DHBW Stuttgart

Datenbanken I

Kapitel 8 – Transaktion und Mehrbenutzersysteme

Modul: T2INF2004

Hinweis

DHBW
Duale Hochschule
Baden-Württemberg

Nutzungshinweis:

**Diese Unterlagen dürfen ausschließlich von Mitgliedern
(das sind Studierende, Bedienstete)
der Dualen Hochschule Baden-Württemberg Stuttgart eingesetzt werden.
Eine Weitergabe an andere Personen oder Institutionen ist untersagt.**

Inhalte Vorlesung

1. Grundlagen und Begriffsdefinitionen
2. Die Relation und ihre Sprache
3. Der konzeptionelle Datenbankentwurf (ER-Modell)
4. Der relationale Entwurf (logischer Entwurf)
5. Normalisierung
6. Einführung zum Datenbankentwurf

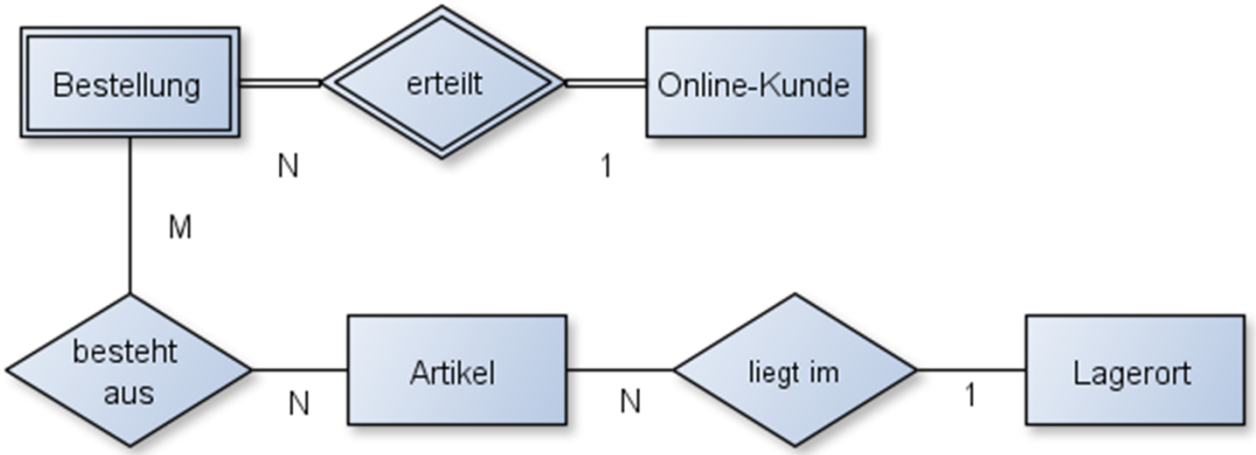
7.1 Die Sprache SQL (Teile DDL und DML)

7.2 Die Sprache SQL (Teile DQL und DCL)

8. **Transaktion und Mehrbenutzersysteme**
9. **Praxisprojekt**

Erweiterung ERM des Softwarehauses

- Das Softwarehaus will einen Onlineshop für diverse Software eröffnen.
- ERM und Datenbank müssen erweitert werden.



Anlegen einer neuen Bestellung

- Bei Änderungen in der Datenbank sind meistens mehrere Datensätze betroffen.
- Bestellt beispielsweise ein Kunde Waren aus dem Onlineshop, dann wird eine neue Bestellung mit verschiedenen Artikeln erstellt.

Frage: Welche Tabellen sind betroffen und was muss getan werden wenn eine neue Bestellung angelegt wird?

1. Es wird ein neuer Datensatz in die Tabelle ..... eingefügt.
2. Ein neuer Datensatz wird in die Zwischentabelle ..... einfügen mit den Attributen ..... und `menge`.
3. Dazu muss zuerst die neue ..... der Bestellung ermittelt werden.
4. Die ..... wird beim Einfügen hochgezählt.

Anlegen einer neuen Bestellung

Aufgabe: Überlegen Sie kurz (mit Nachbarn) welche SQL-Statements wir benötigen und wie diese aussehen werden (grob).

