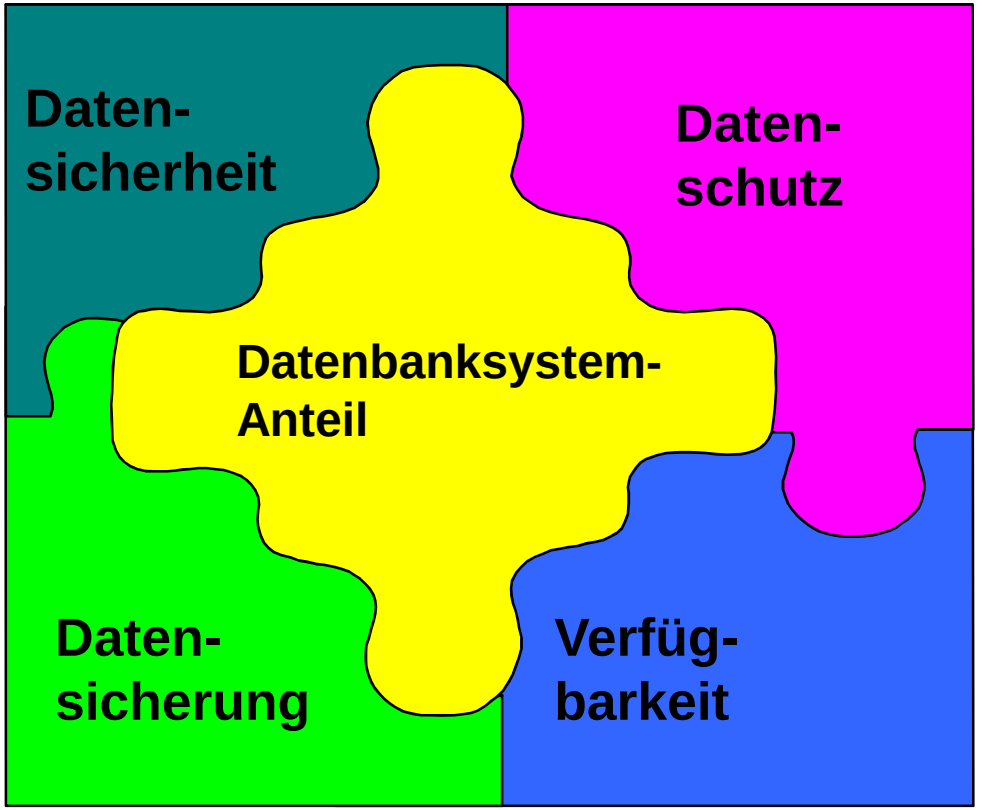
WP Datenbankdesign

Kapitel 5: Datenbankbetrieb

- Datenbanksicherheit
- Transaktionen
- Mehrbenutzerbetrieb

Sicherheit und Datenschutz in Datenbanken

1. Schutz (personenbezogener) Daten gegen Missbrauch
2. Sicherheit der Daten vor Fahrlässigkeit, technischen Fehlern, …
3. Sicherheit der Integrität der Daten
4. Sicherung der Daten gegen Verlust, Löschung
5. Sicherstellung der Verfügbarkeit



Datenschutz

GG, StGB, IuKDG,

BGB, BDSG, BetrVG,

TKG, UrhG, WiKG

Daten werden überwiegend Maße elektronisch gespeichert und übermittelt.

Sie können unbegrenzt gespeichert und orts- und zeitunabhängig zusammengeführt werden.

Gefahren

- **Verfälschung**
- **Ausspähung / Nutzung durch Unberechtigte**

Dadurch Beeinträchtigung von Persönlichkeitsrechten

Datenschutz

- Datenschutz ist der **Schutz des Menschen und seiner persönlichen Daten** vor Missbrauch durch Andere.
- Nicht Schutz der Daten, sondern Schutz der Personen, über die die Daten etwas aussagen.

1. Datenschutzgesetz(e)
2. Selbstregulierung
3. Selbstschutz

Selbstregulierung und Selbstschutz vor allem im Internet

- Gesetze sind hier teilweise inhaltlich nicht passend / nicht anwendbar, hinken der Realität hinterher
- Problem der Gültigkeit der Gesetze

  (Gesetze sind länderspezifisch, Internet ist global)

Sicherung gegen unberechtigten Zugriff

1. **Authentifizierung**
   - durch User-Name und Passwort
   - durch Betriebssystem und andere Dienste
2. **Autorisierung**
   - Ressourcen-Limits (CPU-Zeit, Speicherplatz) innerhalb der DB
   - Vergabe von Zugriffsrechten
   - auch über Views möglich
3. **Auditing**
   - Überwachung spezifizierter Aktionen
4. **Wfstdimvfttfmvoh**

vgl. Kryptografie
Alphabet -1

Rechtevergabe in SQL

- Voraussetzung: Benutzer mit CREATE USER … eingerichtet
- Rechte werden mit GRANT vergeben
- Es wird zwischen System- und Objektrechten unterschieden
  - **Systemrechte** betreffen das
    - CREATE, ALTER und DROP z.B. von TABLE, INDEX, USER, ROLE, TABLESPACE, SESSION etc.
    - Weiterhin einige Spezialbefehle
    - Unterschied zwischen Rechten im eigenen Schema und in allen Schemata
  - **Objektrechte** umfassen insbesondere das SELECT, INSERT, UPDATE, DELETE, EXECUTE etc. und beziehen sich zusätzlich noch auf ein Datenbankobjekt (Tabelle, Stored Procedure, Trigger etc.).
- Bei beiden Varianten wird ein Recht an Benutzer oder Rollen vergeben

Beispiele

- `CREATE USER scott IDENTIFIED BY tiger;`
- Beispiel für Vergabe von Systemrechten:

```
GRANT CREATE TABLE TO s4711;
GRANT DROP TABLE TO s4711;
GRANT DROP USER TO lokalem_administrator;
GRANT CREATE ANY TABLE TO gerken;
```

Rolle

Überall

DB-Objekt

- Beispiel für Vergabe von Objektrechten:

```
GRANT SELECT ON Tabelle1 TO s4711;
GRANT SELECT (name, adresse, ID) ON Tabelle2 TO
s4711;
GRANT ALL ON Tabelle3 TO zkn, grk;
GRANT ALL ON Tabelle3 TO PUBLIC;
```

Alle Rechte

Jeder

Weitergabe von Rechten

- Ein vergebenes Recht kann zunächst nur genutzt, aber nicht weitergegeben werden.
- Dafür zwei Spezialvarianten:
  - GRANT .... **WITH GRANT OPTION**;
    Damit darf der Benutzer das erhaltene Recht an andere Benutzer weitergeben und genau diesen auch wieder entziehen.
  - GRANT .... **WITH ADMIN OPTION**;
    Damit darf der Benutzer das erhaltene Recht an andere Benutzer weitergeben und allen Benutzern entziehen. Er ist der Verwalter des übertragenen Rechts.

- Beispiel:
```
GRANT SELECT ON Tabelle1 TO gerken WITH ADMIN
OPTION;
```

Rechte entziehen

- Ein vergebenes Recht kann mit REVOKE wieder entzogen werden.
- Beispiel:

```
REVOKE DROP USER FROM grk;
REVOKE INSERT ON Tabelle1 FROM s4711;
REVOKE ALL ON Tabelle2 FROM s4711, s0815;
```

- Voraussetzungen
  - Bei System-Rechten hat man selbst das Recht mit ADMIN OPTION erhalten.
  - Bei Objekt-Rechten hat man das Recht selbst an jemand anderen vergeben und „holt es sich jetzt zurück“.
  - Wenn man für ein Objekt Rechte mit ADMIN OPTION erhält (oder Objekt besitzt) kann man Rechte immer entziehen.
  - Funktioniert nur mit solchen Rechten, die der andere Nutzer über GRANT erhalten hat.

