Informationssysteme

Sommersemester 2016

Prof. Dr.-Ing. Sebastian Michel
TU Kaiserslautern

smichel@cs.uni-kl.de

Wie können Größen abgeschätzt werden?

Wichtig: Statistiken über Werte von Attributen, Größen von Relationen
Aber wie kann man diese berechnen und repräsentieren?

Durch Scannen der gesamten Mengen

- Kann hin und wieder berechnet werden, natürlich besser nicht zur Anfragezeit.
- Moderne DMBS haben bestimmte Befehle dafür.

Ausprägungen für Attribut A

- Falls $V(R,A)$ nicht zu groß ist: speichere einen Zähler für jeden Wert von A. Z.B. es gibt 50 Tupel in Relation Professoren mit Rang='C4'.
- Sonst: Gruppierung. Evtl: Exaktes Speichern für eine Teilmenge der Werte (z.B. der häufigsten).

$\Rightarrow$ Histogramme oder parametrisierte Verteilungen!

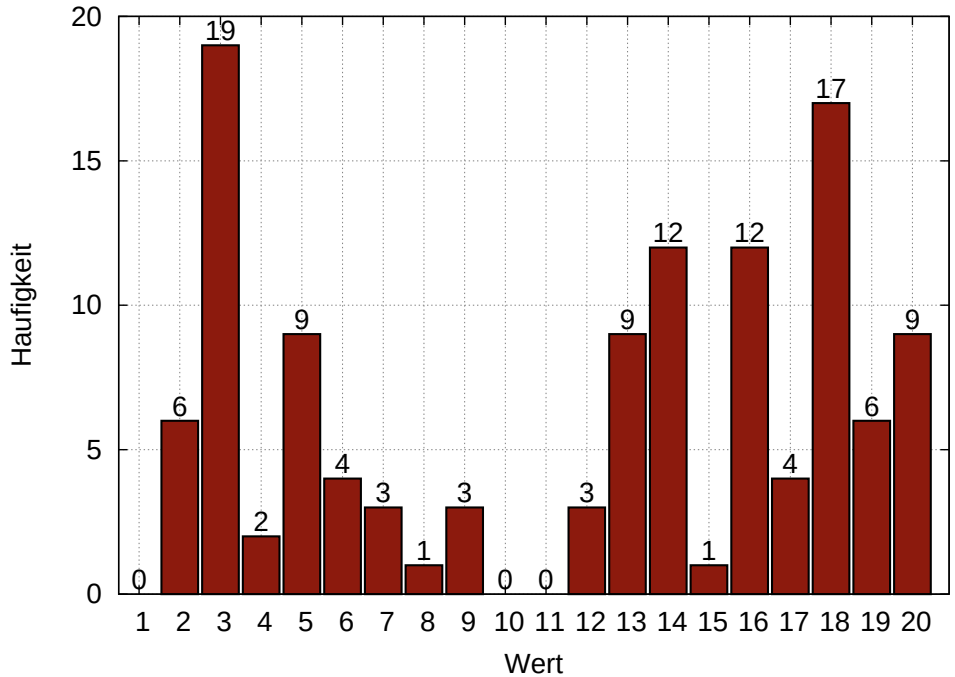
Beispieldaten

{2, 5, 3, 20, 18, 7, 16, 18, 18, 2, 2, 17, 14, 3, 20, 3, 16, 6, 7, 3, 16, 16, 15, 5, 20, 13, 16, 20, 12, 14, 13, 3, 14, 18, 14, 14, 16, 18, 19, 3, 5, 2, 5, 14, 20, 17, 3, 17, 16, 3, 2, 19, 3, 9, 13, 4, 3, 16, 14, 13, 13, 16, 20, 14, 4, 2, 3, 18, 7, 3, 5, 3, 6, 9, 18, 3, 16, 18, 20, 18, 5, 18, 5, 18, 13, 14, 19, 13, 14, 3, 14, 18, 14, 18, 18, 16, 19, 5, 3, 17, 18, 3, 19, 3, 20, 9, 16, 12, 20, 8, 12, 13, 13, 19, 18, 6, 3, 5, 18, 6}

Verteilung der Daten

Wert $\rightarrow$ Häufigkeit.

Dargestellt im "Histogramm-Stil":



Wir sehen: Es liegen 20 Werte im Wertebereich. Es gibt 120 Datenpunkte.

Alternativ: nicht die absolute Häufigkeit sondern normalisiert in [0,1], also "Wahrscheinlichkeit" bei zufälligem Zugriff auf Daten einen bestimmten Datenpunkt zu treffen.

Zusammenfassen/Beschreiben großer Datenmengen

Beschreibung durch parametrisierte Verteilung (Funktion)

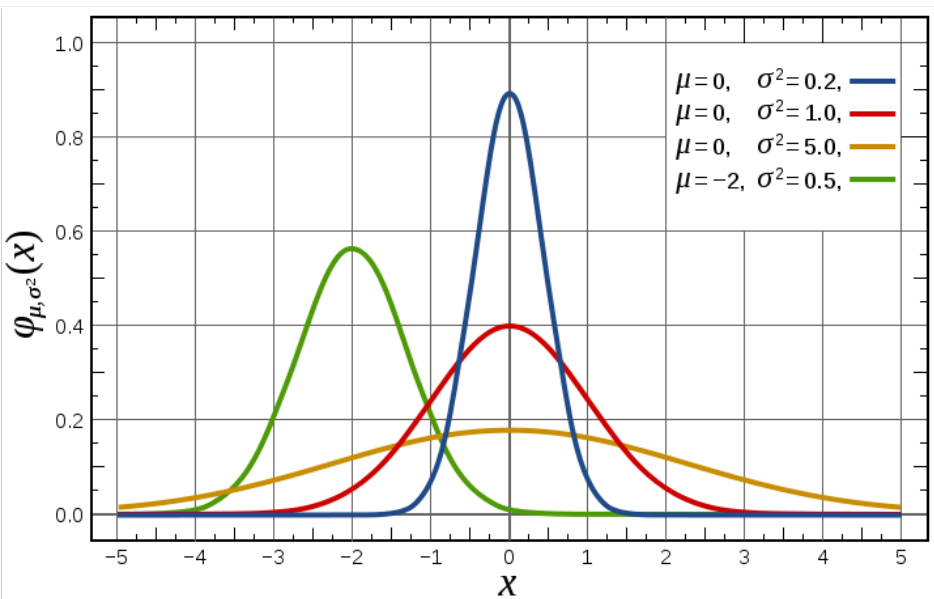
- Z.B. die Daten folgen einer Normalverteilung mit Erwartungswert μ und Varianz σ^2.

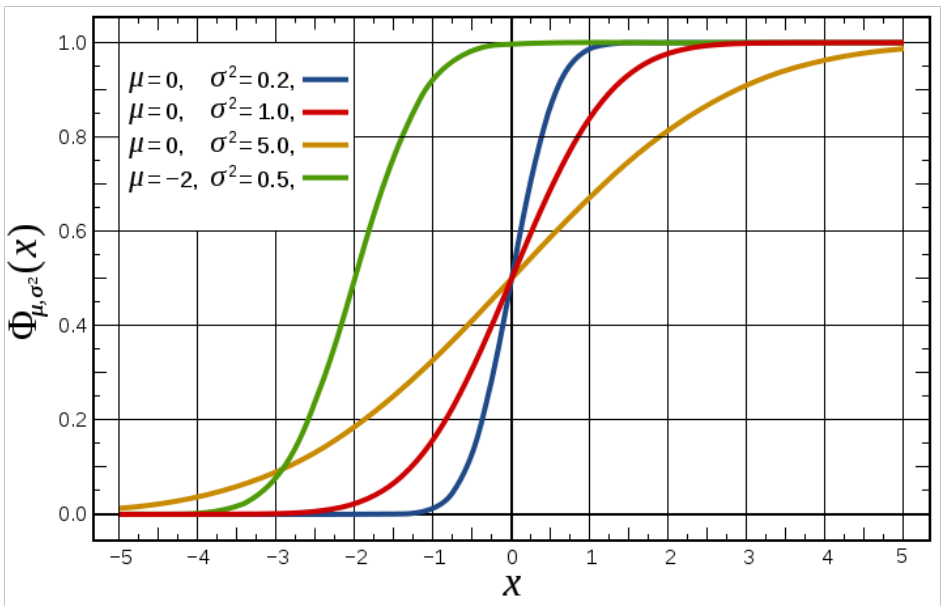
Zusammenfassen von Werten in Zellen (aka. Eimer oder Buckets)

- Wie viele Werte fallen in $[0,5[$, wie viele Werte fallen in $[5,10[$, etc.

Parametrisierte Verteilungen

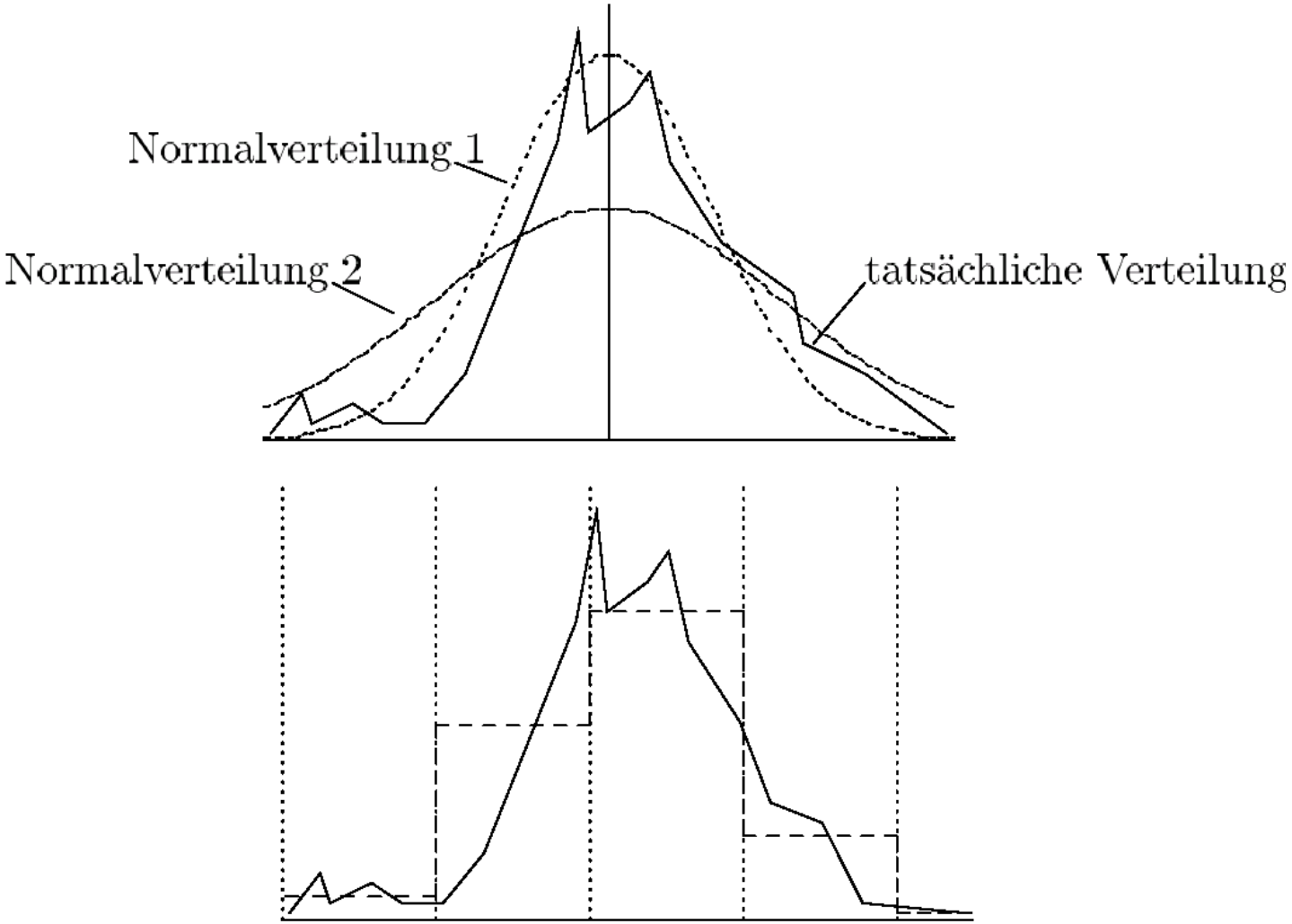
Dichtefunktion und Verteilungsfunktion der Normalverteilung:





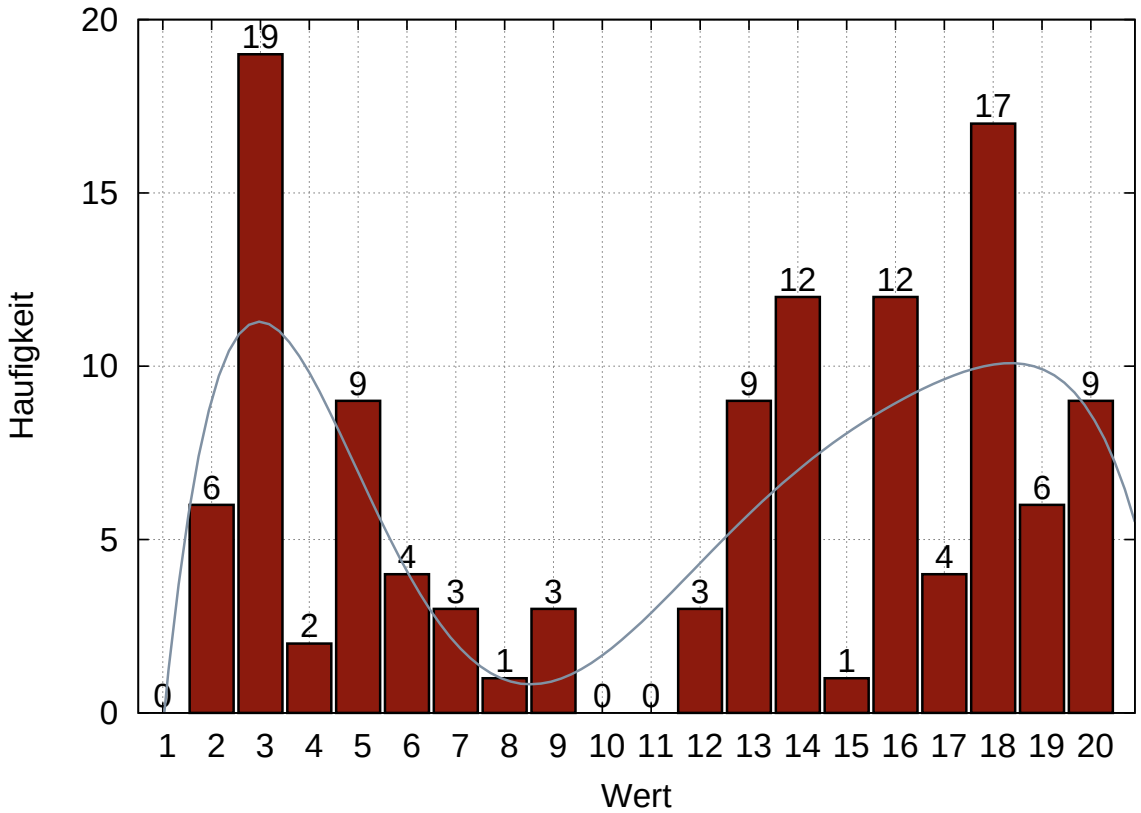
Abbildungen aus Wikipedia

Parametrisierte Verteilungen und Histogramme



Oft ist es schwierig eine tatsächliche Verteilung durch eine parametrisierte Verteilung auszudrücken. Histogramme sind flexibler.

Parametrisierte Verteilungen



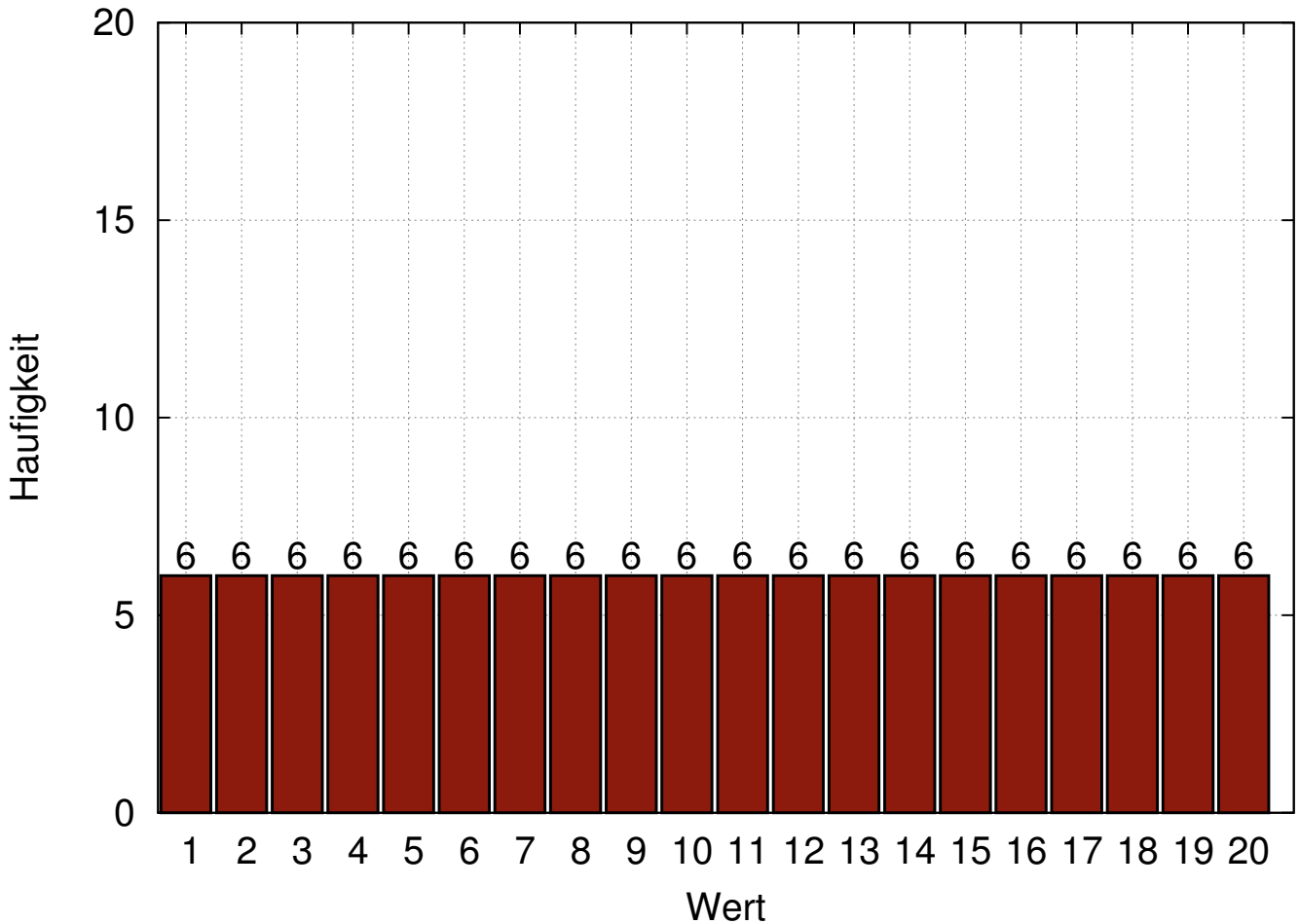
Hier, fit durch Polynom 6. Grades (mit Hilfe des Tools xmgrace):

$$f(x) := -21.884 + 29.533 * x - 9.2268 * x^2 + 1.2529 * x^3 - 0.085019 * x^4$$

$$+0.0028667 * x^5 - 3.8485 * 10^{-5} * x^6$$

Annahme Gleichverteilung

Es liegen 20 Werte im Wertebereich. Es gibt 120 Datenpunkte. Jeder Wert kommt, unter Annahme einer Gleichverteilung, also 6 Mal vor.



Oft nicht sehr realistische Darstellung (aber äußerst kompakt).

Histogramme

Histogram teilt den Wertebereich in Zellen oder Intervalle (Englisch: buckets oder cells). Für jede dieser Zellen wird die Anzahl der Elemente gespeichert, die in die Zelle fallen.

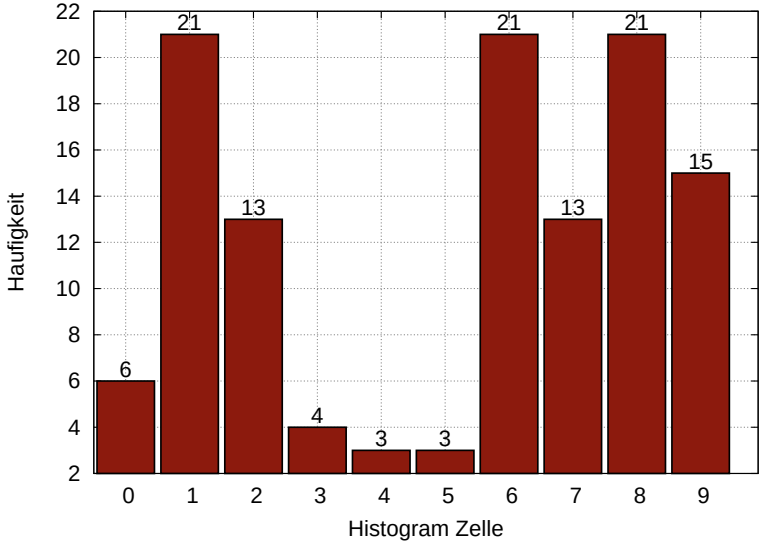
Equi-Width-Histogramme:

- Zellen habe immer die gleiche Breite im Wertebereich

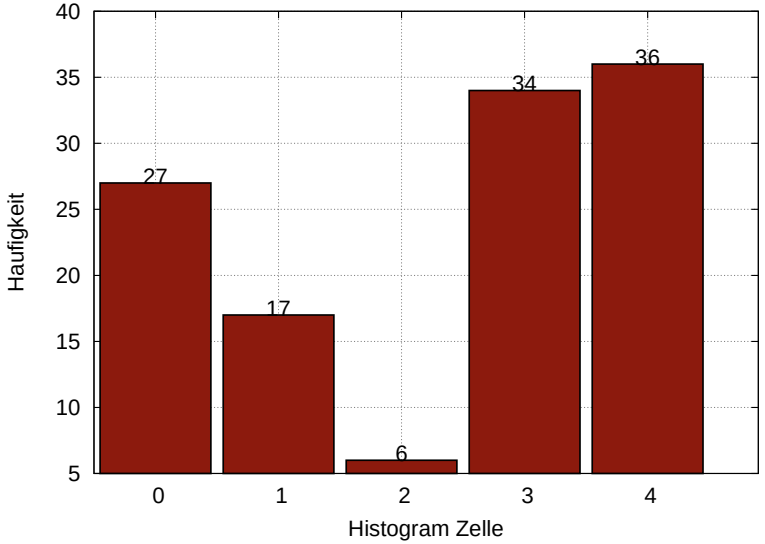
Equi-Depth (oder equi-height genannt) Histogramme:

- Zellen haben die gleiche "Höhe"
- Um dies zu erreichen: Breite der Zellen wird angepasst