- Die erste Anweisung wird nun aus irgendeinem Grund nicht ausgeführt, weil wir z.B. eine falsche Syntax geschrieben haben.
- Wir erhalten dadurch eine Fehlermeldung.

Frage. Was passiert mit der zweiten Anweisung?

```
psql:insert_bestellung.sql:13: FEHLER:  currval von Sequenz »bestellung_bestellung_id_seq« ist in dieser Sitzung noch nicht definiert
onlineshop=# \i insert_bestellung.sql
psql:insert_bestellung.sql:5: FEHLER:  Syntaxfehler bei »ï»¿INSERT«
ZEILE 1: ï»¿INSERT INT bestellung
         ^
psql:insert_bestellung.sql:11: FEHLER:  currval von Sequenz »bestellung_bestellung_id_seq« ist in dieser Sitzung noch nicht definiert
onlineshop=#
```

Das erste Problem

- Die Bestellung ist erstellt und die beiden Artikel wurden bestellt.
- Als nächstes müssen beide Artikel versendet werden.
- Frage. Was ist auf der DB zu tun?

1. In der Tabelle ……. muss der …… auf „versendet“ gesetzt werden.
2. Der Lagerbestand (`menge_aktuell`) in der Tabelle `Lager` muss um die jeweilige Bestellmenge verringert werden.

```
UPDATE Lager
SET menge_aktuell = menge_aktuell - 1
WHERE artikel_id = 2
;
UPDATE bestellung SET status = 'versendet'
WHERE bestellung_id = 1;
```

Das erste Problem

Frage: Was passiert nun wenn eine der beiden Anweisungen aus irgendeinem Grund nicht ausgeführt wird?

1. Lagerbestand wird verringert, aber die Bestellung hat immer noch den Status „offen“. Wird also vermutlich nochmals versendet.
2. oder:
   Die Artikel werden aus dem Lagerbestand entnommen, aber der Lagerbestand wird nicht verringert.

➢ Die Datenbank befindet sich in einem inkonsistenten Zustand.

Forderung: Beide Anweisungen müssen in jedem Fall gesamt ausgeführt werden. Kann eine der Anweisungen nicht ausgeführt werden, darf die andere auch nicht ausgeführt werden.

➔ Die Bündelung mehrerer Anweisungen zu einer logischen Einheit nennt man eine **Transaktion.**

Die Transaktion ACID

- **Eine Transaktion überführt eine Datenbank von einem konsistenten Zustand in einen neuen konsistenten Zustand.**
- Eine Transaktion muss bestimmten Anforderungen genügen, welche unter dem Kürzel ACID zusammengefasst werden.
  - Atomicity (Atomarität) — Damit ist gemeint, dass eine Transaktion die kleinste, nicht mehr sinnvoll weiter unterteilbare Einheit darstellt. Eine Transaktion erscheint von außen als eine Einheit.
  - Consitency (Konsistenz) — Die Datenbank befindet sich immer in einem konsistenten Zustand. Falls dieser Zustand nach Ausführung verschiedener SQL-Anweisungen nicht erreicht werden kann, wird die DB wieder in den vorherigen Zustand zurückgesetzt.