Rechte entziehen

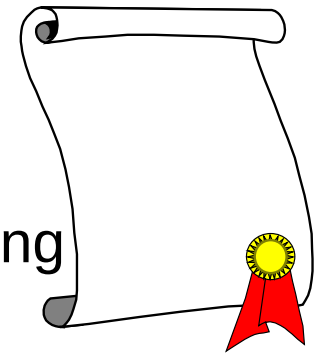
Einige Hinweise zu typischen Problemen:

- Wenn ein Benutzer ein Recht mehrfach erhalten halt, muss es auch mehrfach entzogen werden.
- Das Entziehen mit REVOKE .... FROM PUBLIC; entzieht nur den Benutzern, die das Recht über diesen Befehl erhalten haben. Explizit vergebene Rechte werden nicht entzogen.
- Ein Entziehen mit REVOKE DROP ANY TABLE ... verhindert nicht das Löschen eigener Tabellen!
- Wenn Benutzer A ein Recht mit „ADMIN OPTION“ an Benutzer B vergibt, kann Benutzer B danach Benutzer A das Recht darauf entziehen. **Also vorsichtig sein...**
- Ein Revoke wirkt sich auch auf mit „with grant option“ mögliche weitere Vergaben aus

Rollenbasierte Rechteverwaltung

Was ist rollenbasierte Rechteverwaltung?

- Berechtigungen werden in Rollen geeignet zusammengefasst
- Rechteänderungen werden an wenigen Rollen und Ressourcen und nicht mehr an einer Vielzahl von Benutzern und Objekten vorgenommen



Was ermöglicht dieser Ansatz?

- Vereinfachung und Verbesserung der Rechteverwaltung (Single point of administration)
- Reduzierung der Gesamtkosten der Rechteverwaltung und Beschleunigung der Vorgänge durch geringere Aufwände und größere Sicherheit aufgrund einer besseren Wartbarkeit

	T	U	V
Rolle A	R	-	-
Rolle B	U	R	
.....	-		

	A	B	C
User A	J	-	-
User B	J	J	-
	J	-	

Arbeiten mit Rollen

Definition von Rollen mit „CREATE ROLE“ (In SQL-99 neu definiert)

- Beispiel:
  ```
  CREATE ROLE Student NOT IDENTIFIED;
  CREATE ROLE Kanzler IDENTIFIED BY Angie;
  ```
- Rechte können an Benutzer und Rollen vergeben werden (GRANT und REVOKE funktionieren auch auf Rollen)
- Rollen werden wie Rechte an Benutzer / andere Rollen vergeben. Beispiel: `GRANT Student TO s4711;`
- Erlaubt sind Hierarchien von Rollen.
- Verboten sind zirkuläre Rollen, z.B.
  Student ist Mitarbeiter und Mitarbeiter ist Student.
- Rollen werden beim Login aktiviert.
- Manueller Wechsel möglich:
  ```
  SET ROLE student;
  SET ROLE ALL;
  SET ROLE ALL EXCEPT administrator;
  ```

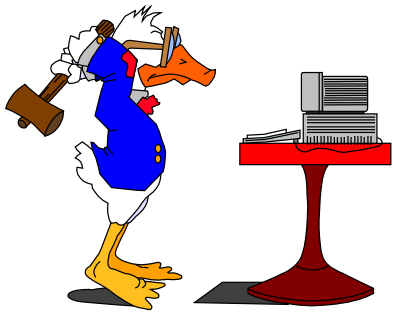
Sicherung der Integrität der Daten

1. **Physische Integrität**
   - Vollständigkeit der Zugriffspfade und Speicherstrukturen
   - Vollständigkeit des Data Dictionaries
2. **Datenmodellintegrität**
   - Datentypen
   - Primary key
   - Referenzielle Integrität
3. **Semantische Integrität**
   - Wertebereiche
   - Not null, default
   - Constraints und Trigger
4. **Operationale (pragmatische) Integrität**
   - Transaktionen
   - Nebenläufigkeit (Mehrbenutzerbetrieb)

eigenes Kapitel

Sicherung der Daten gegen Verlust

Ursachen für Datenbankfehler



1. Hardwarefehler
2. Systemsoftwarefehler (insbes. DBMS)
3. Anwenderprogrammfehler
4. Böswilligkeit

Möglichkeiten für den Fehlerumfang / Auswirkungen

1. der (gesamte) Datenbestand ist physikalisch zerstört
2. es sind fehlerhafte Daten gespeichert worden
3. keine fehlerhaften Sekundärspeicherdaten

Betroffene von Fehlern

1. das gesamte DBMS
2. ein Anwenderprogramm

Konsequenz

Maßnahmen zur Datensicherung und zur Wiederherstellung im Fehlerfall

1. Datenbankkopie
2. After-Image und Before-Image
3. Flashback

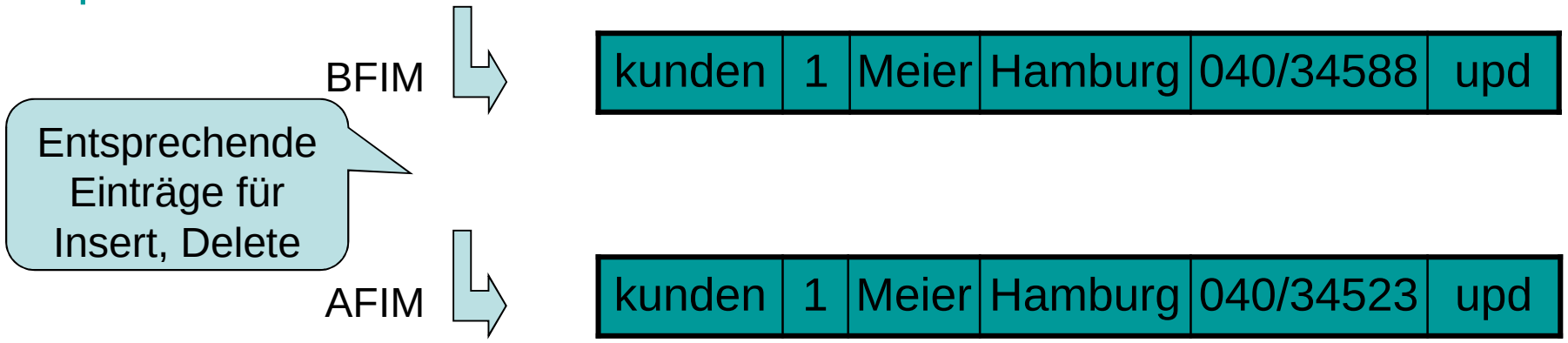
Before- und After Images

Before-Image Geänderte Tupel je TA mit alten Werten, für Rollback
Lebensdauer nur bis EOT

After-Image Geänderte Tupel nach der Änderung für Restore, Roll Forward
Lebensdauer bis zur Erstellung einer neuen DB-Kopie
Heißt bei Oracle Redo-Log

Bespiel:
insert into kunden values(1,‘Meier‘,‘Hamburg‘,‘040/34588‘)

update kunden set Telefon= ‘040/34523‘ where KdNr=1



Ausschreiben von Updates auf Sekundärspeicher

Änderungen von Transaktionen (INSERT, UPDATE, DELETE) werden zuerst temporär im Database Buffer Pool vorgenommen. Diese müssen noch permanent gemacht, d.h. auf Sekundärspeicher ausgeschrieben werden.

