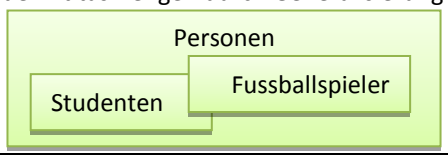
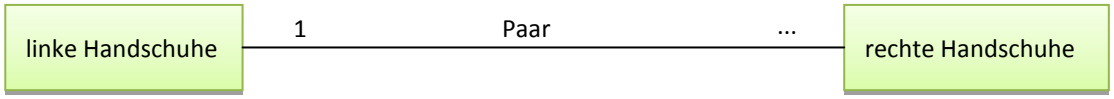
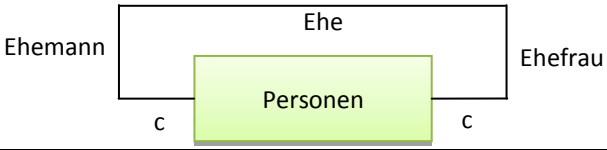
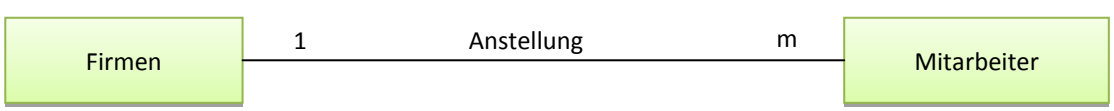
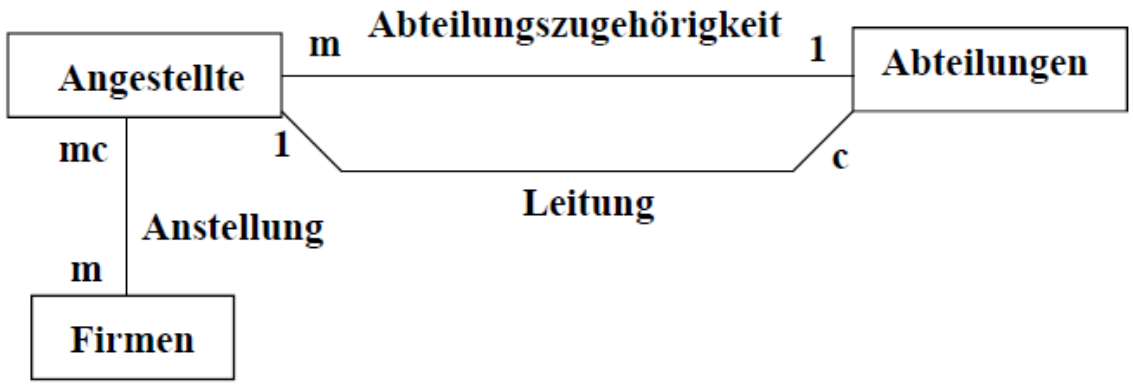
SOFTWARE ARCHITEKTUR

Lektion 1 – ERM (Entity-Relationship-Modell)

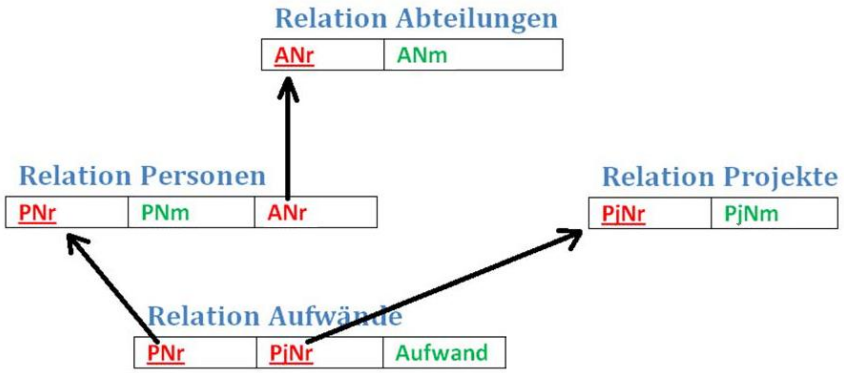
Entität (Entity)	Individuelles Exemplar z.B. Peter Muster
Entitätsmenge (Entity set)	Gruppierung von Entitäten mit identischen Merkmalen aber unterschiedlichen Merkmalswerten z.B. Personen, Studenten Überlappende Entitätsmengen durch Generalisierung vermeiden: 
Beziehungen (Relationships)	Beziehungen sind gerichtet Eine Beziehung hat immer zwei Assoziationen  z.B. Ein linker Handschuh hat ... rechten Handschuh
Assoziation (Association)	Eine quantifizierbare Beziehung
Rollen (Roles)	Beziehungen in der gleichen Entitätsmenge 
ER-Diagramm	 Eine Firma hat mehrere Mitarbeiter angestellt. Ein Mitarbeiter ist bei einer Firma angestellt.
Vorgehen	1. Entitätsmengen identifizieren 2. Beziehungen einsetzen 3. Assoziationstypen beschriften 4. Rolle der Beziehung bestimmen

