Begrüßung

Inhaltsverzeichnis

Datenbanken

Eine Datenbank ist ein elektronisches Verwaltungssystem, das besonders mit großen Datenmengen effizient, widerspruchsfrei, dauerhaft umgehen muss und logische Zusammenhänge digital abbilden kann.

Wie ist ein Datenbanksystem aufgebaut?

Ein Datenbanksystem besteht aus zwei wichtigen Komponenten, die jeder kennen sollte:

- Das Datenbankmanagementsystem ist für die Verwaltung der Datenbank(-en) zuständig.
- Die Datenbank speichert die Informationen in Form von einzelnen Datensätzen ab.

  - Ein Datenbanksystem unterstützt die computergestützte Datenverarbeitung von Informationen, die durch eine Datenbankapplikation erzeugt und verarbeitet werden.

  - Das Datenbanksystem strukturiert und speichert die Informationen in einer Datenbank ab.

Datenbank Beispiele – Wo werden Datenbanken verwendet?

Wir haben zwei Beispiele ausgesucht, worunter sich jeder von uns gut etwas vorstellen kann:

ERP-Systeme

- bilden die Schnittstelle **zwischen unterschiedlichen Datenbanksystemen**. Die Datenbanken unterschiedlicher Systeme müssen miteinander kommunizieren können. Mittels ERP-Systemen wird ein Datenaustausch realisiert. Die unterschiedlichen Recourcen unterhalten sich dann vollkommen autark im Hintergrund, ohne dass ein Benutzer direkt eingreifen muss.

Banken und Versicherungen benötigen Datenbanken

- um ihre Kunden zu verwalten. Sie nutzen dabei ebenfalls CRM-Systeme, aber auch häufig CMS-Systeme für den Bereich des Online-Bankings. Gas- und Stromlieferanten nutzen heute ihre CMS-Systeme auch gerne für die eigene Eingabe der Zählerstände der Gas- und Stromabnehmer.

CRM….Customer-Relationship-Mangement (Kundenbeziehungen)

Die nebenstehende Grafik zeigt, dass ca. 87% aller Programme, direkt oder indirekt mit Datenbanken arbeiten. Man kommt also heutzutage kaum noch drum herum, mit Datenbanken zu arbeiten.

Object-Relational-Mapping

In der Datenbankwelt sind relationale Datenbanken vorherrschend, in der Programmierwelt sind es Objekte. Zwischen den beiden Welten gibt es erhebliche Unterschiede. Kern des objektorientierten Programmierens ist die Arbeit mit Objekten als Instanzen von Klassen im Hauptspeicher. Die meisten Anwendungen haben dabei auch die Anforderung, in Objekten gespeicherte Daten dauerhaft zu speichern, insbesondere in Datenbanken.
Grundsätzlich existieren objektorientierte Datenbanken, die direkt in der Lage sind, Objekte zu speichern. Aber objektorientierte Datenbanken haben bisher nur eine sehr geringe Verbreitung. Der vorherrschende Typus von Datenbanken sind relationale Datenbanken, die jedoch Datenstrukturen anders abbilden als Objektmodelle.
Um die Handhabung von relationalen Datenbanken in objektorientierten Systemen natürlicher zu gestalten, setzt die Software-Industrie seit Jahren auf O/R-Mapper. O steht dabei für objektorientiert und R für relational. Diese Werkzeuge bilden demnach Konzepte aus der objektorientierten Welt, wie Klassen, Attribute oder Beziehungen zwischen Klassen auf entsprechende Konstrukte der relationalen Welt, wie zum Beispiel Tabellen, Spalten und Fremdschlüssel, ab.
Der Entwickler kann somit in der objektorientierten Welt verbleiben und den O/R-Mapper anweisen, bestimmte Objekte, welche in Form von Datensätzen in den Tabellen der relationalen Datenbank vorliegen, zu laden bzw. zu speichern. Wenig interessante und fehleranfällige Aufgaben, wie das manuelle Erstellen von INSERT-, UPDATE- oder DELETE-Anweisungen übernimmt hierbei auch der O/R-Mapper, was zu einer weiteren Entlastung des Entwicklers führt.

Um diesen Strukturbruch zu überwinden, existieren eine Reihe von Lösungen. Dazu gehören objekt-relationale Datenbanken und Frameworks für objekt-orientierte Programmiersprachen, die das Mapping zwischen den Objekten und ihren korrespondierenden Datenstrukturen in relationalen Datenbanken sehr generisch lösen. Zu den Strukturbrüchen, die mit einem Framework gelöst werden, gehören u.a.:

- Die Abbildung von Beziehungen/ Kardinalitäten.
- Strategien zur Generierung von eindeutigen Schlüsseln/ Identifier.
- Umsetzung des objekt-orientierten Konzepts der Vererbung.
- Eine einheitliche Handhabung des Datenbankzugriffs im Hinblick auf den konkurrierenden Zugriff auf diese Ressource.

Um den Widerspruch aufzulösen oder zumindest zu mildern, wurden verschiedene Lösungen vorgeschlagen, beispielsweise objektorientierte Datenbanken oder die Erweiterung von Programmiersprachen um relationale Konzepte (z. B. Embedded SQL). Die direkte objektrelationale Abbildung von Objekten auf Relationen hat den Vorteil, dass einerseits die Programmiersprache selbst nicht erweitert werden muss und andererseits relationale Datenbanken als etablierte Technik in allen Umgebungen als ausgereifte Software verfügbar sind. Nachteil dieses dem OOP-Paradigma entgegenkommenden Ansatzes ist das Nicht-Ausnutzen der Stärken und Fähigkeiten einer relationalen Datenbank, was sich in nicht optimaler Leistung niederschlägt.

Grundlegende Technik

- Im einfachsten Fall werden Klassen auf Tabellen abgebildet, jedes Objekt entspricht einer Tabellenzeile und für jedes Attribut wird eine Tabellenspalte reserviert. Die Identität eines Objekts entspricht dem Primärschlüssel der Tabelle. Hat ein Objekt eine Referenz auf ein anderes Objekt, so kann diese mit einer Fremdschlüssel-Primärschlüssel-Beziehung in der Datenbank dargestellt werden.
- Der Begriff Shadow Information („Schatteninformation“) bezeichnet zusätzliche Daten, die ein Objekt benötigt, um persistent abgelegt zu werden.[2] Dazu gehören Primärschlüssel – speziell wenn es sich um Surrogatschlüssel ohne fachliche Bedeutung handelt – sowie Hilfsdaten für die Zugriffssteuerung, beispielsweise Zeitstempel.

Vorteile bei der Verwendung von ORM-Konzepten

- Unabhängig von Typ der Datenbank, also egal ob SQLite, MySQL, MariaDB oder sonst was verwendet wird
- Es müssen keine SQL-Statements mehr geschrieben werden
- Eignet sich gut um Software schnell und unkompliziert auf verschiedenen Endgeräten zu testen (zum Beispiel, wenn eine für Android entwickelte App auf ISO-Endgeräten implementiert werden soll)

Nachteile bei Einsatz von ORM-Systemen

- Es kann zu Problemen mit unterschiedlichen Versionen kommen:

  Java-Version, ORM-Version, Eclipse-Version, das alles muss zusammenstimmen
- Es kommt zu Leistungseinbüsen – Software wird träger; ORM eigenet sich deshalb nur bedingt für Leistungs- bzw. Zeitkritische Anwendungen
- Es wird mehr Speicher benötigt, auf Geräten mit geringem bzw. begrenztem Speicher ist ORM nur mit vorsichtig zu verwenden (Stichwort: Raspberry PI)
- Mit ORM sind teilweise nicht alle Funktionen einer Datenbank voll nutzbar (manchmal werden Datenbanken wegen spezieller Eigenschaften überhaupt erst verwendet, diese können bei Verwendung von ORM wegfallen – dann ist ORM unter Umständen nicht sinnvoll)
- Der Umgang mit SQL-Statements wird schnell verlernt, da diese Aufgabe vom ORM-System übernommen wird

Wo ist es sinnvoll mit ORM zu arbeiten?

ORM kann und soll immer dann verwendet werden, wenn eine Software mit unterschiedlichen Datenbanktypen zurechtkommen muss. Also zum Beispiel eine Software wird für PC entwickelt und im Anschluss soll noch eine App mit unserer Software kommunizieren können. Am PC verwenden wird MySQL am Smartphone SQLite. In so einem Fall eignet sich ORM um diese Barriere umgehen zu können. Der Großteil der Klasse programmiert ja in JAVA, und dort ist man es gewohnt, dass die entwickelte Software auf allen Betriebssystemen ausgeführt werden kann. Java in Verbindung mit ORM ist deshalb eine gute Kombination um maximal Plattformunabhängig zu sein.

Code-Beispiel

Zusammenfassung

Datenbanken ist als Programmierer unverzichtbar. Knapp 90% aller Software arbeitet mit Datenbanken.

Problem: Die zwei grundlegend verschiedenen Konzepte. Höhere Programmiersprachen arbeiten mit Objekten und relationalen Datenbanken mit Zeilen und Spalten in Form von Tabellen.

Hier kommt ORM ins Spiel. Um diese Unterschiede überbrücken zu können, dient ORM als „Zwischenschicht“ zwischen JAVA, Python etc. und eben einer Datenbank.

ORM-Konzepte kommen immer dann zum Einsatz, wenn ein Programmmit mehreren unterschiedlichen Datenbanktypen arbeiten muss oder soll.

```
package
com.example.helloandroid;

                        import java.text.SimpleDateFormat;
                        import java.util.Date;
                        import java.util.Locale;


                        import com.j256.ormlite.field.DatabaseField;


                        /**
                         * A simple demonstration object we are creating and
                        persisting to the database.
                         */
                        public class SimpleData {


                                // id is generated by the database and set on the
                        object automagically
                                @DatabaseField(generatedId = true)
                                int id;
                                @DatabaseField(index = true)
                                String string;
                                @DatabaseField
                                long millis;
                                @DatabaseField
                                Date date;
                                @DatabaseField
                                boolean even;


                                SimpleData() {
                                        // needed by ormlite
                                }


                                public SimpleData(long millis) {
                                        this.date = new Date(millis);
                                        this.string = (millis % 1000) + "ms";
                                        this.millis = millis;
                                        this.even = ((millis % 2) == 0);
                                }


                                @Override
```

```
        public String toString() {
                StringBuilder sb = new StringBuilder();
                sb.append("id=").append(id);
                sb.append(", ").append("str=").append(string);
                sb.append(", ").append("ms=").append(millis);
                SimpleDateFormat dateFormatter = new
SimpleDateFormat("MM/dd HH:mm:ss", Locale.US);
                sb.append(",
").append("date=").append(dateFormatter.format(date));
                sb.append(", ").append("even=").append(even);
                return sb.toString();
        }
}
```