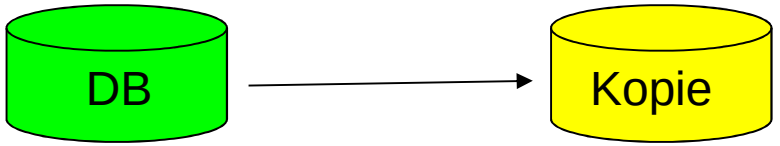
1. **Ausschreiben vor EOT**
   - bei Commit ist nichts zu machen, es müssen aber alle Primärdatensätze geschrieben sein
   - BFIM und AFIM während der TA auf Sekundärspeicher
   - bei Rollback sind die BFIM-Daten einzuspielen
2. **Ausschreiben bei EOT**
   - beim Commit ist zu schreiben
   - BFIM und AFIM während der TA im Hauptspeicher
   - bei Rollback ist nichts zu machen, nur Buffer-Daten löschen

Spezialfall: FAST COMMIT Bei Commit nur AFIM/BFIM, Daten verzögert.

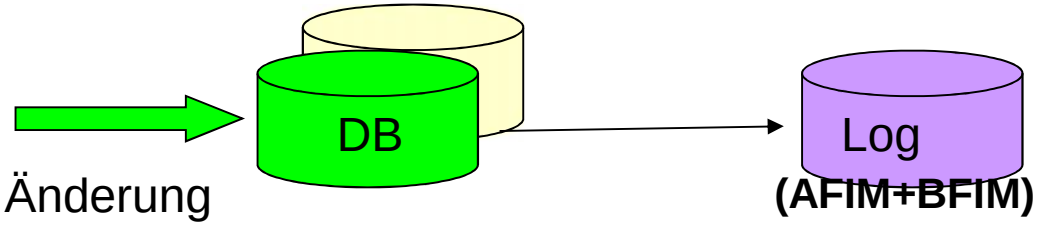
Ablauf von Backup und Recovery

Periodisch im laufenden Betrieb

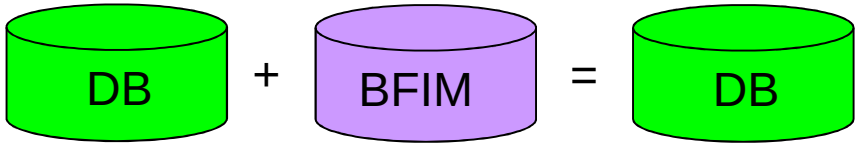
Auch durch Einspielen der AFIM im Laufenden Betrieb möglich



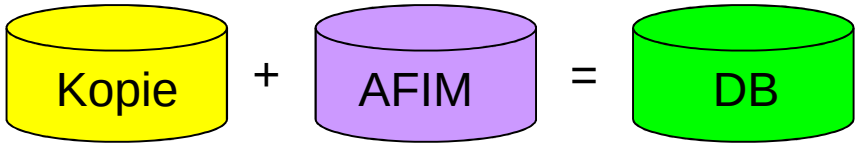
Bei Änderungsoperationen



Zurücksetzen einer TA



Rekonstruktion der DB



Flashback

Rückkehr zu einem historischen Datenbestand, Abfrage historischer Daten.

- Flashback Query
- Flashback Table
- Flashback Drop
- Flashback Database

<u>Voraussetzung</u>:

- Automatic Undo Management
- UNDO tablespace genügend groß
- Recycle bin (für flashback drop)

Flashback Query

Flashback Query: **SELECT <column_list>**
FROM <table_name>
AS OF TIMESTAMP <timestamp>;

```
SELECT * FROM dept
AS OF TIMESTAMP to_timestamp('02-03-2013 07:00:00',
                             'dd-mm-yyyy hh24:mi:ss');

DEPTNO DNAME          LOCATION
------ -------------- -------------
10     ACCOUNTING     NEW YORK
20     RESEARCH       DALLAS
30     SALES          CHICAGO
40     OPERATIONS     BOSTON
50     IT             FRANKFURT
```

Flashback Table

```
FLASHBACK TABLE dept
TO TIMESTAMP to_timestamp('02.03.2013 09:30');

FLASHBACK TABLE dept to BEFORE DROP;
```

Anzeige der Objekte im Recycle Bin

```
DROP TABLE dept PURGE; -- Löscht die Tabelle endgültig

PURGE RECYCLEBIN; -- Leert den Recycle Bin
```

```
show recyclebin
ORIGINAL RECYCLEBIN NAME     OBJECT TYPE  DROP TIME
-------- ------------------- ------------ --------------------
DEPT     RB$$48444$TABLE$0 TABLE        2013-03-02 10:29:40
```

Flashback Database

Herunterfahren der Datenbank

```
SHUTDOWN IMMEDIATE;
```

Hochfahren in den Mount Modus

```
STARTUP MOUNT;
```

Flashback absetzen

```
FLASHBACK DATABASE TO TIMESTAMP to_timestamp
('02.03.2013 09:30:00');
```

Datenbank öffnen

```
ALTER DATABASE OPEN RESETLOGS;
```

SQL Injection

- Webanwendungen bestehen selten nur aus einem System
- Übertragung von Daten + Steuerinformationen an Subsysteme; hier: Subsystem Datenbank
- Metazeichen wie " ’ ; \ - etc. können Verhalten beeinflussen
- SQL Injection: Modifikation eines Wertes, um zusätzliche Steuerinformationen zu enthalten
- Grund für SQL Injection: mangelhafte/fehlende Typ-/Inhaltsprüfung vor Weitergabe an Subsystem
- Auswirkungen: Je nach Inkompetenz von Entwickler und Administrator verheerend

Vgl. hierzu: C. Kunz, P. Prochaska:
Applikationssicherheit in webbasierten Systemen

SQL Injection - Beispiel

- Login-Formular auf Webseite
- Benutzer wird aufgefordert, sich mit Benutzername und Passwort zu authentifizieren
- Resultierende SQL-Anfrage: `SELECT * FROM users WHERE user='"$_GET['user']"' and pass='"$_GET['password'];`
- Angreifer gibt als Benutzer `"john' --"` an
- Login direkt möglich, -- ist Kommentarzeichen
- Weitere beliebte Möglichkeit: `john ' OR 1=1`
- Im Suchfeld: `a' UNION SELECT user, pass FROM admins`

Vgl. Kreisfeuerwehrverband xyz

SQL Injection mit UNION

- Spaltenanzahl und Typ der Spalten müssen bekannt sein
- Verknüpfungsfunktionen erleichtern Angriff
- `SELECT name, strasse, ort FROM users WHERE id=0 UNION SELECT user,pass,null FROM admins`
- `SELECT name FROM users WHERE id=0 UNION SELECT concat(name,pass) FROM admins`