Entity-Relationship-Model				**vs**	**UML**	
Entität					Objekt	
Entitätsmenge (in Mehrzahl)					Klasse ohne Attributen und Operationen (in Einzahl)	
Beziehungen ▪ Verb					Assoziationen ▪ Nomen	
Assoziationstypen	m	multiple	1,2, …		$x..y$	Bereich angeben
	mc	can multiple	0,1,2, …		$*$	mehrere
	1	one	1			
	c	can	0,1			



Lektion 2 – RM (Relationenmodell)

Relation (relation)	Tabelle ohne	▪ Duplikate ▪ Reihenfolgeerhaltung ▪ Sortierung
Tupel (tuple)	Ein Element einer Relation • eindeutig	
Attribut (attribute)	Eine Eigenschaft einer Entitätsmenge, mit Datentyp	
	global	mindestens einmal im Identifikationsschlüssel
	lokal	nur in einer Relation und nicht im Identifikationsschlüssel
Identifikationsschlüssel	Eindeutige Bezeichnung, unterstrichen	
Fremdschlüssel (foreign key)	Abhängigkeit zwischen Relationen	
Mutationsanomalie	Bei Änderung nur ein Datensatz ändern Tritt nur auf bei Daten mit Redundanzen	
Umwandlung ERM -> RM	1. Entitätsmengen -> Relationen 2. Beziehungen -> Attribute und Hilfsrelationen	



Lektion 2 – Normalisierungsprozess (unstrukturierte Tabelle -> RM)

Identifikationsschlüssel

- ein oder minimale Attributskombination
- jedes Tupel ist eindeutig
- Wert ändert sich nicht

Abhängigkeiten

Funktionale Abhängigkeit $R.A \rightarrow R.B$

B ist funktional abhängig von A, wenn zu einem Wert A nur ein Wert von B möglich ist.

A	B
101	Hans
102	Peter

A	B
101	Hans
102	Hans

A	B
101	Hans
101	Peter

Volle Abhängigkeit $R.A \Rightarrow R.B$

B ist voll von A abhängig, wenn $R.A \rightarrow R.B$ aber nicht bereits von Teilen von A

A	B
101	Hans
102	Peter

A		B
101	Hans	X
101	Rolf	Y

A		B
101	Hans	X
102	Rolf	Y

Transitive Abhängigkeit $R.A \rightarrow\rightarrow R.C$

abhängig über Umweg

$$R.A \rightarrow R.B \rightarrow R.C$$

Vorgehen

0. Normalform
- Ausgangslage

1. Normalform
- keine Aufzählungen (A,B,C) in einem Attributswert → Tupel wiederholen
- keine Nullwerte ( ) → Relation aufspalten

2. Normalform
- In 1. Normalform
- jedes nicht zum ID gehörige Attribut voll von diesem abhängig ist

3. Normalform
- In 2. Normalform
- kein Attribut transitiv vom ID abhängt (also nicht abhängig von irgendeinem Attribut)

Globale Normalform
- Globale/Lokale Attribute identifizieren
- Generalisierung (gleiche Attribute verschiedener Relationen → in eine Relation überführen)
- Fremdschlüssel

Lektion 3 – Transformation (ERM → RM)

Beziehungen	**Hierarchisch** 1 muss vorkommen	1-1, 1-c, 1-n, 1-mc	führt zu Fremdschlüsseln
	Konditionell c muss vorkommen	c-c, c-m, c-mc	eine zusätzliche Relation (oder Nullwerte) Männer c:c Frauen -> Männer 1:c Ehen c:1 Frauen
	Netzwerkartig m muss vorkommen	m-m, m-mc	eine zusätzliche Relation Personen mc:m Projekte -> Personen 1:m Tätigkeit mc:1 Projekte
Rekursion	• **Rekursive Beziehungen müssen aufgelöst werden.** • **Ein Fremdschlüssel darf nicht auf einem globalen Attribut in der gleichen Relation basieren.** Personen 1:Vorgesetzter:mc Personen -> Personen 1:unterstellt:c Hierarchie Personen 1:Chefsein:mc Hierarchie		