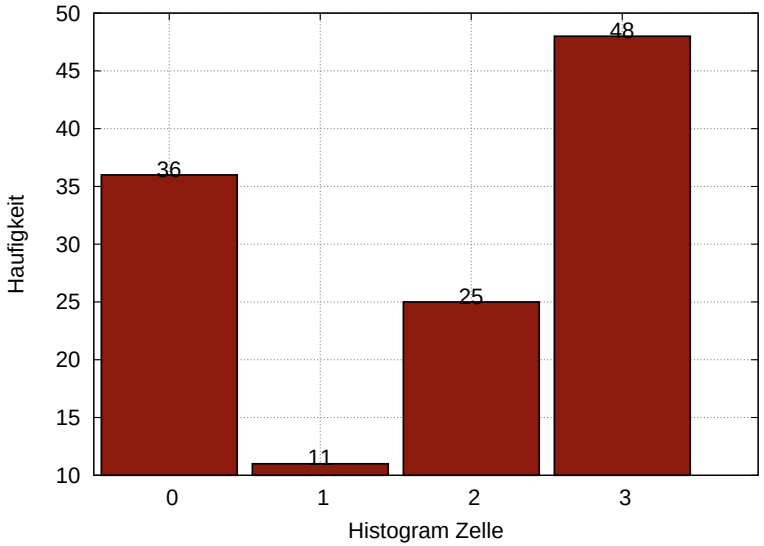
Equi-Width-Histogramme (Beispiele an o.g. Daten)



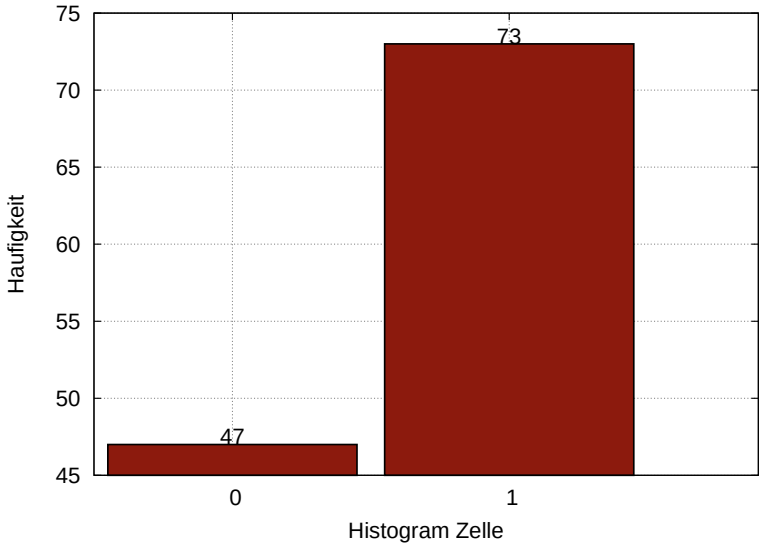
(a) Width = 2



(b) Width = 4

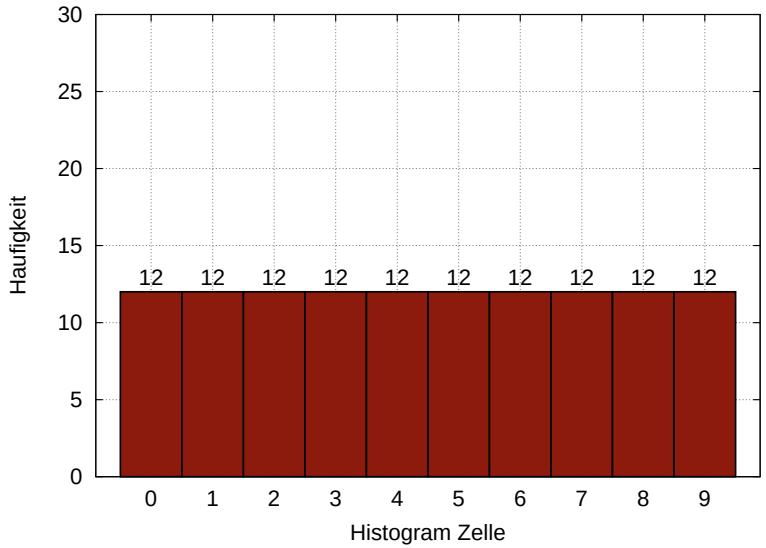


(a) Width = 5

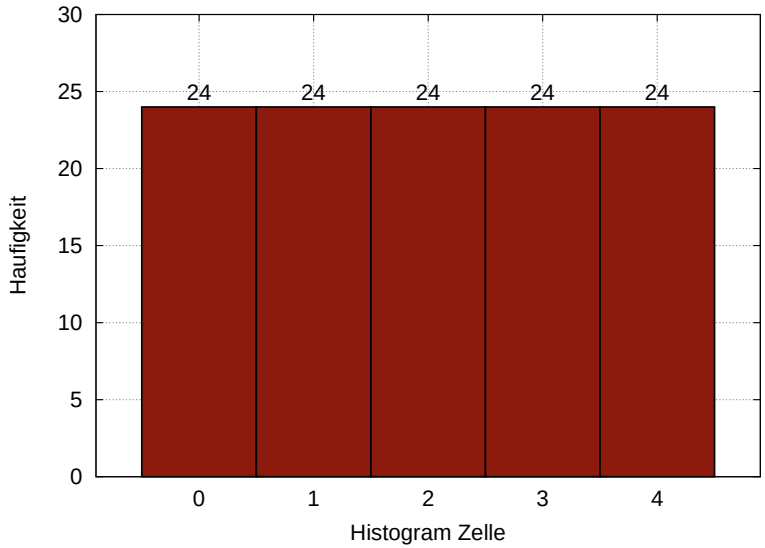


(b) Width = 10

Equi-Depth-Histogramme (Beispiele an o.g. Daten)



(a) Depth = 10%



(b) Depth = 20%

Grenzen der Zellen (je rechter Punkt, d.h. Ende)

- (a) 3,3,6,12,14,16,17,18,19
- (b) 3,12,15,18,20

Punktanfragen und Bereichsanfragen auf Histogrammen

Punktanfragen

- Wie viele Tupel haben den Wert $A = 10$?
- Nachschauen: Zelle in die der Wert 10 fällt.
- Annahme hier: Gleichverteilung innerhalb der Zelle.
- Resultat: Anzahl der Tupel geteilt durch Breite der Zelle.
- Bzw. wenn bereits "normalisiert" dann nur Wert der Zelle.

Bereichsanfragen

- Wie viele Tupel haben einen Wert $A > 10$?
- Nachschauen: In welche Zelle fällt der Wert 10? (Achtung insbesondere bei Equi-Depth Histogrammen)
- Resultat: Summe der Größen der darüber liegenden Zellen und anteilsmäßig diese Zelle (wie oben).
- Oder: Histogramm für kumulative Verteilung betrachten

Schätzung mit Histogrammen

- Obwohl man für diskrete Daten auch Punktanfragen bearbeiten kann, ist dies im kontinuierlichen Fall nicht möglich.
- Es liegen unendlich viele Werte in jeder Histogrammzelle (mit Häufigkeit 0)
- D.h. es kann im kontinuierlichen Fall "nur" nach Häufigkeiten für Intervalle gefragt werden.

Fehlermaße

- Ist für einen exakten Wert x eine Schätzung (Näherungswert) $\hat{x}$ gegeben,
  - so heisst $|\hat{x} - x|$ absoluter Fehler und
  - $\frac{|\hat{x} - x|}{x}$ im Fall $x \neq 0$ relativer Fehler
- Für mehrere solcher Beobachtungen: Sum Squared Error (SSE), Mean Absolute Error (MAE), Mean Squared Error (MSE)

Deskriptive Statistik

- Minimaler und maximaler Wert eines Attributs
- Durchschnittlicher Wert und Median
- Weitere Aussagen über Verteilungen
- Beispielwerte

Anwendungen

- Produkt-Mananger: "In 99,9% aller Fälle liegt die Antwortzeit unseres Systems XYZ unter 10ms."
- Professor: "75% aller Studenten, die die Klausur mitgeschrieben haben, haben mindestens 80 Punkte erreicht."