Sicherstellung der Verfügbarkeit

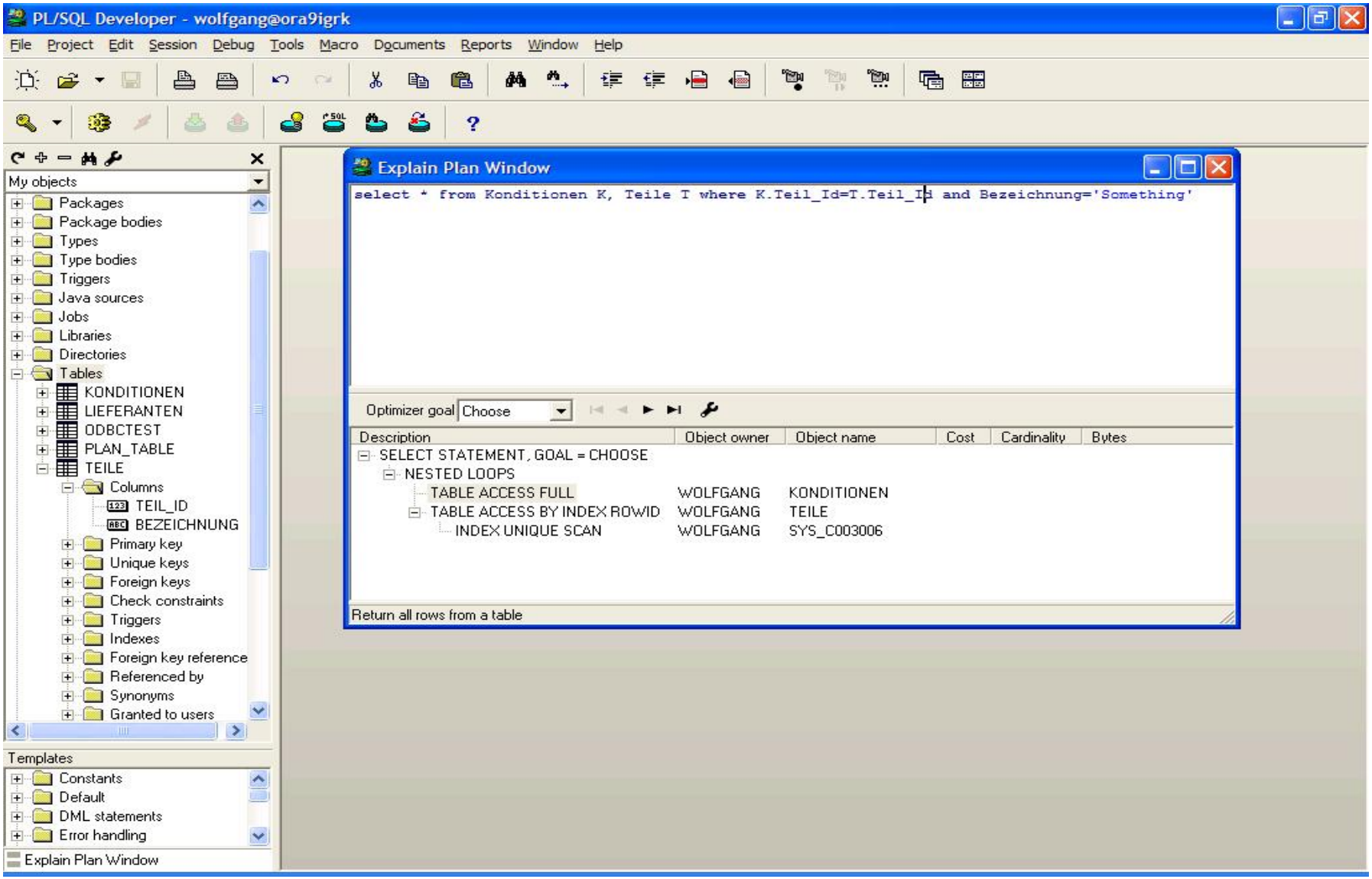
Dazu gehört:

1. Backup-Systeme (cold/hot stand-by)
2. Replikationen
3. Systemoptimierung
   - Reorganisationen
   - Optimierung $\Rightarrow$ explain plan

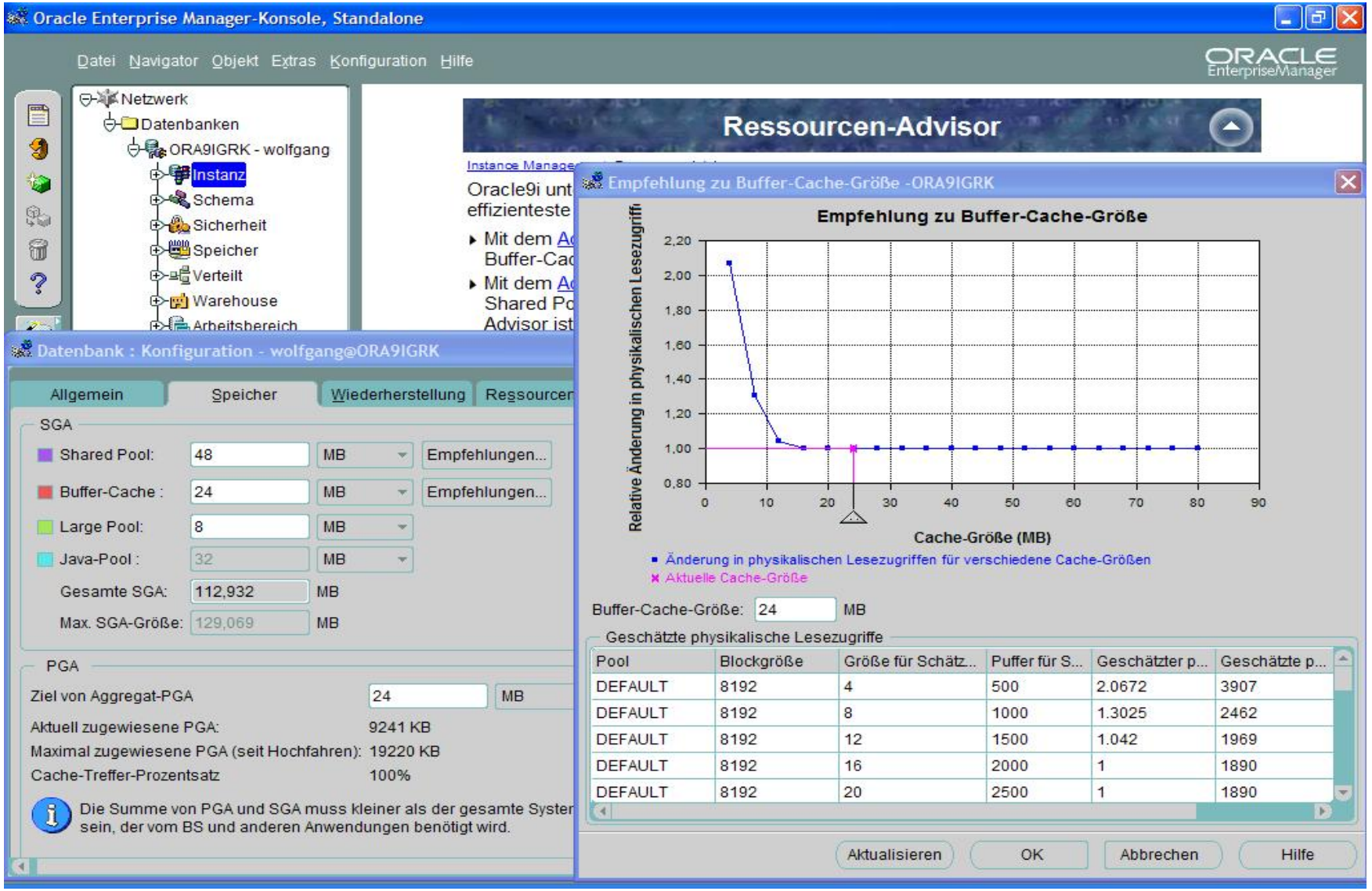
eigenes Kapitel

4. Service Level Agreements
5. Systemadministration über Management-Konsole

Beispiel für explain plan



Oracle Enterprise Management Console



Datenbankreorganisation

Umspeicherung von Datenbanksätzen, um einen effizienteren Ablauf der Zugriffsalgorithmen zu erreichen. Hierdurch werden Speicherplatz gespart und die Antwortzeiten verkürzt.

Objekte für eine Reorganisation:

- Zugriffspfade
- Blocküberläufe
- Verkettete Listen

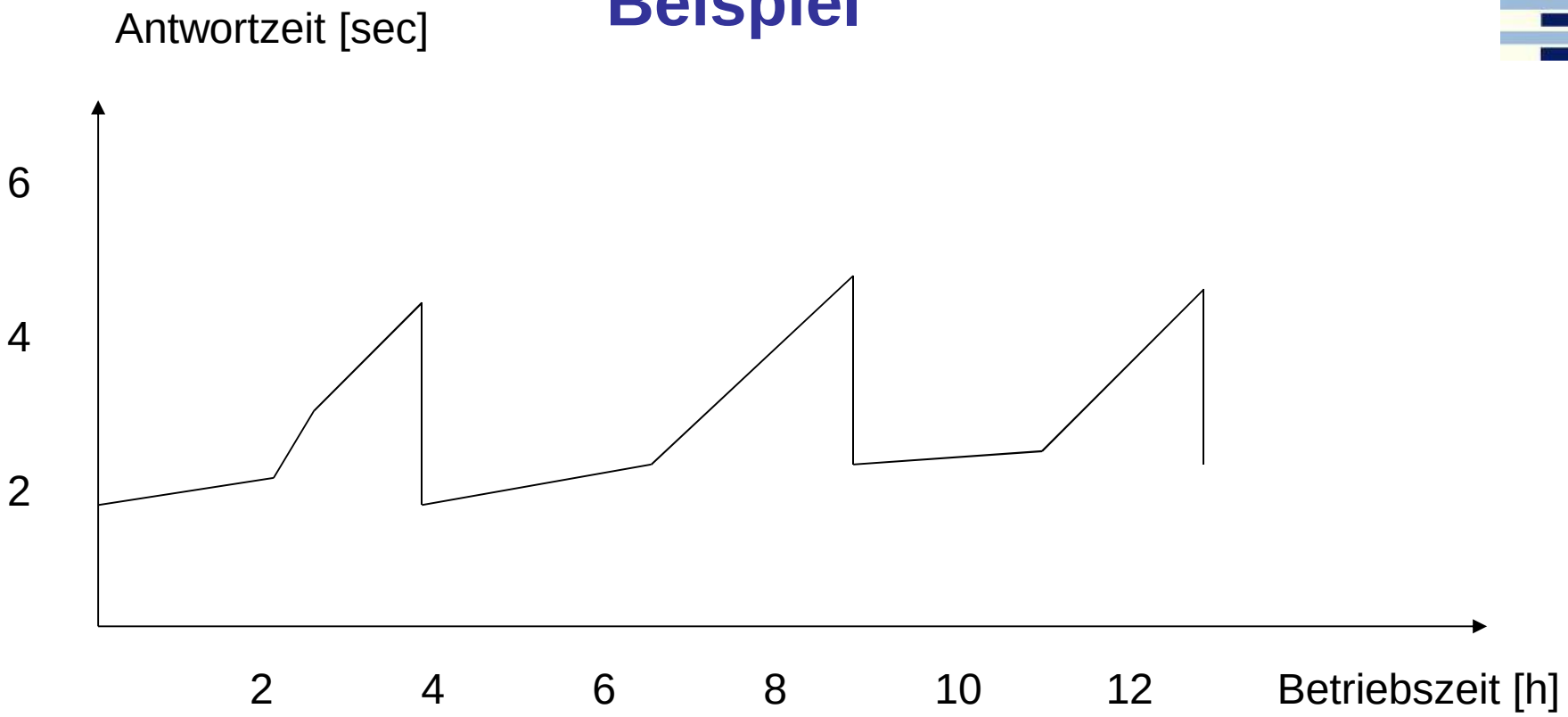
Ursachen für eine Reorganisation:

- Speicherung neuer Datensätze
- Änderungen bestehender Datensätze
  (Längenänderung, Update indizierter Attributwerte)
- Löschung von Datensätzen

Reorganisationsstrategien:

- direkte Reorganisation (in place)
- Entladen und Laden
- schrittweise Reorganisation
  (ausgelöst durch Benutzerzugriff auf die Datenobjekte)
- Parallelität von Reorganisation und Datenbankbenutzung

Beispiel



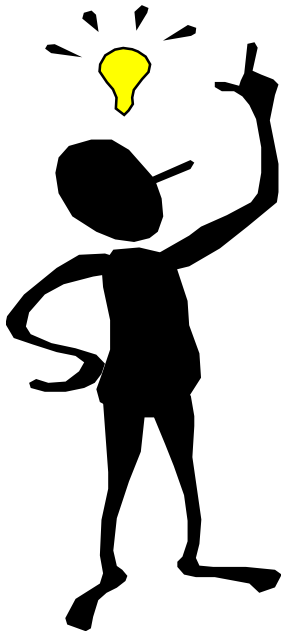
Ausgangssituation: DBMS Oracle, Relation mit 500 Tupeln mit 22 Attributen

Änderungen, Einfügungen, Löschungen im Verhältnis 10:1:1

Ergebnis: Hauptursache der Verschlechterung des Antwortzeitverhaltens ist die zunehmende Verkettung von Datenblöcken.

Und wer macht das alles?

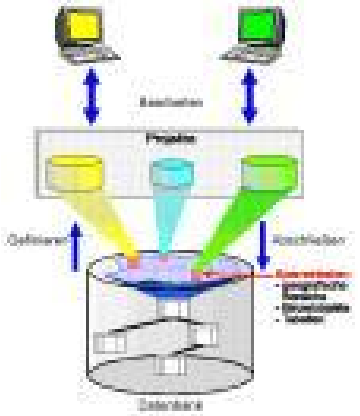
- Systemauswertungen
- Datensicherungsmaßnahmen,
- Recovery-Konzept
- Verwaltung von Benutzerrechten usw.
- Strukturänderungen
- Reorganisationen
- Optimierung, Tuning
- Versionsänderungen (Releasewechsel)
- Schulungen
- …



Der Datenbankadministrator

Transaktionen und Mehrbenutzerbetrieb

- Transaktionen (Wiederholung)
- Fehlermöglichkeiten im Mehrbenutzerbetrieb
- 2-Phasen Sperrprotokoll
- Konsistenzebenen
- Serialisierbarkeit



Transaktionen

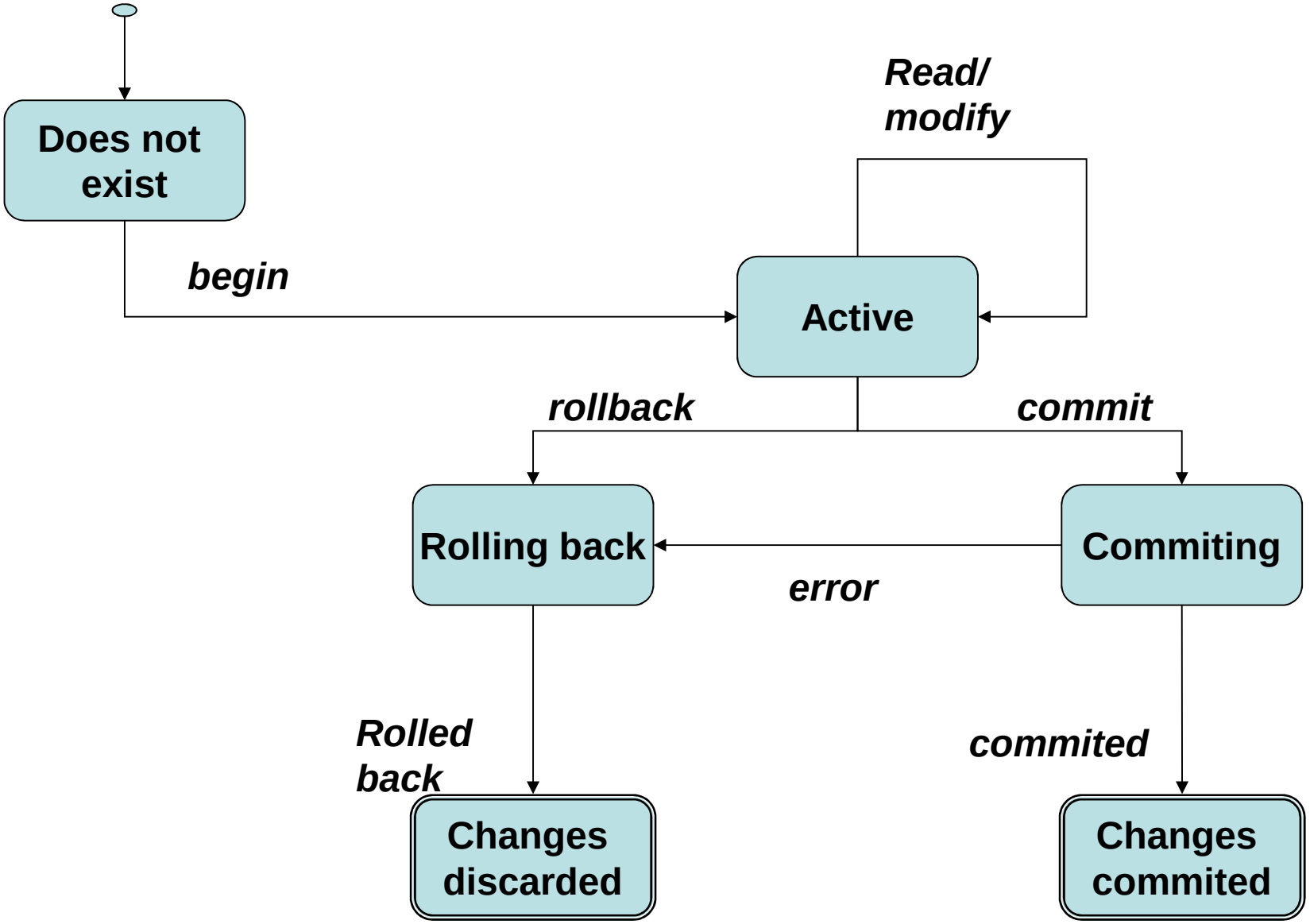
Der Zugriff der Benutzer auf eine Datenbank erfolgt in Form von Transaktionen. Eine **Transaktion** ist eine Folge von Operationen, welche eine Datenbank in ununterbrechbarer Weise von einem konsistenten Zustand in einen (nicht notwendigerweise unterschiedlichen) konsistenten Zustand überführt.

Sie ist gekennzeichnet durch:

1. **Ununterbrechbarkeit (atomicity)**
2. **Konsistenzerhaltung (consistency)**
3. **isolierter Ablauf (isolation)**
4. **Dauerhaftigkeit der Ergebnisse (durability)**

Transaktionen sind i.d.R. kurz. Ein Transaktionssystem arbeitet ereignisorientiert; Ereignisse sind unerwartet, unsortiert und unregelmäßig. Spitzenbelastungen sind nicht ausgleichbar.

Zustandsdiagramm bei Transaktionen



Rollback, Commit, Savepoint

Kein explizites *Begin of transaction* bei SQL
End of transaction: Commit, Rollback
Rollback zu Savepoints möglich

Savepoint A;	1. Savepoint
Delete … ;	1. DML-Befehl
Savepoint B;	2. Savepoint
Insert into … ;	2. DML-Befehl
Savepoint C;	3. Savepoint
Update … ;	3. DML-Befehl
Rollback to C;	Update wird rückgängig gemacht, Savepoint C bleibt definiert
Rollback to B;	Insert wird rückgängig gemacht, C nicht mehr def., B bleibt def.
Rollback to C;	ORA-01086 ERROR; SAVEPOINT C NO LONGER DEFINED
Insert into … ;	Neuer DML-Befehl
Commit;	Gilt für Delete und 2. Insert; alle anderen Befehle (1. Insert und Update) wurden zurückgesetzt

Fehlermöglichkeiten im Mehrbenutzerbetrieb Problem der Nebenläufigkeit, concurrency

Im Mehrbenutzerbetrieb sind besondere Maßnahmen zur Konsistenzerhaltung und Synchronisation beim Zugriff auf Datenbanken notwendig. **Fehlermöglichkeiten**:

1. **Lost Update**
2. **Inkonsistenz der Daten (dirty Update)**
3. **Inkonsistente Sicht auf die Daten (non-repeatable Read)**
4. **Phantome**
5. **Zurücksetzen von Transaktionen (dirty Read)**

Synchronisation paralleler Transaktionen

Die parallele Abarbeitung von Transaktionen muss Regeln unterworfen werden. Das Ergebnis muss so sein, als ob die Transaktionen nacheinander (d.h. seriell) ausgeführt würden.

Lost update

1. Transaktion	2. Transaktion
X:=X+10	**X:=X+5**

T1: liest X
T2: liest X
T1: X:=X+10
schreibt X zurück
T2: X:=X+5
schreibt X zurück

Transaktionen müssen vor Änderungen Sperren setzen und die Sperren anderer Transaktionen beachten.

T1: sperrt und liest X
** T2 erhält keinen Zugriff auf X, wartet
T1: X:=X+10
schreibt X zurück und gibt X wieder frei
T2: sperrt und liest X
X:=X+5
schreibt X zurück und gibt X wieder frei

Inkonsistenz der Daten, dirty update

T1: sperrt und liest X
X:=X*2 schreibt X zurück und gibt X frei
T2: sperrt und liest X und Y
X:=X+10; Y:=Y+10
schreibt X und Y zurück, gibt X und Y wieder frei
T1: sperrt und liest Y
Y:=Y*2 schreibt Y zurück und gibt Y frei

Das Setzen/Freigeben von Schreib-Sperren erfolgt zweiphasig

T1: sperrt und liest X
X:=X*2
schreibt X zurück, gibt X noch nicht frei
** T2 erhält keinen Zugriff auf X, wartet
T1: sperrt und liest Y
Y:=Y*2, schreibt Y zurück
gibt X und Y frei
T2: sperrt und liest X und Y
X:=X+10; Y:=Y+10
schreibt X und Y zurück
gibt X und Y frei

1. Transaktion	2. Transaktion
X:=X*2	**X:=X+10**
Y:=Y*2	**Y:=Y+10**

Inkonsistente Sicht auf die Daten, non-repeatable Read

1. Transaktion	2. Transaktion
Prüft, ob X=Y	**X:=X+10** **Y:=Y+10**

T1: liest X
T2: sperrt und liest X
X:=X+10
sperrt und liest Y
Y:=Y+10
schreibt X und Y, gibt X und Y wieder frei
T1: liest Y
IF X <> Y THEN Fehler ELSE ...

Auch Lesetransaktionen setzen/beachten Sperren

T1: sperrt und liest X (shared)
** T2 bekommt keine Zugriffsberechtigung für X
T1: sperrt und liest Y (shared)
IF X <> Y THEN ... ELSE ok
gibt X und Y wieder frei
T2: sperrt und liest X (exclusive), sperrt und liest Y (exclusive)
X:=X+10, Y:=Y+10
schreibt X und Y,
gibt X und Y wieder frei

Phantome

T1: sperrt und liest alle Objekte X(1) bis X(N) (shared)
Ermittelt N=Anzahl der X(1) bis X(N) // count (*)
gibt alle X(i) wieder frei // nur kurze Lesesperren
T2: speichert ein neues X(i)
T1: sperrt und liest alle Objekte X(1) bis X(N) (exclusive)
setzt X(i):=X(i)+1000/N für alle X(i)
gibt alle X(i) wieder frei

Auch das Setzen von Lesesperren erfolgt zweiphasig.

T1: sperrt und liest das ganze Objekt X(.) (shared)
Ermittelt N=Anzahl der X(1) bis X(N)
** Sperranforderung von T2 für Insert in X nicht erfüllbar
T1: erweitert Shared-Sperren zu Exclusive-Sperren
liest alle Objekte X(1) bis X(N) und X(i):=X(i)+1000/N für alle X(i)
Freigabe von X(1) bis X(N)
T2: speichert neues X(i)

1. Transaktion	2. Transaktion
Berechnet N=COUNT(X)	**Speichert ein neues X(i)**
Erhöht alle X(i) um 1000/N	

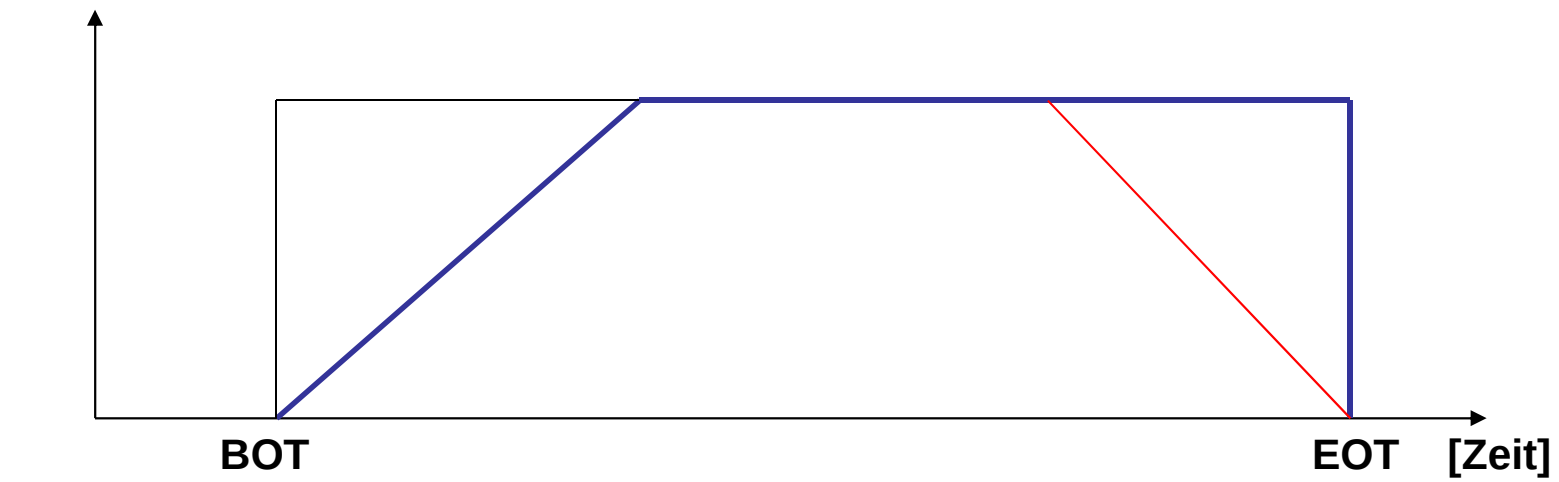
Zurücksetzen von Transaktionen, dirty Read

T1: sperrt und liest X (exclusive)
sperrt und liest Y (exclusive)
X:=X+10
schreibt X zurück
gibt X wieder frei
T2: sperrt und liest X (exclusive)
ändert X
schreibt X und gibt X wieder frei
T1: erkennt Fehler bei Verarbeitung von Y
Zurücksetzen von T1 (Rollback)

Sperren bis zum Transaktionsende (EOT) halten

T1: sperrt und liest X (exclusive)
sperrt und liest Y (exclusive)
X:=X+10
erkennt Fehler bei der Verarbeitung von Y
Zurücksetzen von T1 (Rollback)
T2: sperrt und liest X (exclusive)
ändert X
schreibt X und gibt X wieder frei

Das 2-Phasen-Sperrprotokoll



Zusammenfassung

1. Datenobjekte sind vor der Verarbeitung zu sperren
2. Es gibt Lese- und Änderungssperren (shared und exclusive)
3. Es gibt logische Sperren bzw. unterschiedliche Aggregationsebenen für Sperrobjekte
4. Es wird das 2-Phasen-Sperrprotokoll eingehalten
5. Sperren bleiben bis EOT bestehen (striktes 2-Phasen- Sperrprotokoll)
6. Deadlock-Situationen möglich

Zusammenfassung

Lost Update	T1: read(x)	T2: write(x)	T1: write(x)	
Dirty Update	T1: write(x)	T2: write(y)	T2: write(x)	T1: write(y)
Non-repeatable Read	T1: read(x)	T2: write(x)	T1: read(x)	
Phantome	T1: read(x)	T2: write(x)	T1: read(x)	
Dirty Read	T1: write(x)	T2: read(x)	T1: Rollback	

Arten von Sperren

1. **S-Sperre** Lesesperre
2. **X-Sperre** Schreibsperre
3. **Hierarchische Sperren** (Sperren auf hierarchischen Objekten):
   Attribut, Tupel, Tabelle, Datenbank $\Rightarrow$ ***Absichtssperren*** notwendig (Intension Lock)

   IS-Sperre: Absicht, ein untergeordnetes Objekt zu lesen
   IX-Sperre: Absicht, ein untergeordnetes Objekt zu ändern
   SIX-Sperre: S-Sperre und die Absicht, ein untergeordnetes Objekt zu ändern

Verträglichkeitstabelle

Sperr-wunsch	keine Sperre vorhanden	IS-Sperre vorhanden	S-Sperre vorhanden	SIX-Sperre vorhanden	IX-Sperre vorhanden	X-Sperre vorhanden
IS	ok	ok	ok	ok	ok	-
S	ok	ok	ok	-	-	-
SIX	ok	ok	-	-	-	-
IX	ok	ok	-	-	ok	-
X	ok	-	-	-	-	-

Konsistenzstufen nach GRAY

Konsistenz-stufe	Art der Sperren	Mögliche Fehler
0	Kurze Schreibsperren	Dirty write, lost update*, dirty read, non-repeatable read, Phantome
1	Lange Schreibsperren	lost update*, dirty read, non-repeatable read, Phantome
2	Lange Schreibsperren und kurze Lesesperren	non-repeatable read, Phantome, lost update*
3	Lange Schreibsperren und lange Lesesperren	Phantome

* wenn Lesen und nachfolgendes Ändern 2 Befehle sind (z. B. bei embedded SQL mit Cursor)

Strengere Variante von Konsistenzstufe 2: **Cursor Stability**
Lesesperre bleibt auf einem über einen Cursor angesprochenen Satz solange gesetzt, bis der Cursor auf den nächsten Satz wechselt.

Konsistenzebenen in SQL 2

Sperren beeinträchtigen die Parallelität der Zugriffe und damit die Performance.
In SQL ist Konsistenzebene (Isolation level) 3 Standard, muss implementiert sein. Kann mit SET TRANSACTION ... geändert werden.
Read Uncommitted ist nur für Read-only Transaktionen möglich.

Konsistenz-ebene	Lost Update	Dirty Update/Read	Non Repeat-able Read	Phantome
Read Uncommitted	nicht möglich	möglich	möglich	möglich
Read Committed	nicht möglich	nicht möglich	möglich	möglich
Repeatable Read	nicht möglich	nicht möglich	nicht möglich	möglich
Serializable	nicht möglich	nicht möglich	nicht möglich	nicht möglich

Read Committed: Lese-Konsistenz auf Statement-Level
→ ***default für Oracle***

Serializable: Lese-Konsistenz auf Transaktions-Level

Serialisierbarkeit

Ein System paralleler Transaktionen ist genau dann korrekt synchronisiert, wenn es serialisierbar ist, d.h. wenn es mindestens eine (gedachte) serielle Ausführung derselben Transaktionen gibt, die

1. denselben Datenbankzustand
2. dieselben Ausgabedaten der Transaktionen liefern.

<u>Methoden zur Gewährleistung der Serialisierbarkeit</u>

1. Anwendung eines Verfahrens, das Nicht-Serialisierbarkeit gar nicht erst entstehen lässt (präventive Synchronisationsverfahren); hierzu gehört das 2-Phasen-Sperrprotokoll.
2. Beobachtung der Transaktionsabläufe, ob eine Situation entsteht, die Nicht-Serialisierbarkeit anzeigt; ggfs. Zurücksetzen einer geeigneten Transaktion

Beispiele zur Serialisierbarkeit

T1:	T2:
read a	read b
a:=a+10	b:=b+10
write a	write b
read b	read c
b:=b-10	c:=c-10
write b	write c

Ergebnis der seriellen Ausführung:
A:=A+10;
B bleibt
C:=C-10

Ablauf 1

T1	**T2**
read a	
	read b
a:=a+10	
	b:=b+10
write a	
	write b
read b	
	read c
b:=b-10	
	c:=c-10
write b	
	write c

Ablauf 2

T1	**T2**
read a	
	read b
a:=a+10	
write a	
	b:=b+10
read b	
	write b
b:=b-10	
	read c
write b	
	c:=c-10
	write c

Serialisierbarkeitsbedingung (1)

Betrachten wir die Folge der Read- und Write-Operationen von Transaktionen T1... Tn auf einem Objekt a der Datenbank und bezeichnen wir diese mit Log(a).

Definition: Ein System paralleler Transaktionen ist genau dann **seriell**, wenn es eine totale Ordnung S der Transaktionen gibt, so dass gilt: Ist Transaktion Ti vor Tj bzgl. S, so sind die Operationen von Ti in jedem Log (in dem Operationen von Ti und Tj auftreten) vor den Operationen von Tj .

Mit anderen Worten: Die Operationen werden auf jedem Objekt in gleicher Transaktions-Reihenfolge ausgeführt.

Serialisierbarkeitsbedingung

Definition: Zwei Operationen OP(Ti) und OP(Tj) verschiedener Transaktionen stehen in Konflikt zueinander, wenn sie auf dasselbe Objekt zugreifen und mindestens eine der Operationen ein *write* ist.
Ein System T paralleler Transaktionen ist serialisierbar, wenn die Reihenfolge der Operationen, die in Konflikt zueinander stehen, in allen Logs dieselbe ist.

Definition: Enthalte das System SpT paralleler Transaktionen die Transaktionen {T1, ..., Tn} und sei {Log(o1), ..., Log(om)} die Menge der Logs derjenigen Objekte {o1, ... om} auf die die Transaktionen zugreifen. SpT ist dann **serialisierbar**, wenn eine totale Ordnung S für T existiert, so dass gilt:
Für jedes Paar (Ti, Tj) mit Ti vor Tj bzgl. S ist in jedem Log OP(Ti) vor OP(Tj), wenn OP(Ti) zu OP(Tj) in Konflikt steht.

Beispiel für Konflikte

Log(a) Ti: read a
Tj: read a
Ti: write a
Tj: write a
⇒ nicht serialisierbar

Log(a) Ti: read a
Tj: write a

Log(b) Tj: write b
Ti: read b

Reihenfolge der Operationen von Ti und Tj in Log(a) und Log(b) verschieden; in beiden Logs stehen die Operationen in Konflikt zueinander
⇒ nicht serialisierbar.

… und Serialisierbarkeit

(vgl. Ablauf 1 und 2 des Beispiels):

Ablauf 1, 2	Ablauf 1	Ablauf 2	Ablauf 1, 2
Log(a)	**Log(b)**	**Log(b)**	**Log(c)**
T1: read	T2: read	T2: read	T2: read
T1: write	T2: write	T1: read	T2: write
	T1: read	T2: write	
	T1: write	T1: write	

Im Log(b) gilt bei Ablauf 1:

(T2: read) vor (T1: write) und (T2: write) vor (T1: write)

$\Rightarrow$ T1 und T2 serialisierbar

In Log(b) gilt bei Ablauf 2:

(T1: read) vor (T2: write) und (T2: write) vor (T1: write)

$\Rightarrow$ T1, T2 nicht serialisierbar

Sperrverfahren bei Oracle

- Keine Lesesperren
- Lesende Transaktionen sehen konsistenten DB-Zustand wie beim TA-Start
- Transaktions-IDs mit Zeitmarken; Eintragung der TIDs in die Segment-Header

Update Personal Set Gehalt=Gehalt*1.1 where Abteilung='DV'
Während des Updates: Select * from Personal

Datensegmente ***Rollback-Segmente***

Name	Gehalt	Abteilung
Peter		
Anna	3300	DV
Klaus		
Regine		
Josef	4000	DV
Jutta		

Anna	3000	DV

bereits geändert

Noch nicht geändert

Read Committed vs. Serializable

Operation	Read Committed	Serializable
Dirty write, dirty read	nicht möglich	nicht möglich
Non-repeatable read	möglich	nicht möglich
Phantome	möglich	nicht möglich
Verträglich mit ISO SQL99	Ja	ja
Konsistenzniveau	Statement-Ebene	Transaktions-Ebene
Sperren auf Row-level Ebene	ja	ja
Lese-TA's blockieren Schreib-TA's *	nein	nein
Schreib-TA's blockieren Lese-TA's *	nein	nein
Schreiben anderer Tupel möglich	ja	ja
Warten von/auf Transaktionen	Ja	ja
„Can't serialize access"-Fehler möglich	nein	ja

* Oracle-spezifisch wegen des Lesens der Rollback-Segmente durch Lesetransaktionen

Explizite Sperren bei Oracle

Änderung der Konsistenz-Ebene (Isolation level) durch:

1. Lock table …
2. Select … from ... for update
3. Set transaction to {read-only | read-committed | serializable }

Row Share Table Lock	Lock Table < t > in row share mode Select … from < t > for update of …
Row Exclusive Table Lock	Lock Table < t > in row exclusive mode Insert into < t > … Update < t > … Delete from < t > …
Share Table Lock	Lock Table < t > in share mode
Exclusive Table Lock	Lock Table < t > in exclusive mode

Wer die Wahl hat, hat die Qual

ROW SHARE Mode
Höchster Grad der Parallelität, andere Datensätze können gelesen, geändert oder eingefügt werden; Updates nur möglich, wenn keine anderen Update-TA's

SHARE Mode
Nur für Lesetransaktionen, Serializable

ROW EXCLUSIVE Mode
Transaction-level Leseskonsistenz mit der Möglichkeit zum Ändern; nur jeweils 1 Transaktion mit diesem Mode zurzeit möglich; andere Transaktionen mit Select … for Update müssen warten.

EXCLUSIVE Mode
Transaction-level Lese- und Updatekonsistenz; keine weiteren TA's möglich

Beispiel

Abteilung	AbtNr	Bezeichnung	Ort

Mitarbeiter	PersNr	Name	AbtNr	Gehalt

Budget	AbtNr	Gehaltssumme

```
Lock table Abteilung in SHARE Mode;
Update Mitarbeiter set Gehalt = Gehalt * 1.1 where AbtNr
in(Select AbtNr from Abteilung where Ort = 'Hamburg');
```

/* hier kann jetzt keine Änderung in Abteilung vorgenommen werden */

```
Update Budget set Gehaltssumme = Gehaltssumme * 1.1
where AbtNr in
(Select AbtNr from Abteilung where Ort = 'