich installiert und konfiguriert worden ist, ist APEX unter der URL *http://localhost:7777/apex* erreichbar, wobei der Port 7777 der im Allgemeinen verwendete Port einer OHS Instanz ist. An dieser Stelle kann natürlich auch jeder beliebige freie Port des Systems verwendet werden.
Durch die Erreichbarkeit von APEX konnte jetzt mit der Einrichtung eines $WORKSPACE$

[1] Database Access Descriptor
[2] Transparent Network Substrate
[3] Oracle System ID

begonnen werden. Analog zu der SID der Datenbank wurde auch die Entwicklungsumgebung in APEX $GOLFAPP$ genannt, um eine Beziehung zwischen der Datenbasis und der durch APEX verwalteten Präsentationsschicht auf den ersten Blick sichtbar zu machen. Das Anlegen der Applikation $webapp$ innerhalb dieses $WORKSPACE$ schließt die Konfiguration von APEX ab.

```
1  Alias /i/ "images/"
2  AddType text/xml xbl
3  AddType text/x component htc
4
5  ProxyRequests On
6  ProxyVia      Off
7  ProxyPass /mapviewer http://localhost:8888/mapviewer
8
9  <Location /apex>
10         Order deny,allow
11         PlsqlDocumentPath docs
12         AllowOverride None
13         PlsqlDocumentProcedure wwv_flow_file_mgr.process_download
14         PlsqlDatabaseConnectString localhost:1521:golfapp
15         PlsqlNLSLanguage AMERICAN_AMERICA.AL32UTF8
16         PlsqlAuthenticationMode Basic
17         SetHandler pls_handler
18         PlsqlDocumentTablename wwv_flow_file_objects$
19         PlsqlDatabaseUsername benutzer
20         PlsqlDefaultPage apex
21         PlsqlDatabasePassword passwort
22         Allow from all
23 </Location>
```

Listing 5.2: mod_plsql DAD Konfiguration

5.1.3. Mapviewer

Der Oracle™ Fusion Middleware MapViewer, im Folgenden nur Mapviewer genannt, ist eine Entwicklungshilfe, um *Spatials* und Graphen, die in einer Datenbank liegen, zu visualisieren. In dieser Arbeit wird der Mapviewer benutzt, um eine Schnittstelle zwischen Datenbank und APEX zu schaffen. In den in Abbildung 5.1 angezeigten Schichten nimmt der Mapviewer Schicht 2 die Domänen- oder Logikschicht, ein.
Wie schon für APEX muss auch für den Mapviewer ein Server installiert werden, auf dem die Applikation läuft. Auch hier bieten sich wegen der Kompatibilität Anwendungsserver

von ORACLE™ an. Wie für APEX werden auch für einen Anwendungsserver verschiedene Möglichkeiten geboten, die Schnittstellenfunktionen vom Mapviewer nach außen sichtbar zu machen:

- GlassFish Server
- Oracle Containers for J2EE
- WebLogic Server

In dieser Arbeit wird der Mapviewer auf einem GlassFish Server laufen, da ORACLE™ ein *Quickstart-Kit* für den Mapviewer auf GlassFish Basis vorkonfiguriert anbietet.
Da der Mapviewer eine Schnittstelle zwischen APEX und Datenbank schafft, muss auch der Mapviewer wissen, welche Datenbank als Datenbasis genutzt wird. Hierfür müssen einige MDS[4] Konfigurationen vorgenommen werden. Da es sich um sehr atomare Änderungen handelt, werden diese nicht Teil dieser Arbeit sein und es wird eine gültige und funktionierende Konfiguration angenommen.
Jetzt weiß der Mapviewer, woher er die Geodaten beziehen muss, es ist allerdings noch nicht bekannt, wie diese Daten aussehen und wie diese verarbeitet und dargestellt werden sollen. Hierzu werden sogenannte $Tilelayer$ erstellt und später über den Mapbuilder Geodaten zugewiesen. Daraus ergibt sich nach außen die 3-Tier Architektur, die in Abbildung 5.2 zu sehen ist.

5.1.4. Mapbuilder

Oracle Map Builder, im Folgenden nur Mapbuilder genannt, ist eine eigenständige Anwendung, die zum Verwalten, Erstellen und Zuordnen von Metadaten entwickelt worden ist. Der Mapbuilder arbeitet über den Mapviewer direkt mit der Datenbank, sodass keine weiteren Programme installiert werden müssen. Im Grunde genommen ist der Mapbuilder nur ein Tool, um die Layer, die in XML[5] verfasst sind, GUI basiert zu erstellen.
Die Mapviewer API kann mit vielen verschiedenen Arten von Metadaten parametrisiert werden, natürlich werden für diese Arbeit nur eine Teilmenge dieser verwendet. Die folgende Liste gibt einen Auszug über die Masse der Metadaten, die durch den Mapbuilder verwaltet werden können — jeweils das letzte Element eines Typs wird für diese Arbeit verwendet.

- Styles

[4] Map Data Source
[5] Extended Markup Language

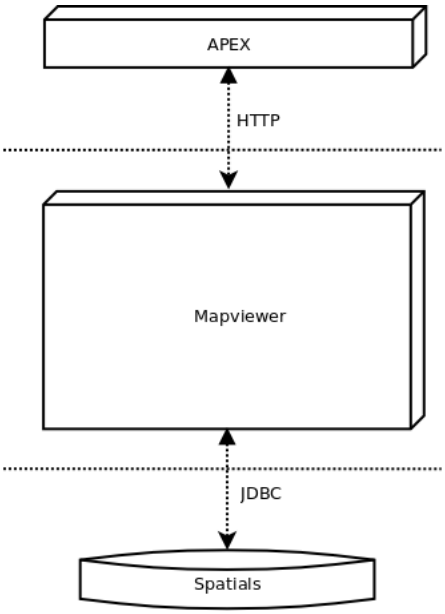

Abbildung 5.2.: Aktuelle 3-Tier Architektur

  - Color
  - Marker
  - Area
  - ...
- Themes
  - Network
  - Image
  - Geometry
  - ...
- Maps
  - Title
  - Legend
  - Basemap
  - ...

Die Reihenfolge der Liste entspricht auch der Reihenfolge, in welcher die Elemente erstellt werden müssen, denn für jede map werden $themes$ und für jedes $theme$ die zugehörigen $styles$ benötigt. Für die Darstellung der $SDO\_GEOMETRY$-Datentypen aus den Tabellenspalten beschränken wir uns bei den $maps$ auf $basemaps$, bei den $themes$ auf $geometries$ und bei den $styles$ aus $areas$.

Styles

Um dem Mapviewer generisch beizubringen, wie verschiedene Ressourcen angezeigt werden sollen, müssen $styles$ definiert werden, also die Darstellung der Geometrien nach außen. Bei der Darstellung eines Golfplatzes liegt es nahe, auch die Entitäten der Abbildung A.5 in $styles$ zu modellieren. Für diese Arbeit werden nur zwei Arten von $styles$ benötigt, die $area$ und der $marker$. Als $area$ werden die Geometrien $hazard$, $green$, $fairway$ und tee als farbige Flächen, passend zu ihren natürlichen Farben, und die Punkte $STROKE.FROM$ und $STROKE.TO$ als $marker$ dargestellt.

Themes

Die eigentliche Verbindung zwischen einer Darstellungsform ($style$) und der Datenbasis (Spalte in einer Tabelle) bauen die $themes$ auf. Sie sind das Verbindungsglied zwischen den in XML abgelegten $styles$ und den in einer Datenbank gespeicherten geometrischen Formen. In dieser Arbeit wird, da nur Polygonen und Punkte verwendet werden, nur das $theme$ *Geometry* benötigt. Abbildung 5.3 zeigt die Zuordnungen der Datenbasis zu den einzelnen $themes$, wie sie über den Mapbuilder vorgenommen worden sind. Weiter kann jetzt jedem $theme$ seine Darstellungsform beigebracht werden, dementsprechend werden die im vorigen Abschnitt definierten $styles$ jetzt einem oder mehreren $themes$ zugeordnet. So ist später dem Mapviewer bekannt, wie eine Geometrie angezeigt werden soll.

Maps

Die Karten, die in dem Mapbuilder erstellt werden können, sind direkt auch im Mapviewer zu verwenden und können als Schicht über eine Basiskarte gelegt werden. Für die Ausgabe über den Mapviewer wird eine sogannte $basemap$ erstellt. Jeder $basemap$ können Themes zugeordnet werden, die dann später auf der von dieser Karten erstellten Schicht dargestellt werden. In dieser Arbeit wird diese Funktion dazu genutzt, um alle Geometrien eines Golfplatzes als eine Karte darzustellen, um dem Benutzer eine Übersicht über die nächsten Löcher darstellen zu können.

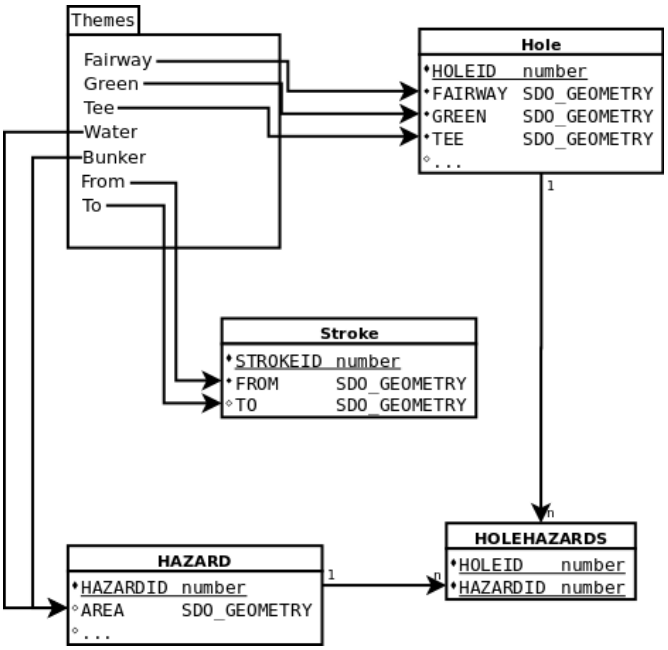

Abbildung 5.3.: Zuordnung der Themes zu der Datenbasis

Anwendung

Nimmt man jetzt alle diese Zuordnungen vor, ist es möglich über APEX eine Verbindung zum Mapviewer herzustellen. Da in diesem jetzt die vorher erwähnten Attribute hinterlegt worden sind, kann die API von Oracle Maps eine Geometrie wie gewünscht darstellen. Abbildung A.9 zeigt die aus der Datenbank gezogene Geometrie, die das Loch 2 vom Golfclub Adendorf überlagert.

5.1.5. Analytical Workspace Manager

Die vorhergehenden Technologien bezogen sich ausschließlich auf die Geoentwicklung der Datenbank und schlossen die Umsetzung der in Abschnitt 4.2 entwickelten Ideen vollkommen aus. Da die Auswertung der Statistiken aus der Datenbank über BI-Ansätze erfolgen soll, wird ein von ORACLE™ entwickeltes Werkzeug für die Erstellung der OLAP Komponenten (2.1) benutzt, der ORACLE™ Analytical Workspace Manager, im Folgenden AWM genannt, in Version 12.1.0.2.0.
Der Workspace Manager erlaubt es dem Benutzer, GUI basiert eine analytische Werkbank für OLAP Zwecke zu erstellen und zu verwalten. Diese Entwicklungsumgebung vereinfacht zudem das Erstellen der in Abschnitt 4.2 ausgearbeiteten *dimensions*, *hierarchies*, *cubes* und

measures. Außerdem kann über den AWM eine Übersetzung (*mapping*) zwischen den analytischen Entitäten und der Datenbasis hergestellt werden.

Dimensionstabellen

Eine Voraussetzung für das Zuordnen der Daten in OLAP Form ist das Vorhandensein von Dimensionstabellen, die dann die logische Basis für jeden Würfel bilden. Wie in dem Abschnitt 4.2.4 beschrieben, muss jetzt für jede Dimension eine Tabelle mit den beschriebenen Hierarchien angelegt werden. Ausgehend von dem Entwurf aus Abschnitt 4.2 ergibt sich für die Dimensionen das in Abbildung 5.4 gezeigte ER Diagramm.

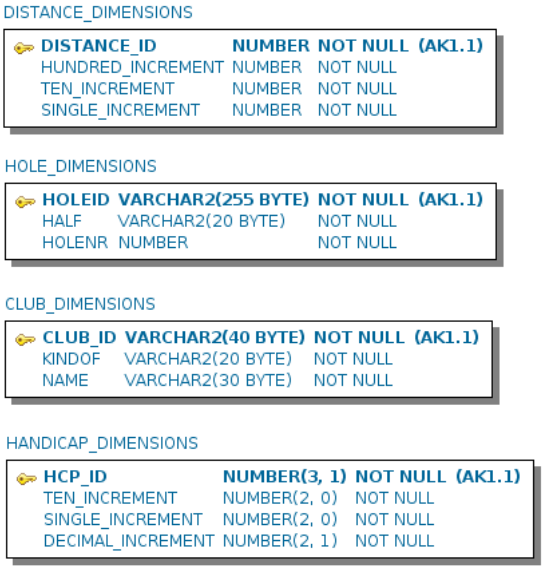

Abbildung 5.4.: Abbildung der Dimensiontabellen

Faktentabellen

Um jetzt die Datenbasis für Kenndaten zu schaffen, müssen Faktentabellen für jeden Messwert (*measure*) angelegt werden. In Abschnitt 4.2.4 wurden bereits die beiden Faktentabellen definiert, die die Kennziffern *length* und *quality* halten.

Cubes

Die eigentlich zu benutzende Grundlage für die Analyse bilden dann die Datenwürfel, die sich aus den in Abbildung 5.4 gezeigten Dimensionen zusammensetzen und an deren Endpunkt die Messwerte Qualität, die zurückgelegte Distanz und der statistische Vorschlag eines Golfschlags stehen.

5.2. Architektur

Ausgehend von den in Abschnitt 5.1 eingesetzten Technologien kann nach deren Anwendungen auf den Entwurf ein vollständiges Bild der Topologie geschaffen werden. Abbildung 5.2 bietet bereits nach dem Einsatz des Mapviewers einen ersten Überblick, doch nach dem Einsatz der OLAP Technologien ergeben sich neue Aspekte auf das Layout und damit der Architektur des Programms. Analog zum Mapviewer, dieser auch nicht direkt auf der Datenbank sitzt, sondern in seinem Fall per JDBC auf die Geoinformationen zugreift, liegen die Fakten direkt in der Datenbank, wobei die Aktionen auf diese über eine externe OLAP API gesteuert und verwaltet werden. Diese OLAP API bietet dann auch die Schnittstellen an, um per HTTP Informationen an APEX, bzw. den HTTP Webserver, weiterzugeben.
In Abbildung 5.5 ist des Weiteren zu erkennen, dass der Mapviewer nicht zum Ganzen als *Blackbox* -System arbeitet, sondern dass durchaus bekannt ist und auch bekannt sein muss, welche Funktionen intern durchgeführt werden.

5.3. DSS - Decision Support System

In den bisherigen Abschnitten der Implementierung wurden nur die vorher entworfenen Datenbankkomponenten der Arbeit implementiert. Dadurch steht jetzt die Datengrundlage, um dem Benutzer (Golfer) bei gegebenen Parametern (Dimensionen) einen Schläger vorzuschlagen. Hierbei beschränkt sich die Arbeit auf einen Prototypen, der mit Hilfe der Position des Golfers (per manueller Eingabe) den statistisch gesehen besten Schläger auswählt.
Bevor an eine Implementierung eines solchen Systems gedacht werden kann, muss klar abgegrenzt werden, welche Faktoren in Betracht gezogen werden müssen. Dies ist zu einem Großteil schon in Abschnitt 1.2 gemacht worden. Das heißt im Grunde, dass die Position des Balles und das angepeilte Ziel entscheidend sind. Hinzu kommen jetzt die Lagen von Hindernissen und die Auswertung der Informationen aus der Datenbasis. Da diese Informationen sich nicht gleich bewerten lassen, wird im Laufe dieser Arbeit auf die Bewertung von Hindernissen verzichtet.

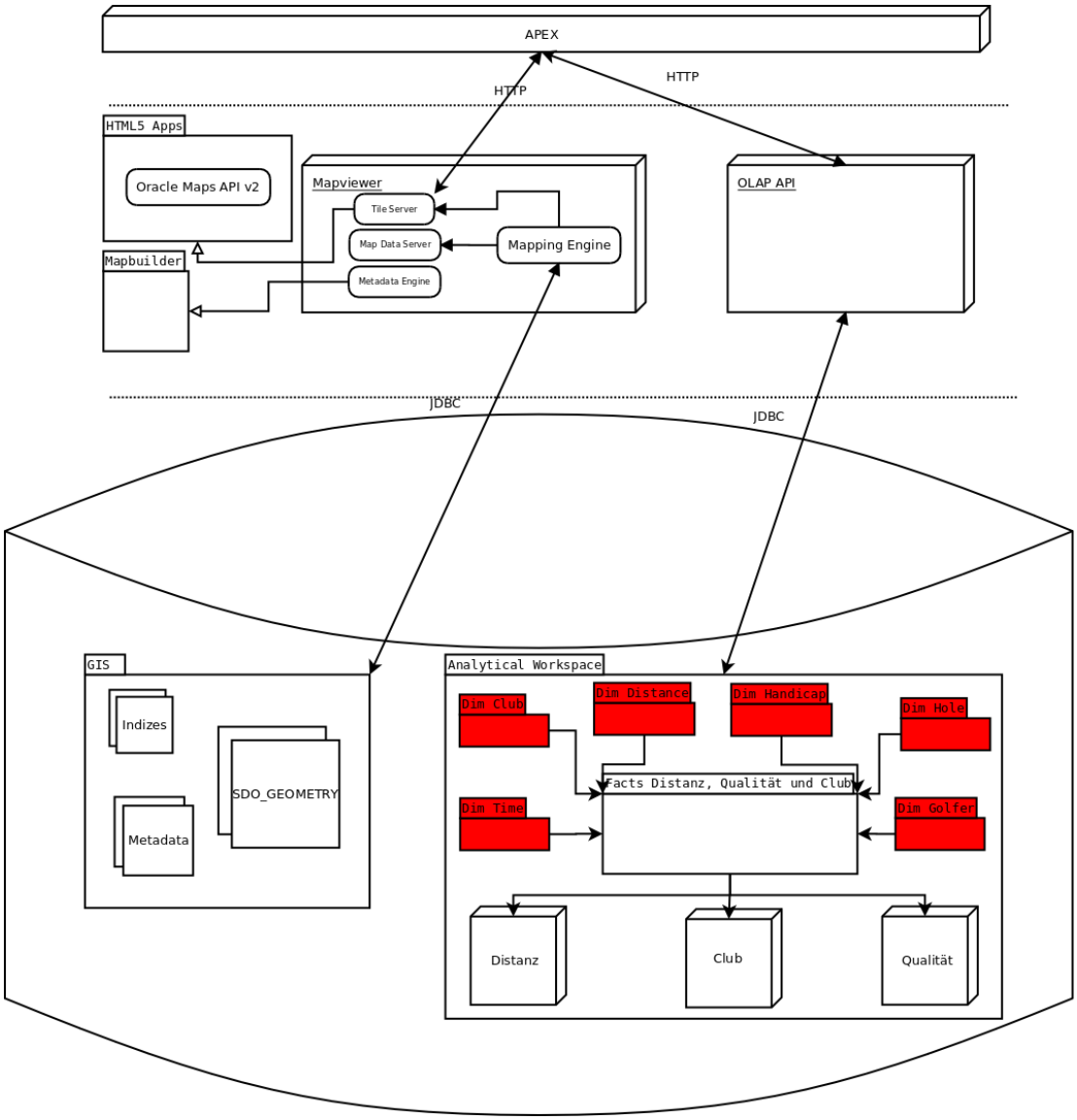

Abbildung 5.5.: Detaillierte Abbildung der finalen Topologie des Systems

Für das DSS kommt jetzt nur der Datenwürfel der *clubprediction* aus Abschnitt 4.2.4 in Frage. Dieser wird durch die Dimensionen der Distanz, des Golfers und der Zeit aufgespannt. Um jetzt einen Vorschlag für einen Schläger zu erhalten, müssen also auch diese Informationen vorliegen.
Die Berechnung der Distanz ergibt sich in diesem prototypischen Fall aus der Entfernung zwischen der aktuellen Position des Balles und des Grüns von dem gerade bespielten Golfloch. Für diese Berechnung wurde die SQL Funktion $DISTANCETOGREEN$(5.3) eingeführt, welche den eindeutigen Identifizierer des Lochs selber und die aktuelle Position in Längen- und Breitengrad akzeptiert. Zurückgegeben wird dann die Entfernung in Metern.

```
1  create or replace FUNCTION DISTANCETOGREEN(
2      HOLE_ID          IN NUMBER ,
3      POS_LATITUDE     IN VARCHAR2 ,
4      POS_LONGITUDE    IN VARCHAR2 )
5    RETURN NUMBER
6  IS
7    DISTANCE NUMBER;
8    POS SDO_GEOMETRY;
9    GREEN SDO_GEOMETRY;
10 BEGIN
11   SELECT SDO_GEOMETRY(
12                     2001,
13                     4326,
14                     SDO_POINT_TYPE(
15                               POS_LONGITUDE,
16                               POS_LATITUDE,
17                               NULL),
18                     NULL,
19                     NULL)
20   INTO POS
21   FROM dual;
22   SELECT GREEN INTO GREEN FROM HOLE WHERE HOLEID = HOLE_ID;
23   RETURN ROUND(sdo_geom.sdo_distance(POS, GREEN, 0.001, 'unit=M'));
24 END DISTANCETOGREEN;
```

Listing 5.3: Funktion zum Berechnen der Distanz zwischen Grün und aktueller Position

Einfach verhält es sich mit dem Füllen der Dimensionen des Lochs und der Zeit. Das Loch ergibt sich aus dem gerade bespielten Lochs und die Statistik der Zeit spannt sich über alle Daten auf, an denen ein Golfschlag im System registriert worden ist. Natürlich ist es über die OLAP Funktionen möglich, auch die Zeit einzugrenzen. Dennoch wird in dem Kontext der Arbeit darauf verzichtet, da für eine Einschränkung auch die Masse an Daten zur Verfügung

stehen muss. Für eine statistische Signifikanz für das Vorschlagen eines Golfschlägers reichen in diesem Fall 100 Schläge in der Datenbank aus.
Da eine Abfrage an einen OLAP-Würfel durchaus sehr komplex werden kann, wurde auch in diesem Fall eine PL/SQL-Funktion geschrieben, die den Vorschlag returniert. Die Funktion $PREDICTION$(5.4) nimmt als Parameter eben die Dimensionen des Golfers und des Lochs an. Außerdem werden auch hier, wie in der vorigen Funktion, Längen- und Breitengrad mitgegeben. Innerhalb dieser Funktion wird dann mit Hilfe der Funktion $DISTANCETOGREEN$(5.3) die dritte Dimension berechnet. Hier zeigt sich die Schwäche oder vielmehr Inkompatibilität von OLAP zu nicht betriebswirtschaftlichen Daten. Für das Selektieren des Kennwerts müssen Aggregatfunktionen verwendet werden. Diese Aggregatfunktionen nehmen aber nur gültige Zahlen als Parameter an. Da der Golfer aber einen Schläger vorgeschlagen haben möchte, der als Zeichenkette in der Datenbank steht, musste jedem Schläger ein Wertigkeit gegeben werden. So sind die Schläger nach Schlaglänge in der Datenbank über ihren Primärschlüssel sortiert. Dadurch hat ein 9-Eisen bei einer erwarteten Schlaglänge von 120-135 Meter den Schlüssel 4 und ein 3-Holz bei einer erwarteten Schlaglänge von 200-220 Meter den Schlüssel 10. Das System benutzt dann die Schlüssel, um über die AVG Aggregatfunktion einen Durchschnitt des Schlägers zu berechnen. Hat das System jetzt den Durchschnitt der Schlüssel berechnet, kann über den Schlüssel wieder eine Zuordnung zu dem Schläger gemacht werden und der Spieler erhält den Vorschlag als aussagekräftige Zeichenkette.

```
1  create or replace FUNCTION PREDICTION(
2      GOLFER          IN VARCHAR2,
3      HOLE_ID         IN NUMBER  ,
4      POS_LATITUDE    IN NUMBER  ,
5      POS_LONGITUDE   IN NUMBER  )
6    RETURN VARCHAR2
7  IS
8    predicted_club_id CLUBPREDICTION_VIEW.CLUB%TYPE;
9    predicted_club       VARCHAR2(32767);
10   distance NUMBER;
11   hole_description HOLE.DESCRIPTION%TYPE;
12 BEGIN
13   predicted_club_id    := NULL;
14   distance := DISTANCETOGREEN(HOLE_ID,POS_LATITUDE,POS_LONGITUDE);
15   SELECT ROUND(CLUB)
16   INTO predicted_club_id
17   FROM CLUBPREDICTION_VIEW,
18     DIM_TIME_TIME_HIERARCHIE_VIEW,
19     DIM_GOLFER_GOLFER_HIERARC_VIEW,
20     DIM_DISTANCE_DIM_DISTANCE_VIEW
```

```
21    WHERE
22    CLUBPREDICTION_VIEW.DIM_TIME = DIM_TIME_TIME_HIERARCHIE_VIEW.DIM_KEY
23    AND
24    CLUBPREDICTION_VIEW.DIM_GOLFER = DIM_GOLFER_GOLFER_HIERARC_VIEW.DIM_KEY
25    AND
26    CLUBPREDICTION_VIEW.DIM_DISTANCE = DIM_DISTANCE_DIM_DISTANCE_VIEW.DIM_KEY
27    AND (DIM_TIME_TIME_HIERARCHIE_VIEW.LEVEL_NAME IN ('YEAR'))
28    AND (DIM_GOLFER_GOLFER_HIERARC_VIEW.DIM_KEY    IN (GOLFER))
29    AND (DIM_DISTANCE_DIM_DISTANCE_VIEW.DIM_KEY    IN ('DISTANCE_'
30       || TO_CHAR(distance)));
31    SELECT NAME
32    INTO predicted_club
33    FROM CLUB_DIMENSIONS
34    WHERE CLUB_ID = predicted_club_id;
35    RETURN predicted_club;
36  EXCEPTION
37  WHEN NO_DATA_FOUND THEN
38    RETURN 'Es konnten keine Statistik zu der Entfernung ' ||
39                              TO_CHAR(calculated_distance) ||
40                              'm gefunden werden';
41  END PREDICTION;
```

Listing 5.4: Funktion zum Auswählen eines Schlägers

Beispiel

Das folgende Szenario durchläuft einmal komplett das in diesem Abschnitt vorgestellte DSS. Als Loch wird das Loch 2 des Adendorfer Golfclubs gewählt, welches in der Datenbank die ID 24 hat. Für die Dimension des Golfers wird die ID 1 verwendet, wohinter der Golfer mit der Bezeichnung „Dennis Haseloff“steht. Außerdem wird der Wert 53.29391101469918 als Breiten- und 10.45828402416817 als Längengrad der Position des Balles verwendet. Abbildung A.2 zeigt diese definierten Vorbedingungen. Klickt der Benutzer jetzt auf die Schaltfläche „Schlägervorschlag“, werden über eine dynamische APEX Aktion die PL/SQL-Funktionen $PREDICTION$ und $DISTANCETOGREEN$ mit den aktuellen Vorbedingungen aufgerufen. Innerhalb der Funktion $PREDICTION$ werden in der Abfrage gegenüber der OLAP Instanz dann die Schlüssel der vorher für diese Distanz genutzten Schläger geholt. Die Distanz wurde vorher in der Funktion $DISTANCETOGREEN$ auf 98 Meter bestimmt. Aus diesem Wert ergeben sich die Schläger mit dem Schlüssel 2 (lob wedge) und 3 (sand wedge). Daraus wird jetzt über die Aggregatfunktion des OLAP-Würfels der Durchschnitt von 2.5 errechnet. Da die Schlüssel aber immer ganzzahlige Werte haben, muss jetzt auf der Fließkom-

mazahl von 2.5 ein $ROUND$ ausgeführt werden, um einen gültigen ganzzahligen Schlüssel zu erhalten. Mit dem dann bestimmten Schlüssel mit dem Wert 3 ordnet die Funktion diesem Identifizierer den Schläger „sand wedge“zu. Abbildung A.3 zeigt die Ausgabe einer erfolgreichen Abfrage des Schlägers, der die statistisch gesehen bestmögliche Längenkontrolle über den kommenden Schlag bereitstellt.

6. Fazit

Abschließend in dieser Arbeit sollen die vorangegangenen Inhalte rekapitulierend betrachtet werden. Auf Basis der Zielsetzung, der Entwicklung einer prototypischen App zur Auswertung von Statistiken im Golf, sah die Ausgangslage wie folgt aus; Bisher wurden in vergleichbaren Anwendungen entweder OLAP oder Spatials verwendet. Nie wurden diese beiden Techniken miteinander verbunden, sodass die Synthese der beiden Gebiete eine große Herausforderung darstellte. Schon um ein grundlegendes Verständnis der Theorie hinter OLAP zu erlangen, wäre es möglich, eine komplette Abschlussarbeit mit dieser Thematik zu befüllen. Analog verhält es sich dazu mit den Geoinformationssystemen. Im Laufe der Arbeit wurde klar, dass sich mit Hilfe von immer nur einer Technik das Ziel dieser Arbeit erreichen lassen würde. So wäre es möglich, alleine mit Hilfe von ORACLE™ Spatial und den Grundfunktionen von relationalen Datenbanken die Anwendung zu realisieren. Andersherum wäre es allerdings auch möglich gewesen, mit den reinen Statistikfunktionen von OLAP und der rohen Masse der Daten eine App zu entwickeln, die es einem Hobbygolfer ermöglicht, seinen Score zu verbessern. Die eigentliche Herausforderung der Arbeit war also gar nicht die Entwicklung der App, sondern die Verbindung der Techniken zueinander, sodass diese sich nicht im Weg standen, sondern kooperativ zusammenarbeiteten.
Am Anfang dieser Arbeit stand die These, ob eine programmunterstützte Golfrunde das eigentliche Golfspiel verbessern kann. Diese These wurde durch die Ergebnisse und Auswertungen in dieser Arbeit bestätigt. Betrachtet man jetzt die reinen Informationen, die dem Golfspieler durch das Geoinformationssystem geliefert werden, so reichen die Daten über die Ausmaße und die Verteidigung durch Hindernisse eines Grüns aus, um bestimmte Parameter am Schlag und der Ausführung des Schwungs anzupassen. Alleine diese Informationen werden nur durch den Spatial-Anteil der Anwendung geliefert. Addiert man jetzt noch die reinen Statistiken der Auswertung der OLAP Daten hinzu, so erhält der Benutzer nicht nur rohe Daten, sondern auch greifbare und sofort umzusetzende Informationen, welcher Schlag und Schläger das statistisch gesehen beste Resultat liefern wird.
Ob die Herangehensweise mithilfe von Datenbankfunktionen die einzig Wahre ist, kann und wird nicht abschließend geklärt werden können. Natürlich gibt es andere Möglichkeiten, um

Daten zu analysieren und zu speichern. Dennoch ist es die Quintessenz dieser Arbeit, dass die datenbanklastige Option auf einem Level mit Lösungen, welche die Datenbank nur als reines Datenbehältnis verwenden, steht. Ein ganz klarer Vorteil der Konzentration der Logik in der Datenbank ist, dass die Daten immer und in jedem Kontext zur Verfügung stehen. Selbst die Entwicklung der GUI ist dank der Nutzung von APEX und dessen Eigenschaft, auf einer relationalen Datenbank zu basieren, als reiner Datenbankprozess anzusehen. Natürlich kann man jetzt argumentieren, dass die Trennung zwischen Logik und Datenhaltung einen besseren Gesamtüberblick liefert. Doch ist es nicht anstrengend und vor allem teuer, für jede Aktion immer eine Verbindung zur Datenbasis aufzubauen, in welcher Form auch immer diese vorliegt?
Insgesamt lässt diese Arbeit aber viel Raum für Erweiterungen und natürlich auch Verbesserungen. So ist es denkbar, eine App zu entwickeln, die die Daten im Mobilgerät auf der Golfrunde speichert und erst nach einem Synchroniserungsprozess die gesammelten Daten an die Datenbank weitergibt. Analog dazu würde dann auch eine Synchronisierung vor jeder Golfrunde von Nöten sein, um etwaige benötigte Daten auf dem Mobilgerät zu persistieren. In Verbindung dazu wäre es dann auch nötig, die Aggregatfunktionen zur Berechnung der Vorschläge in der mobilen Anwendung zu realisieren. Diese Herangehensweise würde es dann ermöglichen, auch offline die Statistiken (ohne Google Maps Kartenunterstützung) abrufen zu können, da die Logik, die vorher mit der Datenbank abgeglichen worden ist, auf dem Gerät vorliegt.
Auch inhaltlich ist das Thema noch lange nicht erschöpft. So sind während dieser Arbeit auch die Daten über die Qualität der Schläge in der Datenbasis hinterlegt worden. Das macht es möglich die Schläge auf einer Karte innerhalb der APEX Anwendung so zu visualisieren, dass der Benutzer direkt sieht ob seine Schläge von einem Punkt aus gut oder schlecht gewesen sind. Außerdem ist es denkbar noch viel mehr Statistiken, wie zum Beispiel Wetterdaten (Regen, Luftdruck, Temperatur etc.) auszuwerten, um noch genauere Vorschläge zu erhalten. Diese Daten müssten dann natürlich mit in die Fakten- und Dimensionstabellen integriert werden, welche aktuell den OLAP Entwurf bilden.
Doch ist Golf nicht immer so objektiv und deterministisch, wie es der Informatiker gerne hätte. Daher führt eine zu statistiklastige Auswertung der Schläge leider nicht immer zu dem erwarteten Ergebnis. Aus diesem Grund können derartige Apps, wie die in dieser Arbeit entworfene, immer nur eine Beihilfe zu einer guten Golfrunde sein. Das eigentliche Gros der Arbeit muss noch der Golfer selbst erledigen.

A. Anhang

Abbildung A.1.: Anzeige der Löcher eines Glofplatzes nach dem Parsen

Abbildung A.2.: Vorbedingungen des Beispiels zum Vorschlag eines Golfschlägers

Abbildung A.3.: Ausgabe nach dem Auslösen des Systems zum Vorschlagen des Schlägers

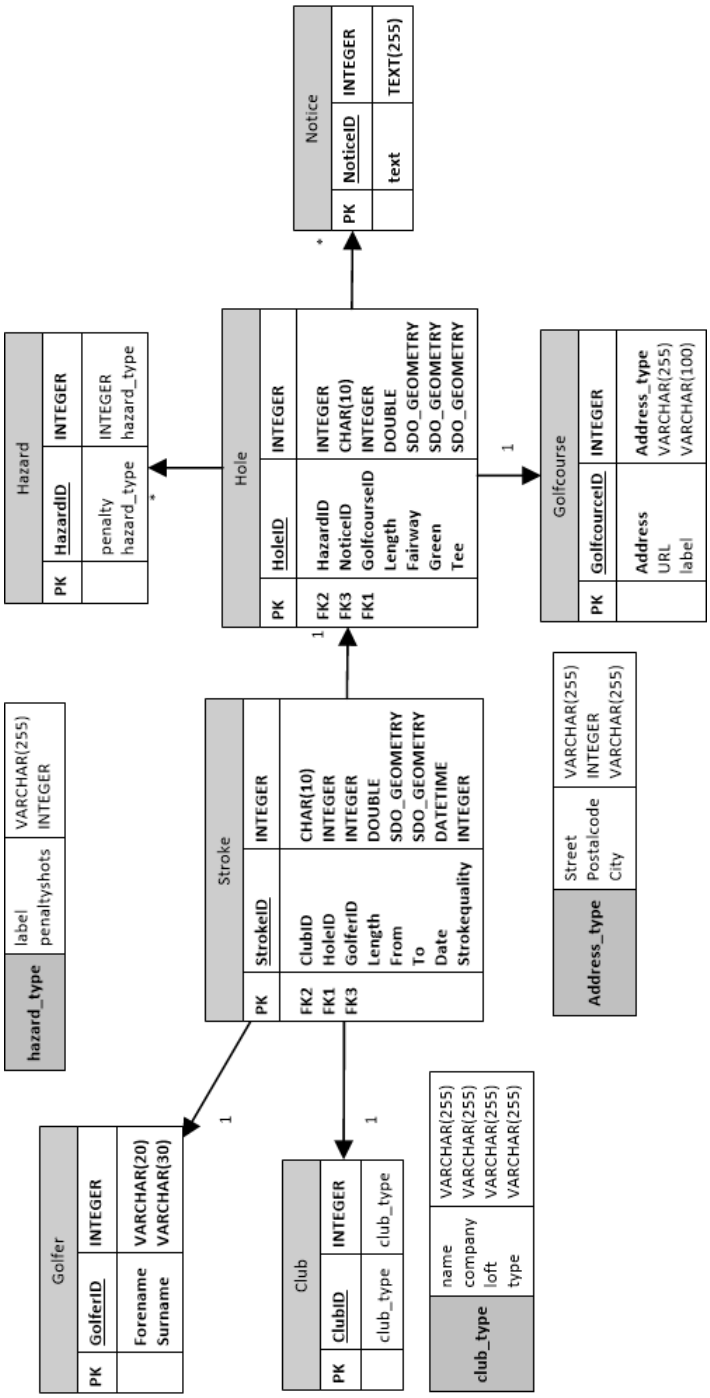

Abbildung A.4.: Entity-Relationship-Modell mit den Beziehungen der zu erstellenden Datenbank und den fachlichen Datentypen innerhalb der Tabellen

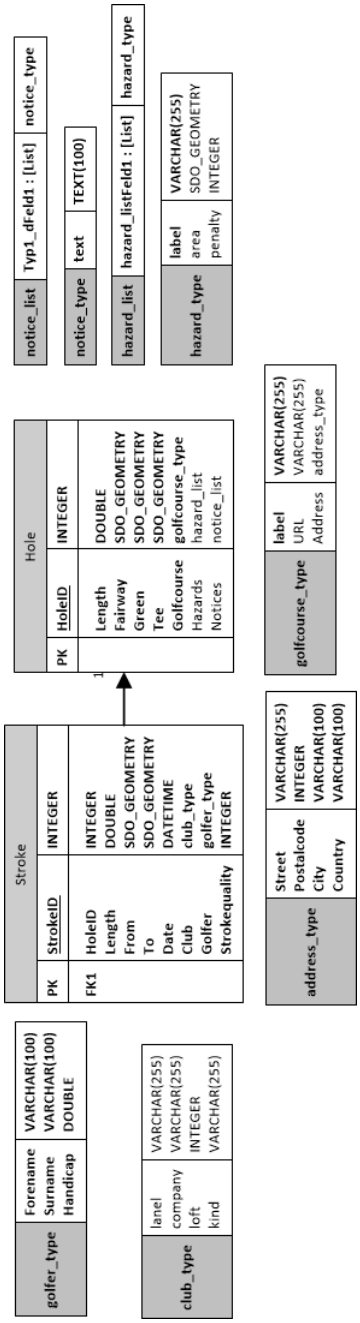

Abbildung A.5.: Hybridsystem aus relationalen und objektrelationalen Bausteinen

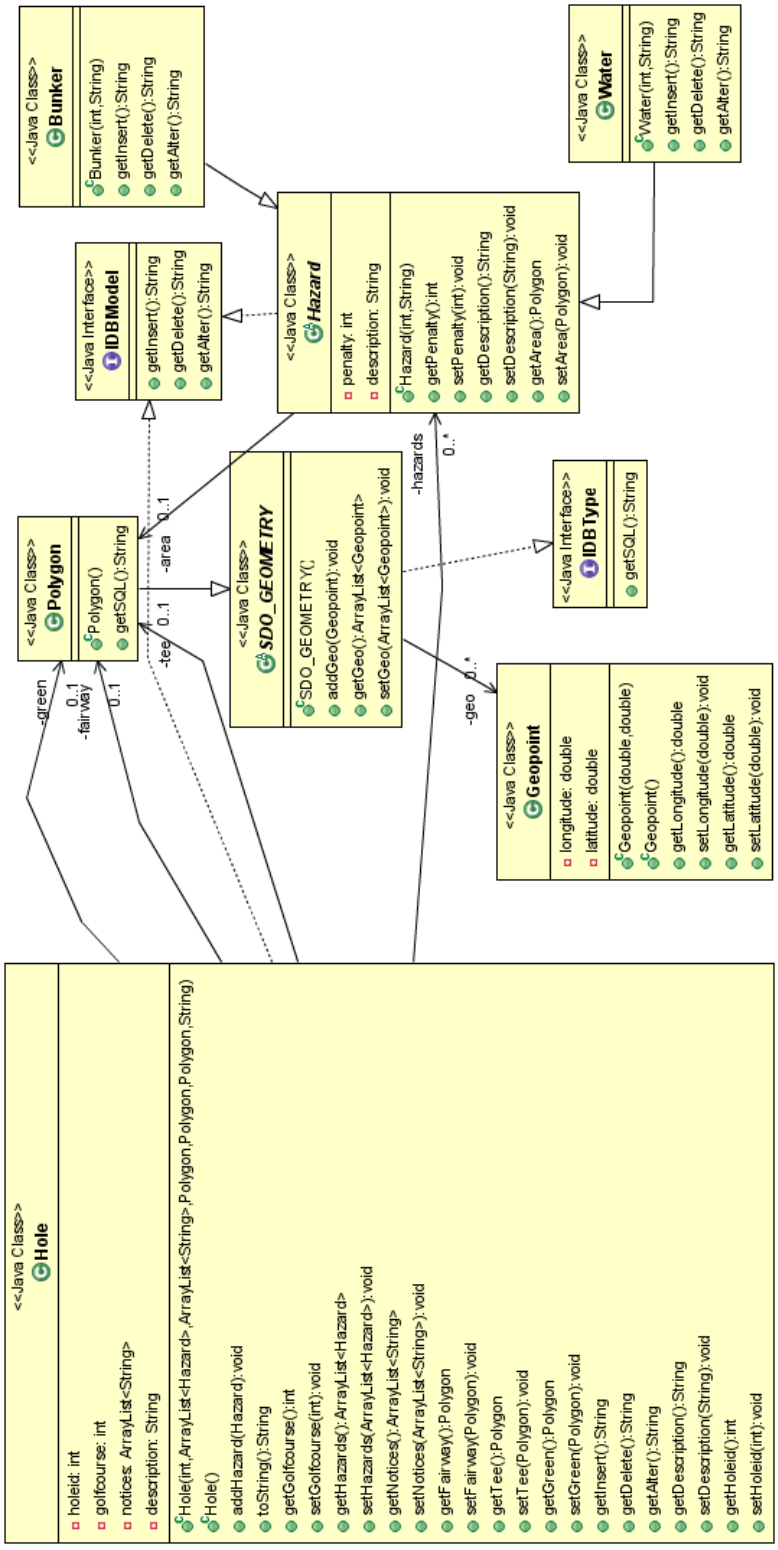

Abbildung A.6.: Klassendiagramm des KML Parser

Abbildung A.7.: Sequenzdiagramm des KML Parser

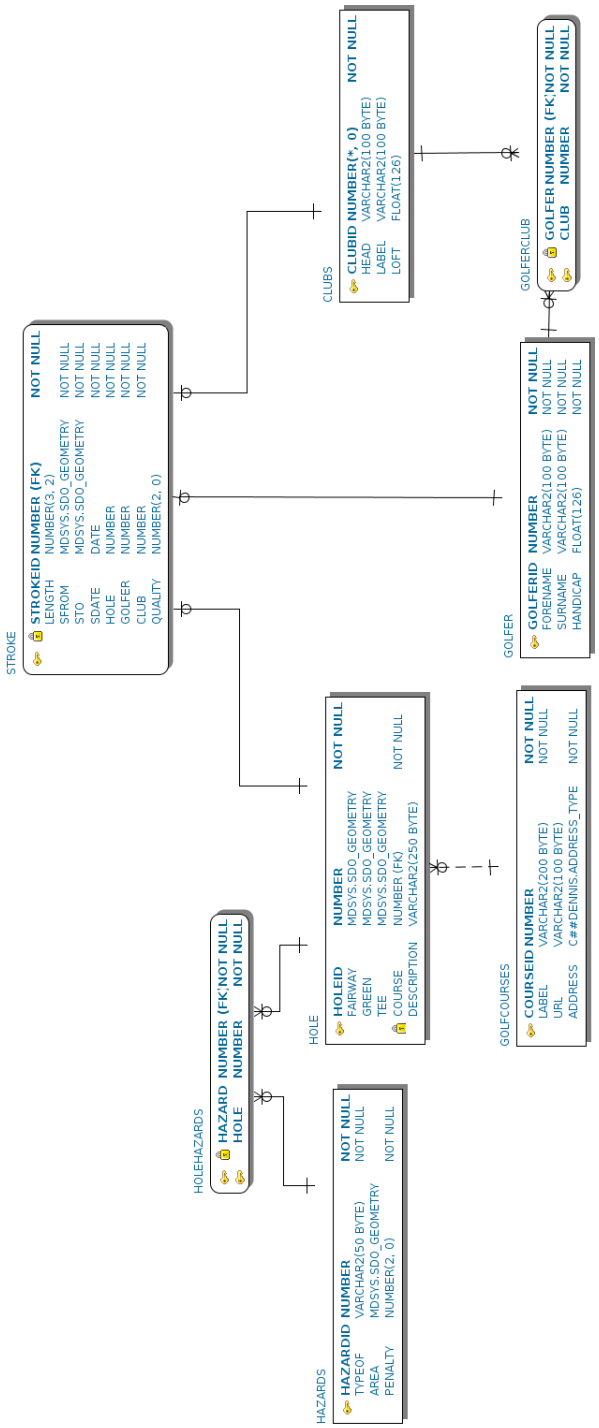

Abbildung A.8.: Detailiertes ERM der entworfenen Datenbank

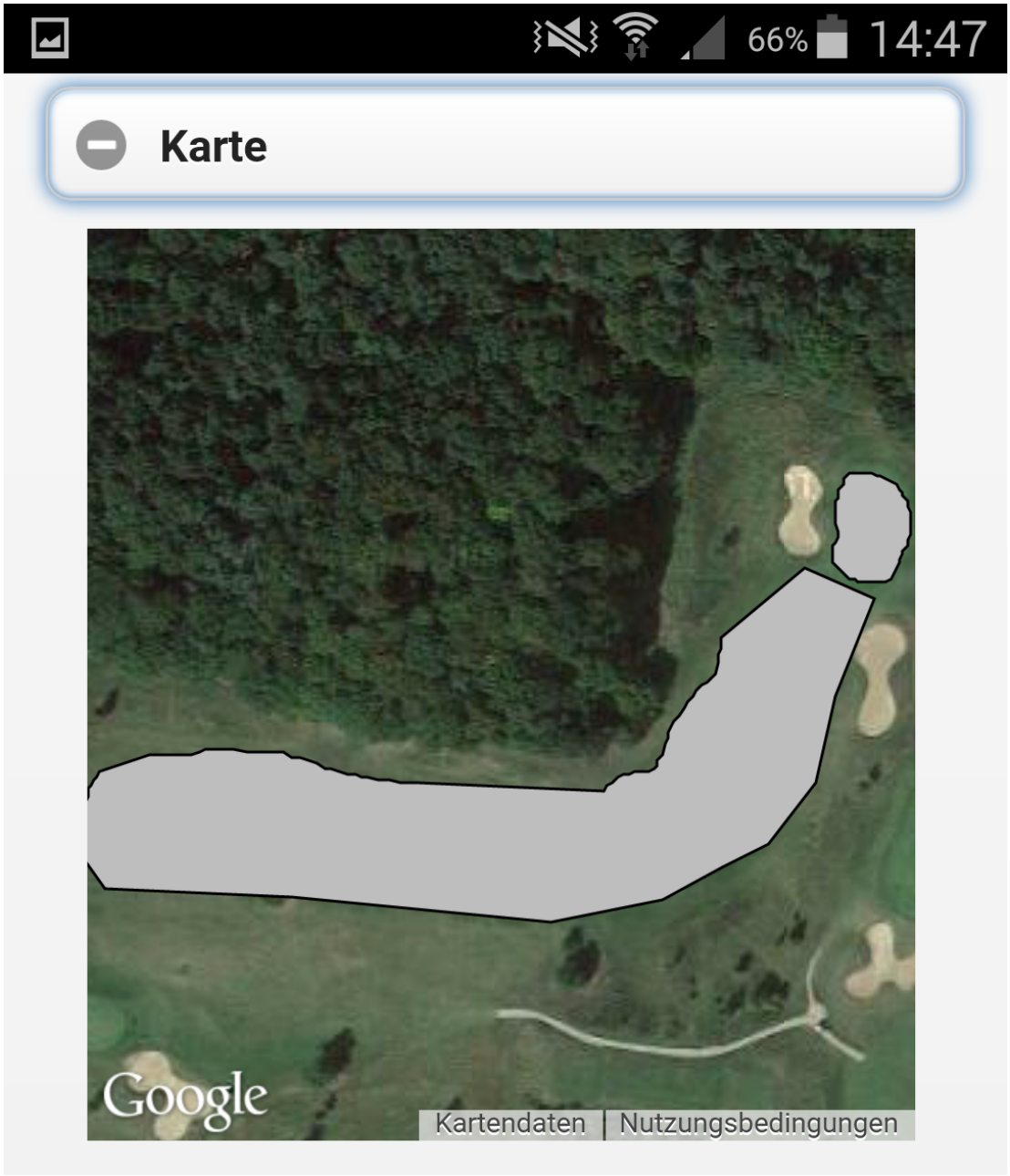

Abbildung A.9.: Anwendung der Mapbuilder-Themes auf das Szenario eines Golflochs

Literatur

[Bro+09] Brosius et al. »BI Reporting mit SQL Server 2008«. In: (2009).

[CCS93] E.F. Codd, S.B. Codd und C.T Salley. »Providing OLAP to User-Analysts: An IT Mandate«. In: (1993).

[Coc97] Rick Cock. Is a hybrid database in your future? 1997. URL: http://sunsite.uakom.sk/sunworldonline/swol-02-1997/swol-02-objects.html.

[Dar03] Priya Darshane. Oracle® HTTP Server. 2003.

[Goo14] Google-Maps-Group. Keyhole Markup Language. 2014. URL: https://developers.google.com/kml/.

[ID00] National Imagery und Mapping Agency: Department of Defense. World Geodetic System 1984. 2000.

[JA05] Bill Jelen und Mike Alexander. Pivot Table Data Crunching. Paperback, 2005.

[Mur11] Chuck Murray. »Oracle Spatial Users Guide and Reference«. In: (2011).

[Ora08] Oracle-OLAP-Group. Oracle OLAP Users Guide and Reference. Oracle. 2008.

[Pow04] Daniel Power. Decision Support Systems: Frequently Asked Questions. iUniverse, Inc., 2004. ISBN: 0595339719.

[PU03] Günther Pernul und Rainer Unland. Datenbanken im Unternehmen. Oldenbourg, 2003. ISBN: 3486272101.

[Ree14] Carl Reed. Memeber Orientation. 2014. URL: https://portal.opengeospatial.org/files/61449.

[Sau02] Hermann Sauer. Relationale Datenbanken . Theorie und Praxis. Addisson-Wesley, 2002. ISBN: 3827320607.

[Sch08] Scholl. Data Warehousing und OLAP. 2008. URL: http://www.inf.uni-konstanz.de/fileadmin/dbis/DWOLAP/DWOLAP-komplett-4.pdf.

[SO12] Declan G. DePaor Steven J. Whitmeyer John E. Bailey und Tina Ornduff. Google Earth and Virtual Visulizations in Geoscience Education and Research. Geological Society of Amer, 2012. ISBN: 0813724929.

[Tot12] Andreas Totok. Modellierung von OLAP- und Data-Warehouse-Systemen. Deutscher Universitätsverlag, 2012. ISBN: 3824471108.

[Wei11] Alexander Weigl. »Geo-Informationen mit Oracle 11g«. In: (2011).