Quantile einer Verteilung

Definition

Gegeben eine Zufallsvariable X mit Verteilungsfunktion F. Für ein $p \in [0,1]$, definieren wir $F^{-1}(p)$ als kleinsten Wert x, so dass $F(x) \geq p$.

Dieser Wert $F^{-1}(x)$ wird das p Quantil von X genannt. Die Funktion F^{-1} wird **Quantilfunktion** genannt.

- Das p Quantil wird auch $100p$ **Perzentil** genannt.
- Das $1/2$ Quantil, bzw. das 50. Perzentil einer Verteilung wird auch **Median** genannt.
- Das $1/4$ Quantil, bzw. das 25. Perzentil, wird auch **erstes Quartil** genannt,
- das $3/4$ Quantil, bzw. das 75. Perzentil, wird auch **drittes Quartil** genannt.

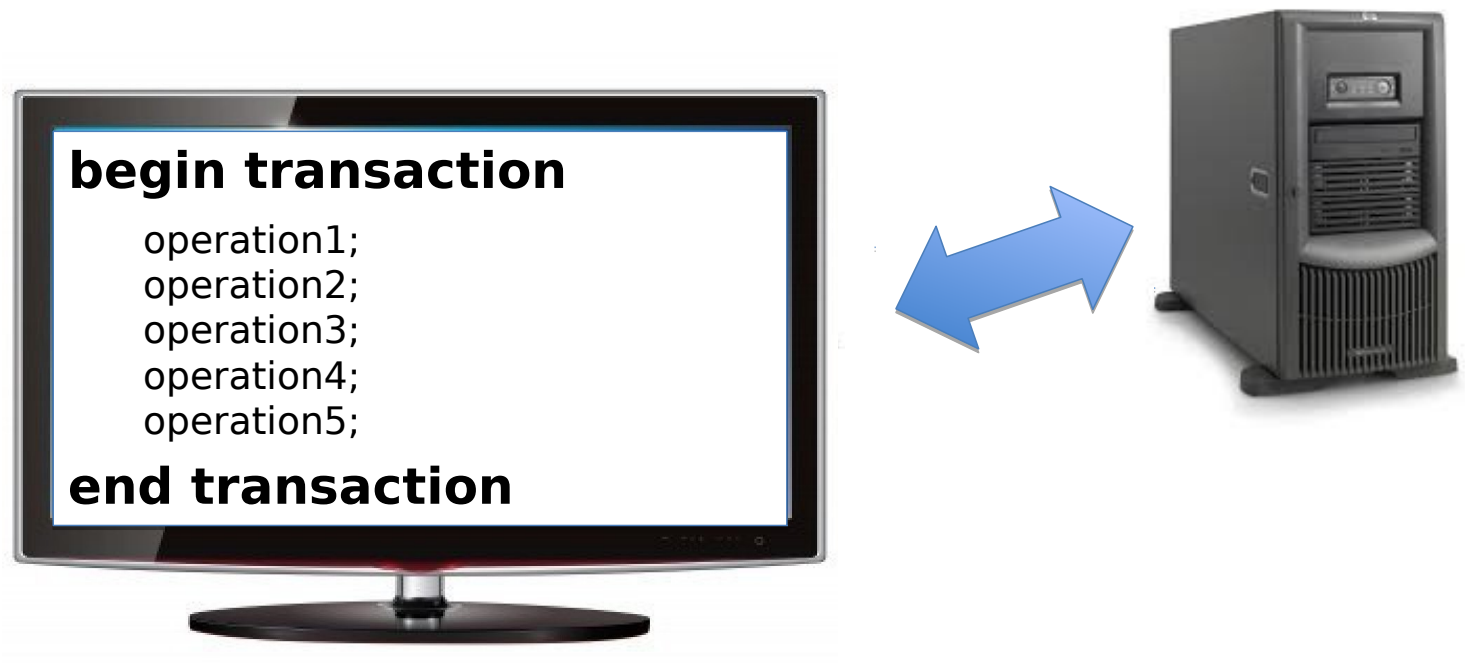
Zusammenfassung Anfrageoptimierung

- Deklarative Anfrage in Form von SQL wird vom Datenbanksystem in konkreten **Anfrageplan** überführt.
- Reihenfolge/Ordnung der Operatoren (der rel. Algebra) des Anfrageplans können durch Regeln verändert werden.
- Haben hier einfache **regelbasierte Anfrageoptimierung** betrachtet: Richtlinien wie Operatoren im Baum generell angeordnet sein sollten.
- Ebenfalls betrachtet: **Einfaches Kostenmodell**, mit dessen Hilfe Kosten (=Anzahl Zwischenergebnisse) abgeschätzt werden können.
- Bessere Abschätzung anhand konkreter Verteilungsmodelle.
- **Ausblick Vorlesung Datenbanksysteme:** Mehr zu Kostenschätzung, Optimierung der Join-Reihenfolge, verschiedene Histogramme, Abschätzung von Anzahl Unique Values durch "Probabilistic Counting", Index-Tuning, Schema Denormalisierung.

Transaktionsverwaltung

Anwendungsprogrammierung: Transaktionen

- Nicht nur eine einzelne SQL-Anweisung, sondern ganze Folge davon, je nach Anwendung.
- Eine oder mehrere Anweisungen werden als Transaktion zusammengefasst bzw. betrachtet. Z.B. Abheben von Geld am Geldautomat.



Einführung

Bei einer typischen Transaktion in einer Bankanwendung:

1. Lese den Kontostand von A in die Variable a: $read(A,a)$;
2. Reduziere den Kontostand um 50 Euro: $a := a - 50$;
3. Schreibe den neuen Kontostand in die Datenbasis: write(A,a);
4. Lese den Kontostand von B in die Variable b: $read(B,b)$;
5. Erhöhe den Kontostand um 50 Euro: $b := b + 50$;
6. Schreibe den neuen Kontostand in die Datenbasis: $write(B,b)$;

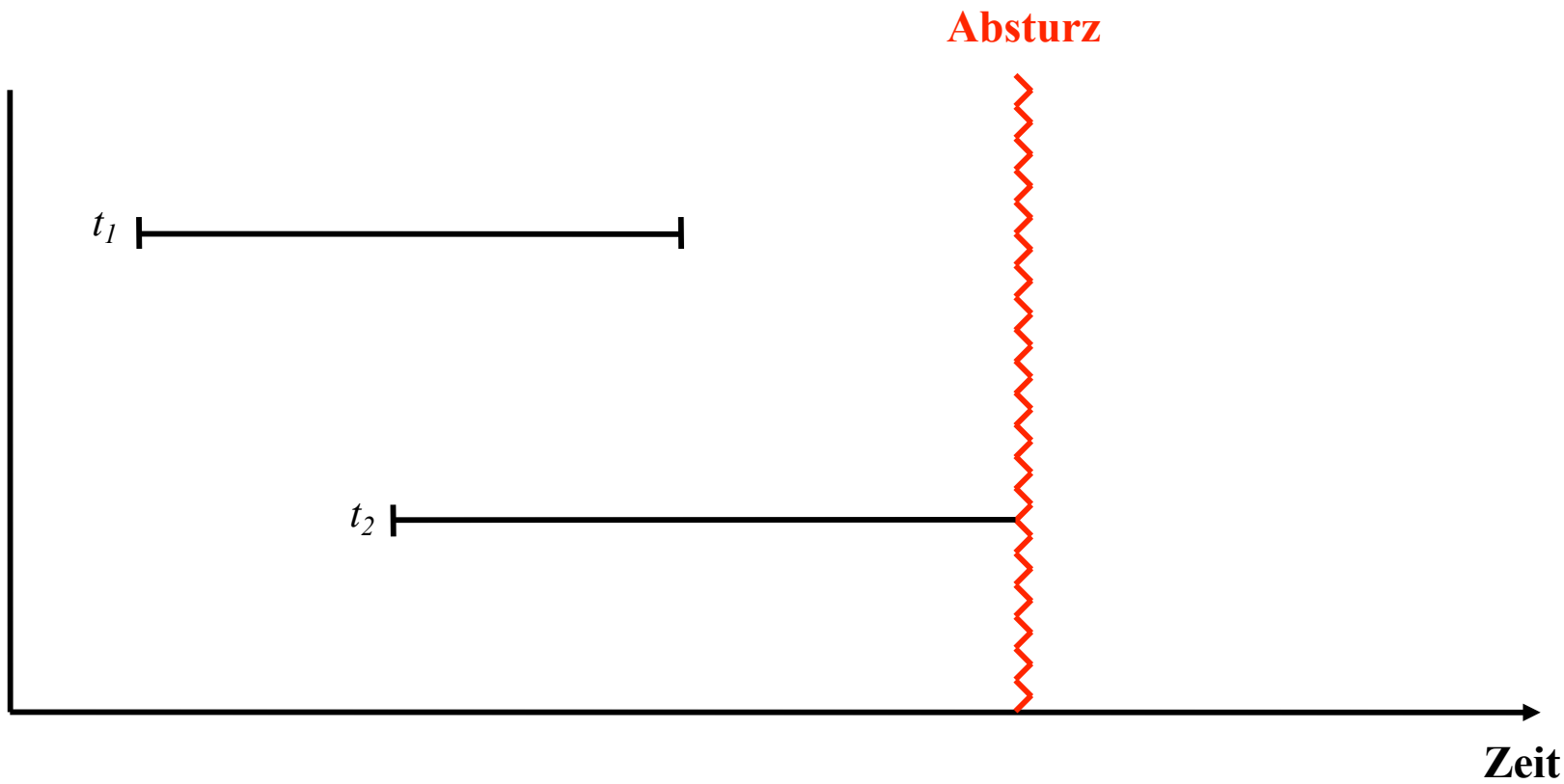
Eigenschaften von Transaktionen: ACID

- **Atomicity (Atomarität):**
  Alles oder nichts
- **Consistency:**
  Kosistenter Zustand der DB $\rightarrow$ konsistenter Zustand
- **Isolation:**
  Jede Transaktion hat die DB "für sich allein"
- **Durability (Dauerhaftigkeit):**
  Änderungen erfolgreicher Transaktionen dürfen nie verloren gehen.,

Operationen auf Transaktione-Ebene

- **begin of transaction (BOT):** Mit diesem Befehl wird der Beginn einer eine Transaktion darstellende Befehlsfolge gekennzeichnet.
- **commit:** Hierdurch wird die Beendigung der Transaktion eingeleitet. Alle Änderungen der Datenbasis werden durch diesen Befehl festgeschrieben, d.h. sie werden dauerhaft in die Datenbank eingebaut.
- **abort:** Dieser Befehl führt zu einem Selbstabbruch der Transaktion. Das Datenbanksystem muss sicherstellen, dass die Datenbasis wieder in den Zustand zurückgesetzt wird, der vor Beginn der Transaktionsausführung existierte.

Transaktionen bei System-Crash



Wie verhält sich solch ein Szenario mit ACID? Was muss beachtet werden? Transaktionen wie t_1 nennt man auch **Gewinner**, Transaktionen wie t_2 dementsprechend **Verlierer**.

Abschluss einer Transaktion

Für den Abschluss einer Transaktion (TA) gibt es drei Möglichkeiten:

1. Den **erfolgreichen** Abschluss durch **commit**.
2. Den **erfolglosen** Abschluss durch ein **abort**.
3. Den **erfolglosen** Abschluss durch einen **Fehler**.

Transaktionsverwaltung in SQL

Beispielsequenz auf Basis des Universitätsschemas:

```
insert into Vorlesungen
            values (5275, 'Kernphysik', 3, 2141);
insert into Professoren
            values (2141, 'Meitner', 'C4', 205);
commit;
```

Transaktionsverwaltung in SQL

- **commit [work]:** Die in der Transaktion vollzogenen Änderungen werden – falls keine Konsistenzverletzung oder andere Probleme aufgedeckt werden – festgeschrieben. Das Schlüsselwort work ist optional, d.h. das Transaktionsende kann auch einfach mit commit "befohlen" werden.
- **rollback [work]:** Alle Änderungen sollen zurückgesetzt werden. Anders als der commit-Befehl muss das DBMS die "erfolgreiche" Ausführung eines rollback-Befehls immer garantieren können.

Sicherungspunkte

- **Sicherungspunkt:**
  Punkt innerhalb einer TA, auf den sich aktive TA zurücksetzen lässt
- **savepoint <name>:**
  definiert den Sicherungspunkt
- **rollback [work] to <name>:** setzt aktive TA zurück bis zum Sicherungspunkt <name>

Beispiel

```
begin;
insert into tab values ...
savepoint A;
insert into tab values ...
savepoint B;
SELECT * FROM tab;
rollback to A;
SELECT * FROM tab;
...
```

JDBC - Transaktionen

Transaktionen

- Bei der Erzeugung eines Connection-Objekts ist (in der Regel) als Default der Modus **autocommit** eingestellt. D.h. nach jeder Aktion wird ein Commit ausgeführt.
- Um Transaktionen als Folgen von Anweisungen abwickeln zu können, ist dieser Modus auszuschalten.

```
conn.setAutoCommit(false);
```

- Für eine Transaktion können sogenannte Konsistenzstufen (isolation levels) wie TRANSACTION_SERIALIZEABLE, TRANSACTION_REPEATABLE_READ usw. eingestellt werden.

```
conn.setTransactionIsolation(
        Connection.TRANSACTION_SERIALIZABLE
);
```

JDBC - Transaktionen (2)

Beendigung oder Zurücksetzung

```
conn.commit();
```

bzw.

```
conn.rollback();      //oder: conn.rollback(savepoint)
```

Sicherungspunkte (Savepoints)

```
Savepoint sp = conn.setSavepoint();   //bzw. mit Namen
Savepoint namedSp = conn.setSavepoint("mySavePoint");
```

Programm kann mit mehreren DBMS verbunden sein

- Selektives Beenden/Zurücksetzen von Transaktionen pro DBMS
- Kein global atomares Commit möglich

JDBC - Transaktionen: Beispiel

http://www.tutorialspoint.com/jdbc/jdbc-transactions.htm

```
try{
   //Assume a valid connection object conn
   conn.setAutoCommit(false);
   Statement stmt = conn.createStatement();
   String SQL = "INSERT INTO Employees  " +
          "VALUES (106, 20, 'Rita', 'Tez')";
   stmt.executeUpdate(SQL);
   //Submit a malformed SQL statement that breaks
   String SQL = "INSERTED IN Employees  " +
               "VALUES (107, 22,  'Sita', 'Singh')";

   stmt.executeUpdate(SQL);
   // If there is no error.
   conn.commit();
}catch(SQLException se){
   // If there is any error.
   conn.rollback();
}
```

Wie unterstützt das DMBS Transaktionen?

Mehrbenutzersynchronisation (Isolation)

- Semantische Korrektheit bei Nebenläufigkeit
- Serialisierbarkeit
- Schwächere Isolationsstufen (Isolation Levels)

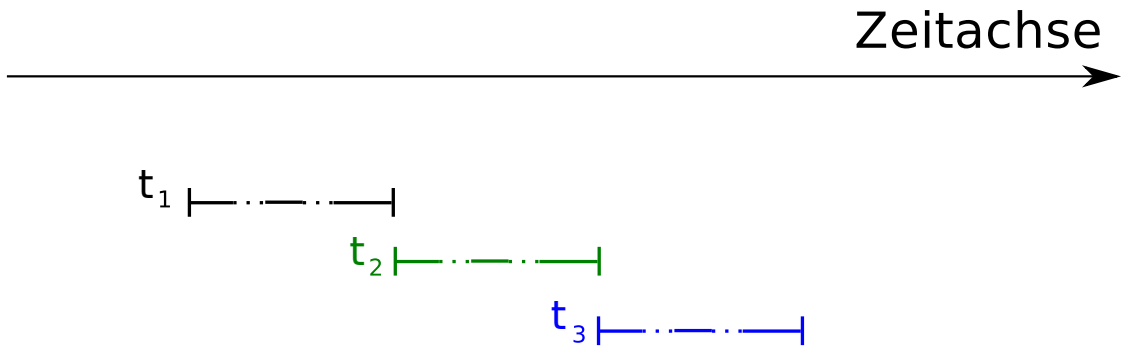
Recovery (Atomicity and Durability)

- Zurücksetzen teilweise ausgeführter TA
- Vervollständigen der Aktionen von TA
- Sicherstellen der Persistenz von TA

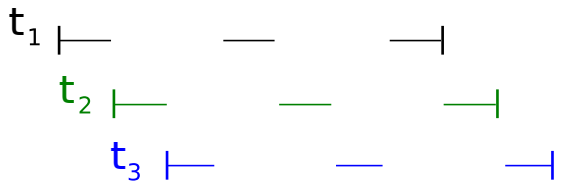
Warum Mehrbenutzerbetrieb?

Ausführung der drei Transaktionen t_1, t_2 und t_3:

(a) im Einzelbetrieb



(b) im (verzahnten) Mehrbenutzerbetrieb



Man möchte, dass die verzahnte Ausführung "semantisch" äquivalent zu einer seriellen Ausführung (d.h. z.B. $t_1\ t_3\ t_1$) ist.

Das lost-update Problem

t_1	Time	t_2
	/* $x = 100$ */	
$r(x)$	1	
	2	$r(x)$
/*update $x := x + 30$ */	3	
	4	/* update $x := x + 20$ */
$w(x)$	5	
	/* $x = 130$ */	
	6	$w(x)$
	/* $x = 120$*/	

Das dirty-read Problem

t_1	Time	t_2
$r(x)$	1	
/* $x := x + 100$ */	2	
$w(x)$	3	
	4	$r(x)$
	5	/* $x := x - 100$ */
failure & rollback	6	
	7	$w(x)$

Wir betrachten nun ganz kurz Recovery, danach etwas ausführlicher Mehrbenutzersynchronisation.

Welche Aspekte von

werden durch Recovery adressiert?

Implikationen von ACID auf Anforderungen zu Recovery

Durability

- Änderungen an der Datenbank, die durch erfolgreich(!) abgeschlossene (d.h. committed) Transaktionen verursacht wurden, müssen dauerhaft gespeichert sein.
- D.h. bei einem Crash der DB muss nach Wiederanlauf geschaut werden, ob dies tatsächlich der Fall ist.

Atomicity

- Falls eine Transaktion noch nicht erfolgreich abgeschlossen wurde und ein DB-Crash auftritt, muss beim Wiederanlauf darauf geachtet werden, dass etwaige Änderungen in der Datenbasis rückgängig gemacht werden.

Fehlerklassifikation

Lokaler Fehler in einer noch nicht festgeschriebenen (committed) Transaktion:

- Wirkung der TA muss zurückgesetzt werden

Fehler mit Hauptspeicherverlust:

- Abgeschlossene TAs müssen erhalten bleiben
- Noch nicht abgeschlossene TAs müssen zurückgesetzt werden

Fehler mit Hintergrundspeicherverlust:

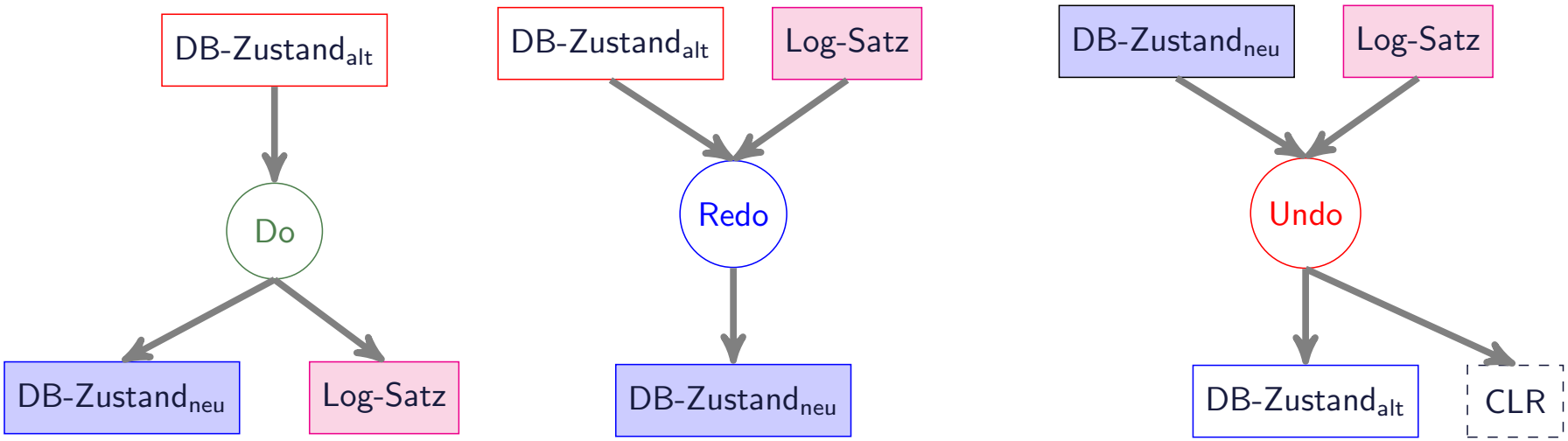
- Archiv einspielen

Vorsorge für den Fehlerfall

Logging

- Sammlung redundanter Daten bei Änderungen im Normalbetrieb, als Voraussetzung für Recovery
- Einsatz im Fehlerfall (Undo-, Redo-Recovery)

Do-Redo-Undo-Prinzip



Recovery-Oriented Computing

Systemverfügbarkeit A (availability)

- MTTF: Mean Time To Failure
- MTTR: Mean Time To Repair

$$A = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

Warum Recovery-Oriented Computing?

- Hardwarefehler, Softwarefehler passieren (sind nicht vermeidbar) und müssen adressiert werden.
- Lange Systemausfälle sind sehr sichtbar (z.B. Amazon, Facebook, Ebay!)

Number of Nines & Entwicklungsziele

Availability wird manchmal beschrieben in Anzahl von Neunen (Nines), wie in "five nines", oder 99.999%. 99.99% entspricht ungefähr 1 Minute Ausfall in einer Woche, bzw. circa 50 Minuten Ausfall in einem Jahr.

- "Build a system used by millions of people that is always available – out less than 1 second per 100 years = 8 9's of availability" (J. Gray: 1998 Turing Award Lecture)

Jim Gray: We have added three 9s in 45 years (starting with 90%), or about 15 years per order-of-magnitude improvement in availability. We should aim for five more 9s: an expectation of one second outage in a century. This is an extreme goal, but it seems achievable if hardware is very cheap and bandwidth is very high. One can replicate the services in many places, use transactions to manage the data consistency, use design diversity to avoid common mode failures, and quickly repair nodes when they fail. Again, this is not something you will be able to test: so achieving this goal will require careful analysis and proof.

Quelle: bild.de, 27.01.2015

- Ausfall hat (angeblich) ca. 1 Stunde gedauert
- Nehmen wir an, dass dies ein Mal im Jahr geschieht.
- Wie groß ist die Availability? Wie viele "Neunen"?

Wie kann man annährend $A = 1{,}0$ erreichen?

- MTTF $\rightarrow \infty$?
- MTTR $<<$ MTTF!

Es gibt Fehler die man nur sehr schwer oder gar nicht nicht durch testen in den Griff bekommen kann. Sie treten nichtdeterministisch auf, oft aufgrund von Nebenläufigkeit in Threads, unter hoher Last, oder in anderen seltenen Fällen.

Solche Probleme werden auch "Heisenbugs" genannt (Jim Gray); nach Heisenbergs Unschärferelation.

D.h. ein schneller Wiederanlauf (Recovery) ist essentiell für hohe Verfügbarkeit (Availability); da diese "Heisenbugs" die MTTF in der Praxis einschränken.

Grundlagen der DB-Recovery

Aufgabe des DBMS

- Automatische Behandlung aller erwarteten Fehler

Was sind erwartete Fehler?

- DB-Operation wird zurückgewiesen, Commit wird nicht akzeptiert, ...
- Stromausfall, DBMS-Probleme
- Geräte funktionieren nicht (Spur, Zylinder, Platte defekt)
- Beliebiges Fehlverhalten der Gerätesteuerung
- ...

Fehlermodelle von (zentralisierten) DBMS

- Transaktionsfehler
- Systemfehler
- Gerätefehler
- Katastrophen

Recovery-Arten

Transaktions-Recovery

- Zurücksetzen einzelner (noch nicht abgeschlossener) TA im laufenden Betrieb (TA-Fehler, Deadlock, etc.)

→ Vollständiges Zurücksetzen auf BOT (TA-Undo)

→ Partielles Zurücksetzen auf Rücksetzpunkt (Savepoint) innerhalb der Transaktion

Crash-Recovery nach Systemfehler

- Wiederherstellen des jüngsten transaktionskonsistenten DB-Zustands:

→ (partielles) Redo für erfolgreiche TA (Wiederholung verlorengegangener Änderungen

→ Undo aller durch Ausfall unterbrochenen TA (Entfernung der Änderungen aus der DB)

Recovery-Arten (2)

Medien-Recovery nach Gerätefehler

- Spiegelplatten, bzw.
- vollständiges Wiederholen (Redo) aller Änderungen auf einer Archivkopie

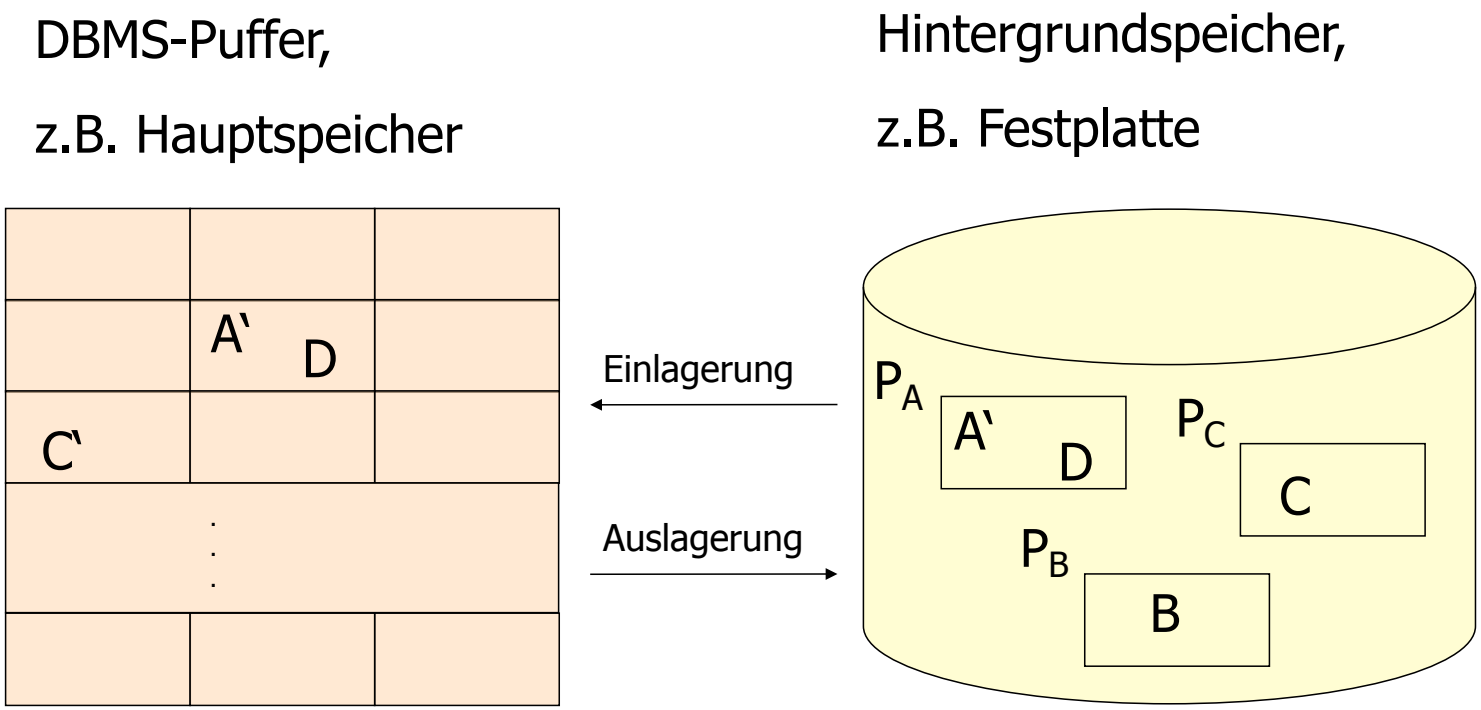
Katastrophen-Recovery

- Nutzung einer aktuellen DB-Kopie in einem "entfernten" System oder
- stark verzögerte Fortsetzung der DB-Verarbeitung mit repariertem/neuem System auf Basis gesicherter Archivkopien

Betrachten im Folgenden kurz Überlegungen zu Crash-Recovery nach Systemfehler.

Zweistufige Speicherhierarchie

- Seiten P_i
- Datensätze A,B,C,D, ..



Force und Steal

Ersetzung von Puffer-Seiten:

- ¬**steal:** Ersetzung von Seiten, die von einer noch aktiven Transaktion modifiziert wurden, ausgeschlossen ⇒ "dreckige" Seiten (dirty pages) müssen im Puffer bleiben.
- **steal:** Jede nicht fixierte Seite ist prinzipiell ein Kandidat für die Ersetzung, falls neue Seiten eingelagert werden müssen, auch "dreckige" Seiten.

Einbringen von Änderungen abgeschlossener TAs:

- **force:** Änderungen werden bei commit auf den Hintergrundspeicher geschrieben (flush).
- ¬**force:** geänderte Seiten können auch nach commit im Puffer verbleiben.

dirty page = Inhalt der Seite im HS ≠ Inhalt der Seite auf FS

Auswirkungen auf Recovery

Undo: entfernt ungültige Einträge aus der DB.

Redo: fügt gültige Einträge in die DB sein.

	force	**¬force**
¬steal		
steal		

- Was ist besser? steal oder ¬steal?
- Was ist besser? force oder ¬force?
- Annahme: Schreiboperationen von Seiten müssen atomar sein.

Auswirkungen auf Recovery

Undo: entfernt ungültige Einträge aus der DB.

Redo: fügt gültige Einträge in die DB sein.

	force	**¬force**
¬steal	• kein Undo • kein Redo ⇒ **keine** Recovery	• Redo • kein Undo
steal	• kein Redo • Undo	• Redo • Undo

- Was ist besser? steal oder ¬steal?
- Was ist besser? force oder ¬force?
- Annahme: Schreiboperationen von Seiten müssen atomar sein.

WAL-Prinzip und Commit-Regel bei Log-basierter Recovery

Write-Ahead-Log-Prinzip (WAL)

Bevor eine modifizierte Seite ausgelagert werden darf, müssen alle Log-Einträge, die zu dieser Seite gehören, in die Log-Datei und das Log-Archiv ausgeschrieben werden.

Commit-Regel (Force-Log-at-Commit)

Bevor eine Transaktion festgeschrieben (**committed**) wird, müssen alle "zu ihr gehörenden" Log-Einträge auf stabilen Storage ausgeschrieben werden.

Details zur Realisierung von Recovery werden in der VL Datenbanksysteme besprochen.

C. Mohan et al. ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging. `http://www.cs.berkeley.edu/~brewer/cs262/Aries.pdf`