Die Transaktion ACID

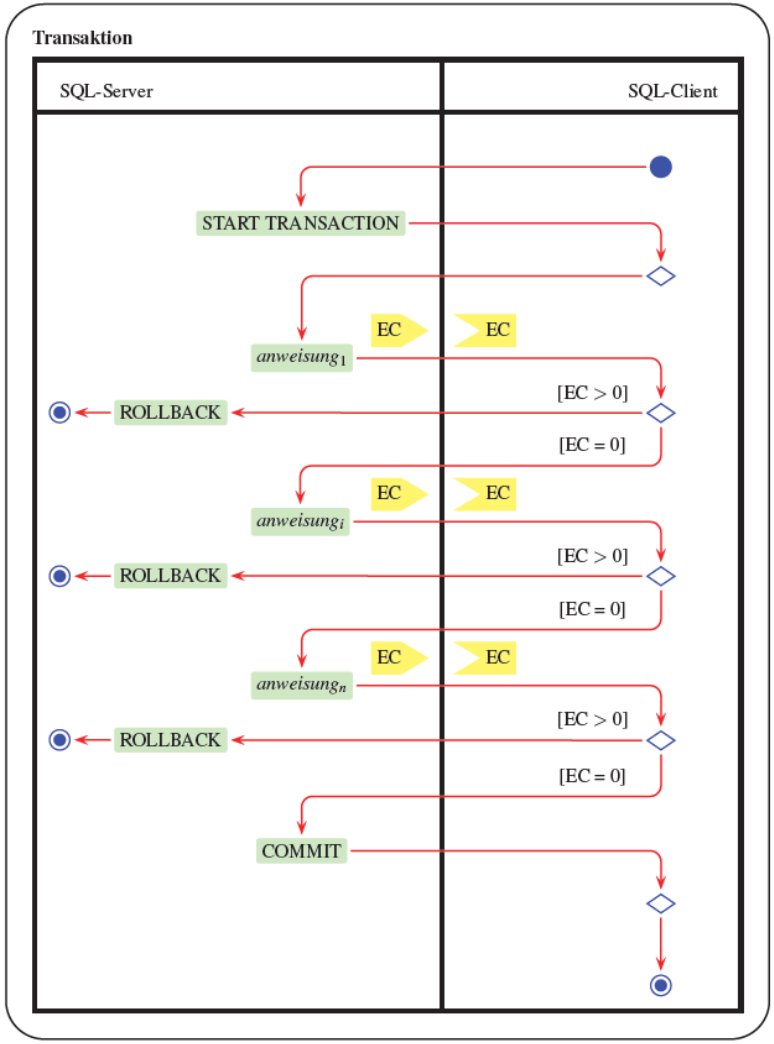
- **Isolation** — Transaktionen sind voneinander isoliert und beeinflussen sich dadurch nicht gegenseitig. Die Ergebnisse einer Transaktion sind bis zu ihrem Ende (commit) für andere Transaktionen unsichtbar. Den Grad der Unsichtbarkeit bestimmt die sogenannte „Isolationsebene“.
- **Durability** (Dauerhaftigkeit) — Alle Aktionen einer abgeschlossenen Transaktion (nach commit) bleiben dauerhaft (persistent) in der DB erhalten. Auch ein Systemabsturz kann dies nicht mehr gefährden.

- Das DBMS ist für die Transaktionsverwaltung zuständig und gewährleistet ACID.
- Die „logische“ Konsistenz kann nicht automatisch geprüft werden und muss durch die Festlegung von Integritätsbedingungen sichergestellt werden.

Zustand einer Transaktion

- Das DBMS befindet sich standardmäßig im AUTO-COMMIT Modus. Dieser kann mit `\SET AUTOCOMMIT = OFF` abgeschaltet werden.
- **begin of transaction** (**BOT**):
  Die Anweisung `BEGIN {TRANSACTION}`(Postgres) kennzeichnet den Beginn einer Transaktion. Der AUTO-COMMIT Modus wird ausgeschaltet.
- **commit**:
  Eine erfolgreiche Transaktion wird mit `COMMIT` abgeschlossen. Es kann auch `END TRANSACTION` angegeben werden. Alle Änderungen werden dauerhaft (persistent) in die DB eingespeichert.
- **rollback**:
  Eine Transaktion wird mit der Anweisung `ROLLBACK` aufgegeben und die Datenbank wird dann in den ursprünglichen Zustand (vor Beginn der Transaktion) zurückgesetzt. Nach einem `COMMIT` oder `ROLLBACK` beginnt eine neue Transaktion. Das DBMS befindet sich wieder im AUTO-COMMIT Modus.

Ablauf einer Transaktion

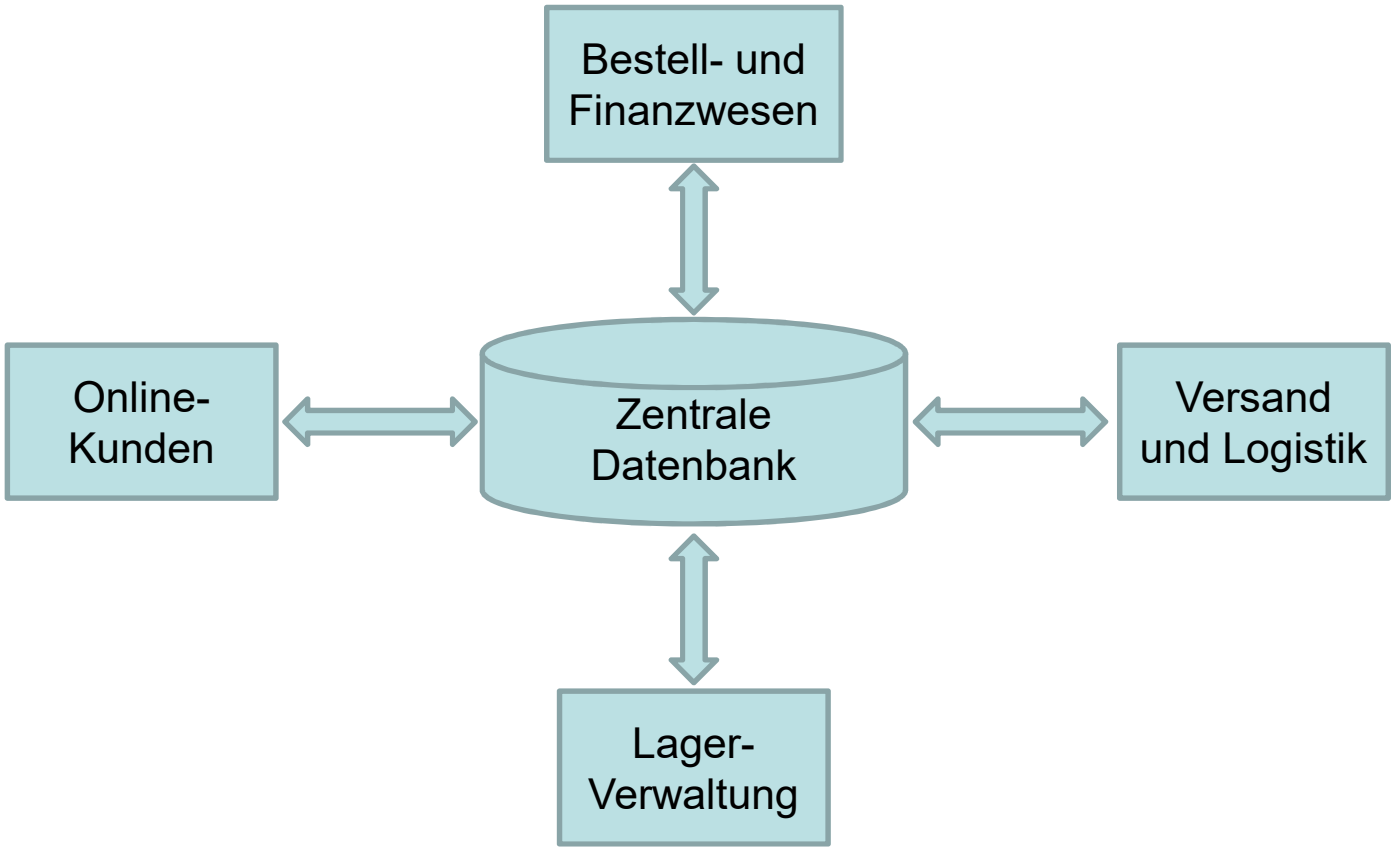


Siehe Buch
(Ralf Adams (2016)
SQL Der Grundkurs für Ausbildung und Praxis 2. Aufl.
Carl Hanser Verlag München ISBN: 978-3-446-45074-5)
Seite 292

Beispiel für eine Transaktion

```
Begin;
UPDATE Lager
SET menge_aktuell = menge_aktuell - 1
WHERE artikel_id = 2
;
UPDATE bestellung SET status = 'versendet'
WHERE bestellung_id = 1
;
Commit;
```

Das zweite Problem: Alle auf Einen



Konkurrierende Zugriffe

- **Szenario:**
  - Kunde (Nr.1) interessiert sich für den Artikel 3 (Office-Produkt) und nimmt diesen in seine Bestellung auf. Gleichzeit wird die Menge des Lagerbestands abgefragt (noch ein Artikel vorhanden). Es wird angezeigt, dass der Artikel geliefert werden kann.
  - Zu selben Zeit entnimmt der Mitarbeiter, der für die Versendung der Artikel in Softwarehaus zuständig ist, den Artikel 3 aus dem Lager.
  - Es ist nun kein Artikel mit der Nummer 3 mehr im Lager. Dieser Artikel ist derzeit auch nicht lieferbar.
  - Die Bestellung beruht auf falschen Daten.
  - Wir sprechen hier von konkurrierenden Zugriffen auf denselben Datenbestand.
  - Wenn Transaktionen parallel und gleichzeitig ausgeführt werden, spricht man auch von **Nebenläufigkeit**.
    Bei ACID gab es die Forderung nach Isolation, also dass sich die Transaktionen nicht gegenseitig beeinflussen.

Unkontrollierter Mehrbenutzerbetrieb

- Für parallel laufende Transaktionen (konkurrierenden Zugriff) gibt es verschiedene Szenarien die auftreten können.

1. **Szenario:**
   - Die Abteilung Versand entnimmt einen Artikel (Nr. 2) und verringert den Lagerbestand.
   - Die Lagerverwaltung arbeitet parallel und verbucht gleichzeigt neue Ware zum Artikel (Nr. 2) .

Versand:	Lager:
1. Lesen `menge_aktuell` = 5	2. Lesen `menge_aktuell` = 5
3. `menge_aktuell = menge_aktuell - 1`	4. `menge_aktuell = menge_aktuell + 10`
5. Schreiben `menge_aktuell` = 4	6. Schreiben `menge_aktuell` = 15

- Der Lagerbestand ist falsch, da die erste Entnahme verloren ging.
- Es entstand der Verlust einer Modifikation oder engl. **lost update**.

Unkontrollierter Mehrbenutzerbetrieb

2. Szenario:

Versand:	Lager:
1. Lesen `menge_aktuell = 5`	4. Lesen `menge_aktuell = 4`
2. `menge_aktuell = menge_aktuell - 1`	5. `menge_aktuell = menge_aktuell + 10`
3. Schreiben `menge_aktuell = 4`	6. Schreiben `menge_aktuell = 14`
7. **`ROLLBACK`**	
8. `menge_aktuell = 5`	

- Der Lagerbestand der Abt. Lager beruht nach Rücksetzen der Änderung auf falschen Daten.
- Das Lesen „ungültigen“ Daten wird **dirty read** genannt.

Unkontrollierter Mehrbenutzerbetrieb

3a. Szenario:

Versand:	Lager:
1. Lesen `menge_aktuell = 5`	2. Lesen `menge_aktuell = 5`
3. … `(keine Update)`	4. `menge_aktuell = menge_aktuell + 10`
6. Lesen `menge_aktuell = 15`	5. Schreiben `menge_aktuell = 15`

- Wiederholte Abfrage der selben Tabellenzeilen führt zu unterschiedlichen Ergebnissen.
- Dieses Lesen „ungültigen“ Daten wird **non repeatable read** genannt.

Unkontrollierte Mehrbenutzerbetrieb

3b. Szenario:

Versand:	Lager:
1. `count(*) from lager` (4)	
2. ……	3. `insert into lager value(5,…)`
3. `count(*) from lager` (5)	

- Entspricht im Prinzip dem Szenario 3a. aber ein neuer Artikel taucht durch Transaktion 2 (Lager) auf.
- Dies wird **Phantom** (Phantomproblem) genannt, da aus dem „Nichts“ neue Zeilen auftauchen.

Isolationsstufen und Daten sperren

- Keine Probleme würden entstehen wenn Transaktionen seriell ablaufen.
- Ist jedoch nicht immer möglich und auch ein Performanceproblem.
- Das DBMS muss die beschriebenen Probleme verhindern.
- Zugriffe auf Daten von anderen Transaktionen (Programmen) werden temporär verhindert.
- Die Daten werden gesperrt.
- Es gibt verschiedene Sperrumfänge (Zeilen, Tabellen, Seiten).
- Über die sogenannten Isolationsstufen wird festgelegt, in welchem Maße Leseoperationen (`SELECT`) gegen konkurrierende Änderungen geschützt werden.

Isolationsstufen

- Es sind vier Isolationsebenen bei einer Transaktion im SQL-Standard festgelegt.
- Davon ist SERIALIZABLE die am strengsten isolierte (alle laufen parallel).
  Es gibt aber pragmatische Gründe den Grad der Nebenläufigkeit zu erhöhen.
- Der Isolationsgrad einer Transaktion gibt an, welches der Phänomenen (dirty read,…) das DBMS zulässt und welches er verhindern soll.
  1. **Read uncommitted**: Erlaubt ein dirty read .
  2. **Read committed**: Ist in Postgres die Voreinstellung. Erlaubt das nicht wiederholbare Lesen (non repeatable read).
  3. **Repeatable read**: Ermöglicht das Lesen von Phantomen.
  4. **Serializable**: Erlaubt kein paralleles Lesen.

Isolationsstufen

- **dirty read**: Lesen von Daten welche von einer gleichzeitigen, uncommitted Transaktion geschrieben wurden.
- **nonrepeatable read**: Das wiederholte Lesen von Daten durch eine Transaktion welche von einer anderen Transaktion zuvor verändert wurde (commit seit dem intitialen Lesen).
- **phantom read**: Beim nochmaligen Ausführen einer Abfrage durch eine Transaktion, entspricht das Ergebnis der Zeilen nicht mehr dem zuvor ermittelten Wert, da eine andere Transaktion zuvor dies geändert (commit) hat (z.B. neue Zeile eingefügt).

Isolationsgrad	dirty read	non repeatable read	phantoms
Read uncomitted	ja	ja	ja
Read committed	nein	ja	ja
Repeatable read	nein	nein	ja
Serializable	nein	nein	nein

Isolationsstufen

- Die Isolationsstufe muss nach dem Beginn der Transaktion festgelegt werden. Dies erfolgt durch das Statement:

```
\SET TRANSACTION ISOLATION LEVEL <Isolationsgrad>
```

Was wäre noch interessant gewesen?

- **Trigger:**
  Ein Trigger ist eine benutzerdefinierte Prozedur, die automatisch immer dann eingesetzt wird, wenn ein Ereignis auf einer Tabelle zwingend mit einer Aktion verbunden werden muss. In ihr können Überprüfungen und Berechnungen durchgeführt werden.

  Z.B.

  - Wenn eine Bestellung gespeichert wird, übernehme die Bestelldaten in eine neue Rechnung.
  - Wenn das Ende-Datum in einem Projekt gesetzt wird, prüfe ob auch alle zugehörigen Leistungen bereits als abgeschlossen gemeldet wurden (Ende-Datum in Leistung not NULL).

- **Stored Procedure (function)**:
  Hierbei handelt es sich um eine Skriptsprache (PL/pgSQL) mit der eine bestimmte Anwendungslogik auf den Datenbankserver verlagert werden kann. Das jeweilige Datenbankprodukt stellt meist eine proprietäre Sprache zur Verfügung, welche einen großen Umgang an Sprachmitteln anbietet.

Was wäre noch interessant?

- **Benutzerrechte und Privilegien**:
  Außer einem Datenbankadministrator sollte jeder Datenbankbenutzer nur die Daten verwalten können, welche er für seine Aufgaben auch benötigt. Dazu können Benutzer(gruppen) angelegt und mit Rechten versehen werden.
- Indizes erstellen, Denormalisierung, Performanceoptimierung
- ….

Ende Kapitel 8