Lektion 4 + 5 – Transaktionen

ACID Eigenschaften

Atomicity (Atomität)	Transaktion wird als Einheit ausgeführt (Alles oder nichts)
Consistency (Konsistenz)	Referentielle Integrität (=nur gültige Daten/Schlüssel in DB) bleibt erhalten (am Ende einer erfolgreichen Transaktion)
Isolation	Modifikationen sind erst beim Abschluss einer Transaktion nach aussen sichtbar
Durability (Dauerhaftigkeit)	Modifikationen einer erfolgreich abgeschlossenen Transaktion sind dauerhaft vorhanden (z.B. auch nach Systemabsturz) nach Commit und Bestätigung der Datenbank

Nebenläufigkeit (Concurrency)

Probleme die auftreten, wenn Transaktionen gleichzeitig ablaufen:

Lost Updates

Zwei Transaktionen lesen und aktualisieren unabhängig voneinander

Transaktion A	Transaktion B
retrieve 1	
	retrieve 1
update 1	
	update 1

retrieve = abfragen
update = aktualisieren
commit = bestätigen
rollback = zurücksetzen

Uncommitted Dependency (Dirty Read)

Zweite Transaktion liest ein Tupel, das von der ersten geändert aber noch nicht commited wurde.

Transaktion A	Transaktion B
update 1	
	retrieve 1
rollback 1	

Inconsistent Analysis (Non Repeatable Read)

Erste Transaktion bekommt unterschiedliche Resultate, wenn sie die Tupel mehrfach ausliest.

Transaktion A	Transaktion B
retrieve 1	
retrieve 2	
	retrieve 3
	update 3
	retrieve 1
	update 1
	commit
retrieve 3	

Phantome

Erste Transaktion liest mehrere Tupel. Beim zweiten Mal sind neue hinzugekommen.

Transaktion A	Transaktion B
retrieve 1, 2	
	insert 3
	commit
retrieve 1,2,3	

Abwicklung von Transaktionen

- Sequentiell -> korrekte Ergebnisse (ACID)
- Parallel -> erfordert Schutzmassnahmen

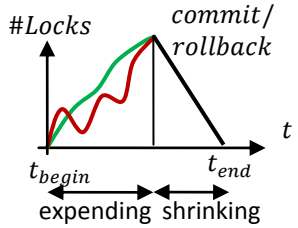
Serialisierbarkeit (=sequentiell)

Gegeben sind mehrere Transaktionen: T1, T2, T3

- Jede Transaktion sei konsistenzhaltend
- Serielle Abarbeitung aller Transaktionen ist somit auch korrekt
  T1, T2, T3 oder T3, T1, T2 etc...

Eine verschränkte Ausführung ist nur dann korrekt, wenn sie äquivalent zu einer seriellen Abarbeitung ist!

two-phase locking Protocol



Sperren / Locks

Um die Nebenläufigkeit zu ermöglichen werden Sperren auf Datensätze sowie Bereiche eingeführt

- Shared Locks (Read Locks) S
- Exclusive Locks (Write Locks) X

Ergebnis
Alle Concurrency-Probleme lassen sich durch Locks und das Two-Phase Locking Protokoll lösen.
-> ACID Eigenschaften erfüllt.

Aber es resultiert:

- Blockierungen -> Timeouts
- Deadlocks -> Deadlock victim (Opfer)

T_A hält → / T_B will ↓	X	S	
X	N	N	J
S	N	J	J
	J	J	J

Lost Update mit Locks – (Deadlock)

Transaktion A	Transaktion B
retrieve 1	
request S on 1	
S on 1 acquired	
	retrieve 1
	request S on 1
	S on 1 acquired
update 1	
request X on 1	
wait---	update 1
wait---	request X on 1
wait---	wait---

Isolationsstufen unter SQL

- Steuert den Grad der zulässigen Interferenz zwischen Transaktionen
- Ziel der Isolationsstufen ist die **Steigerung des Durchsatzes** (Anzahl T / s) sowie **Steigerung der Performance**.

Read uncomitted

- T kann Änderungen von anderen Ts. lesen, sobald sie gemacht wurden. T weiss allerdings nicht, ob Änderungen Bestand haben.
- Keine S-Locks, Lange X-Locks

Read comitted

- T kann Änderungen von anderen Ts nur dann lesen, wenn diese Ts beendet wurden
- T wird blockiert, wenn andere Ts Daten ändern (insert, update, delete)
- Kurzfristige S-Locks, Lange X-Locks

Repeatable Read

- T kann Tupel mehrfach lesen, evtl. kommen weitere Tupel hinzu
- Alle anderen Ts, die das Nachlesen **verhindern**, werden blockiert, also wenn andere Ts **update**, **delete** ausführen, insert möglich!
- S-Locks nur auf bestehenden Tupeln!
- Selects werden i.A. nicht blockiert!

Serializable

- T kann Tupel mehrfach lesen und die Anzahl bleibt sich gleich
- Alle anderen Ts, die das Nachlesen **verhindern**, werden blockiert, also wenn andere Ts **update**, **delete**, **insert** ausführen
- S-Locks auf Tupeln, die ein Prädikat erfüllen
- selects werden i.A. nicht blockiert!

		Isolationsstufe			
		Serializable	**Repeatable Read**	**Read Committed**	**Read Uncommitted**
Concurrency Probleme	**Dirty Read Uncommitted Dependency**	verhindert	verhindert	verhindert	
	Nonrepeatable Read Inconsistent Analysis	verhindert	verhindert		
	Lost Updates				
	Phantoms	verhindert			

Andere Isolationsstufen

Snapshot-Isolation (Multiversion-Isolation)

- T liest nur Daten, die zum Transaktiosstartzeitpunkt t von T gültig waren (Snapshot zum Zeitpunkt t)
- Alle Schreiboperationen von T werden in den Snapshot integriert

Dadurch wird das **Lesen nie blockiert**.
Es entstehen jedoch Probleme beim Zusammenfügen von Snapshots.
Durch die "**First commiter wins**" Strategie müssen alle nachfolgenden Commits ein Merge (Zusammenführen) durchführen.

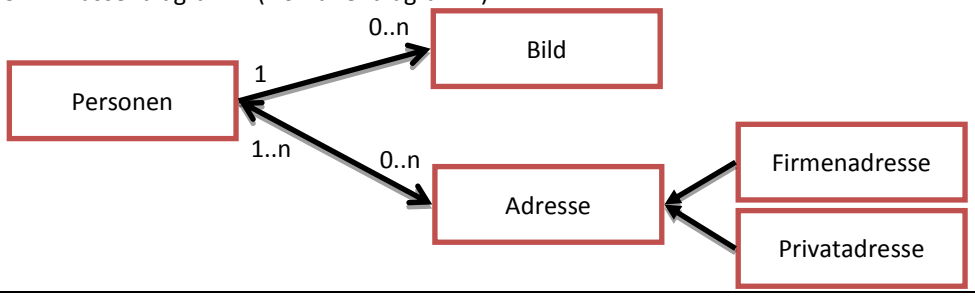
Regeln für Transaktionsverwendung

- Genaue Analyse für Wahl der richtigen Isolationsstufe unerlässlich
  - Read committed in den meisten Fällen ausreichend
- Transaktionen auf kurze Codesequenzen beschränken
- Keine Benutzerinteraktion innerhalb einer Transaktion
- Verhalten des konkreten DBMS austesten

Lektion 6 – Transformation eines Klassenmodells in ein Datenmodell

0. Ausgangslage

UML-Klassendiagramm (Domänendiagramm)



1. Auswahl der persistenten Klassen

persistent = dauerhaft abspeichern	→	Personen, Bild, Adresse, Firmen-, Privatadresse
transient = Verwendung während Laufzeit	→	- (z.B. Jobmanager)

2. Hinzufügen eines OID-Attributs

OID = persistenter Ersatz für Referenzen

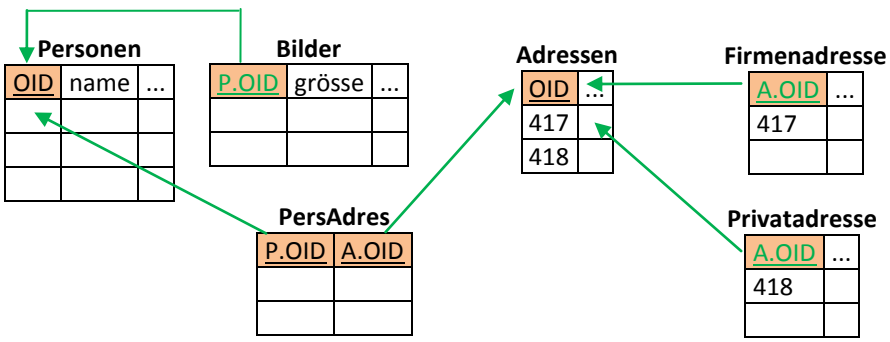
Eigenschaften des OID

- eindeutig
- werden nicht verändert

3. Überführung ins physikalische Datenmodell

persistente Klasse	→	Entität	→	Relation
persistentes Attribut	→	Attribut einer Entität	→	Spalte einer Relation
pro Klasse ein OID	→	Primärschlüssel		
1:n-Beziehung	→	Fremdschlüsseln		
m:n-Beziehungen	→	Beziehungsrelation (Spalten sind Primärschlüssel)		
Vererbung	→	1-c Beziehung		

4. Relationen Modell

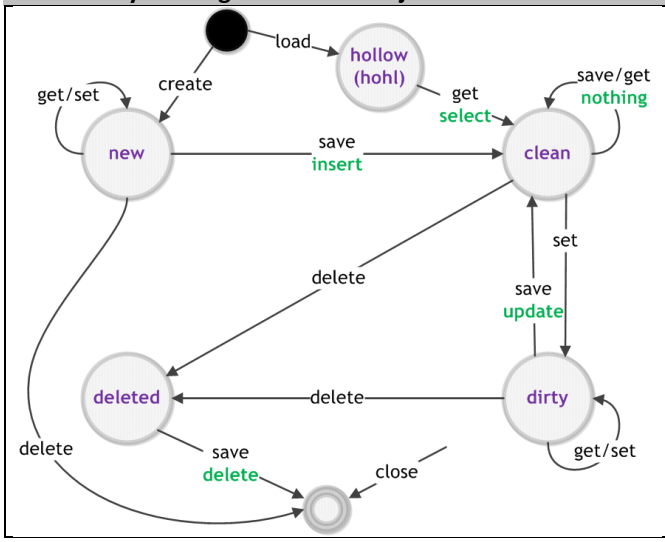


Verantwortlichkeiten der Persistenzschicht

- Objektidentität -> durch OID
- Laden von Objekten
- Speichern von Objekten
- Verwalten von Beziehungen
- Transaktionen

Dienste von JPA

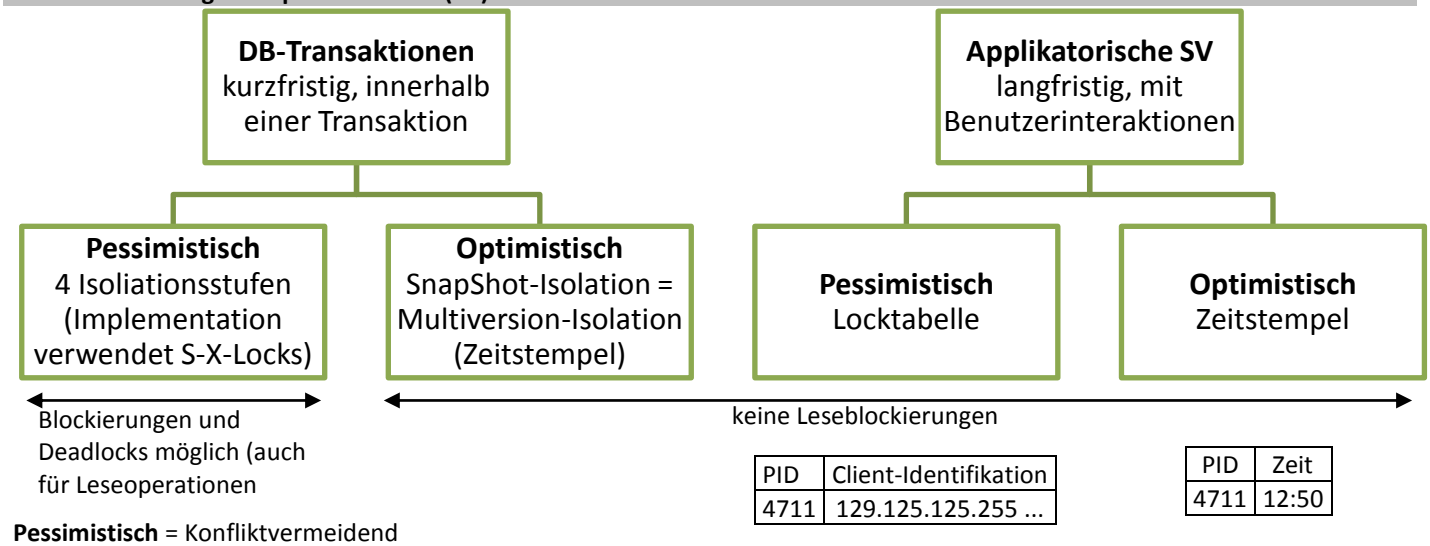
1. Lebenszyklusdiagramm eines Objekts



Aktionen
create
load (select)
save (insert/update/delete)
get/set
delete

Zustand	Client		DB	DB-Aktion
New	vorhanden		fehlt	**Insert**
Clean	vorhanden	=	vorhanden	**-**
Dirty	vorhanden	≠	vorhanden	**Update**
Deleted	fehlt		vorhanden	**Delete**
Hollow	nur OID		vorhanden	**Select**

2. Multiuser-Fähigkeit: Sperrverfahren (SV)



PID	Client-Identifikation
4711	129.125.125.255 ...

PID	Zeit
4711	12:50

Pessimistisch = Konfliktvermeidend

- Selten angewendet, da mergen schwierig ist

Optimistisch = Konflikterkennung und –behebung

1. Beim Ersten Lesen (load) Zeit merken.
2. Beim Speichern (update / delete) zuerst Zeitstempel lesen und mit ursprünglicher Zeit vergleichen.
   => Sind sie identisch liegt kein Konflikt vor. Es kann gespeichert werden. Sonst: Konflikt

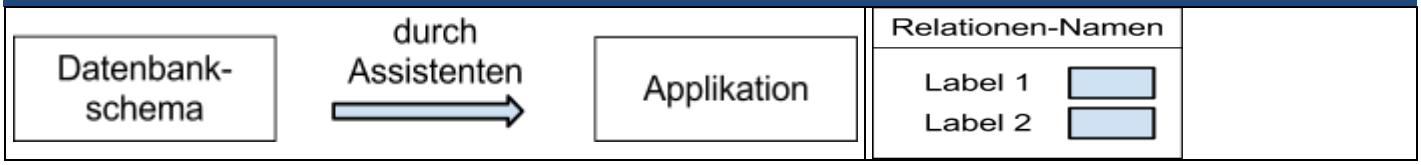
3. Performance-Optimierung

möglichst viel Last wegnehmen (=wenig SQL-Statements)

4. Query-Language

5. Relationenmodell erstellen aus Klassenmodell

Lektion 7 – Formulargenerator



Transformationsregeln

Relation -> Fenster

Attribut:Typ -> Label: Eingabefeld

$$R_1 * A -> R_2$$

Lektion 7 – Client-Server-Architektur

DIY-Verfahren (Do it yourself)

- Mapping zwischen Oo – RDBMS
- Objektlebenszyklus (Zustandsdiagramm)
- Applikatorische Sperrverfahren (pess./optim.)
- Performanceoptimierungen (Caching)

dazu gibt es Entwurfsmuster

Framework verwenden

Java Persistance API

- Teil von Java EE (Enterprise Edition)

Aufgaben

- alle von oben
- Query Language (JQL, LINQ)
- Caching

Implementierungen

- Hibernate
- EclipseLink(Referenzimplementierung)
- OpenJPA

Spezifikation eines OO-RDBMS-Mappers(Teil von JEE 6.0; auch nutzbar unter JSE)

// TODO: Wie funktioniert das?

Mapping = Zuordnung von Objekten zu Datenbankeinträgen

DB-Provider unabhängig

Aufgaben

- Mapping (Impedance Missmatch)
  - mit ***Annotationen*** oder ***XML-Konfigurationsdateien***
  - "Convention over Configuration"
- Verwaltung des Objektlebenszyklus
- Concurrency Control / Transaktion
- Abfragesprache JPQL (;;)
  - auf der OO-Seite!
  - ähnlich wie SQL
  - wird zur Laufzeit geprüft
  - Criteria API wird dagegen Statisch geprüft

Objektpersistenz	Das Objekt wird dauerhaft gespeichert
Mapping	Java-Objekt in Relationen umwandeln z.B. Dictionary, Hashmap

EntityManager	Fassade zu meinem Framework:
	`EntityManager em = ...`
	`Kunde kunde = new Kunde ("Max", " Mustermann ");`
	`em.getTransaction().begin();`
	`em.persist(kunde);`
	`...`
	`kunde = em.find(Kunde.class, <primaerschluessel>);`
	`em.getTransaction().commit();`

Voraussetzungen	
@Entity	Die Annotation @Entity macht aus der gewohnlichen, nicht persistenten Java-Klasse ein persistentes Entity.
@Id	Die Instanzvariable id ist mit @Id annotiert und wird damit von JPA als Primarschlusselvariable verwendet
@GeneratedValue	Wie wird dieses Objekt erzeugt? `@GeneratedValue ( strategy = GenerationType . AUTO )`
@Temporal	da Datentypen nicht harmonieren
@Column	Property-Definition verändern (z.B. Name, Länge, ...)

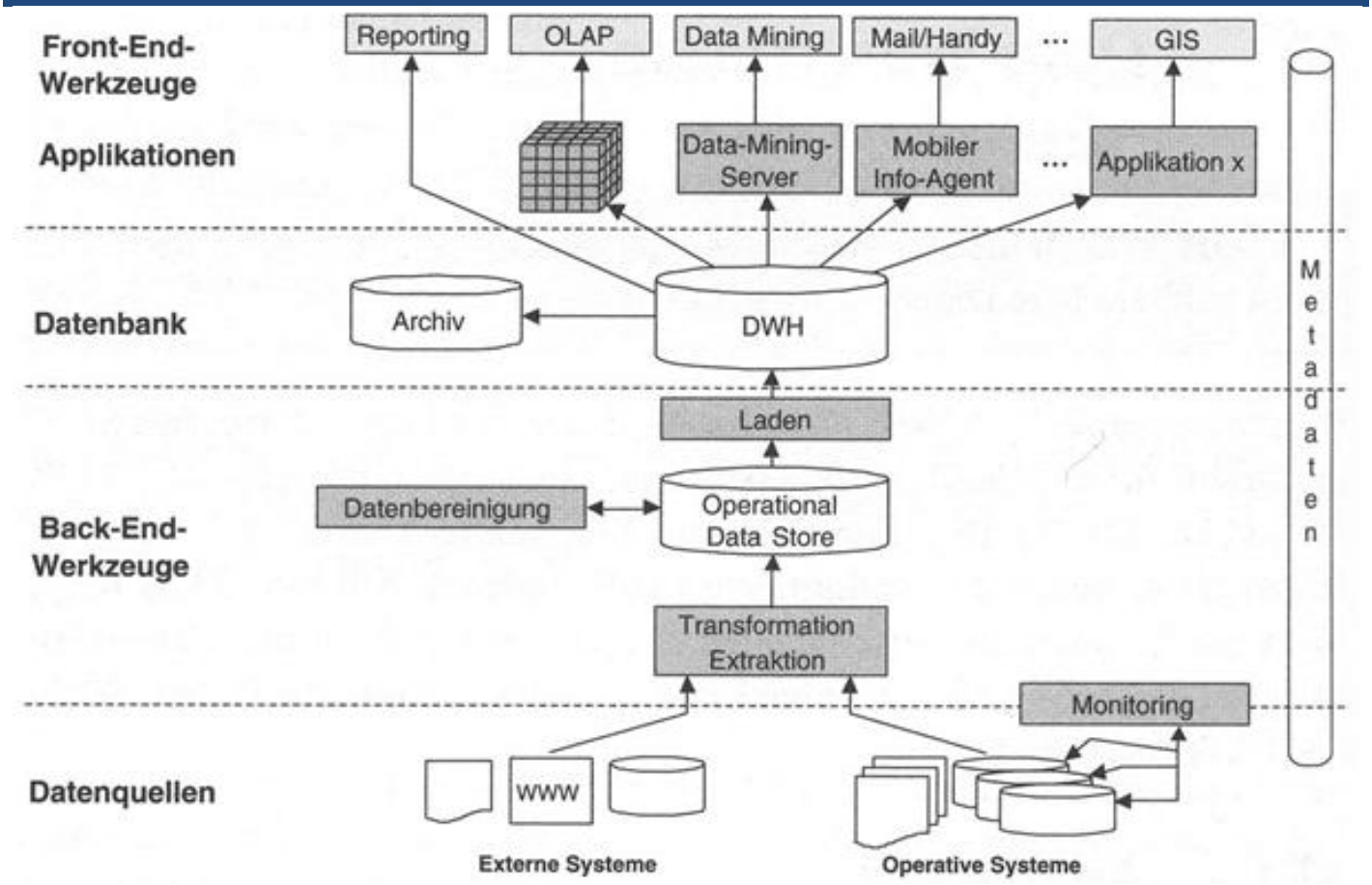
Lektion 8 – LINQ (Language Integrated Query)

Allgemein	Sprachenunabhängige Datenbankzugriffssprache
	LINQ versucht Daten aus unterschiedlichen Quellen mit ihren unterschiedlichen Zugriffmethoden zu vereinheitlichen. Die Syntax ist ähnlich zu SQL, jedoch nutzt LINQ die vorhanden Beziehungen.
	gehört zum .NET Framework
LINQ-Anbieter	• LINQ to Objects zum Zugriff auf Objektlisten und -Hierarchien im Arbeitsspeicher • LINQ to SQL zur Abfrage und Bearbeitung von Daten in MS-SQL-Datenbanken • LINQ to Entities zur Abfrage und Bearbeitung von Daten im relationalen Modell von ADO.NET; • LINQ to XML zum Zugriff auf XML-Inhalte • LINQ to DataSet zum Zugriff auf ADO.NET-Datensammlungen und -Tabellen • LINQ to SharePoint zum Zugriff auf SharePoint-Daten.
Muster einer Abfrage	$\underbrace{var\ query}_{lokale\ Variable} = Produkt.Where\underbrace{(p => p.Preis < 20)}_{Lambda-Ausdruck}.Select(p => new\ \underbrace{\{p.Name, p.Preis\}}_{Objekt-Initialisierer}$

Lektion 9 – JAVA Enterpreise Edition (JAVA EE)

Allgemein	Sammlung von vielen Technologien; eine Erweiterung von JAVA für Verteilte Anwendungen und Multi-Tier Architekturen
Technologien	JPA (Java Persistence API) JTA (Java Transaction API) JMS (Java Message Service) EJB (Enterprise Java Beans) – Komponenten die Funktionalität und Daten kapseln Servlets – Java-Klassen, deren Instanzen innerhalb eines Webservers Anfragen von Clients beantworten. JSF (JavaServer Faces) JAX-RS, JAX-WS
Konkurrenz	.NET Plattform von Microsoft

Lektion 10 – Data Warehousing



OLAP	Online Analytical Processing	
Bestandteile	Fakten	Ein Eintrag im Würfel
	Dimensionen	Achsen im Würfel
	Hierarchien	Hierarchie in den Achsen
Operationen	Drilling	Wechsel zwischen Hierarchie-Ebenen
	Roll up	Wechsel zu gröberen Hierarchie-Ebene
	Drill down	Wechsel zu feineren Hierarchie-Ebene
	Slice	Dimensionalität verringern
	Dice	Datenvolumen reduzieren
	Rotieren	Ansicht ändern
Typen von Speicherungen	ROLAP	Relationale Speicherung
	MOLAP	Mulitdimensionale Speicherung
	HOLAP	Hybride Speicherung
MDX	multidimensionale Datenbanksprache ähnlich zu SQL	

Lektion 11 – Data Mining

Allgemein	Computerbasierte Lerntechniken anwenden Standardisierter Ablauf
oder auch	BigData Analysis, Knowledge Discovery, Business Intelligence / Analytics
Vorgehen nach CRISP-DM	1. Zielidentifikation 2. Zieldatenmenge bestimmen 3. Datenvorverarbeitung 4. Datentransformation 5. Data Mining 6. Interpretation und Evaluation 7. Ergebnisnutzung

Lektion 12 – GIS

GIS	Geografische Informationssysteme
	Informationssystem zur Erfassung, Bearbeitung, Analyse und Präsentation von räumlichen Daten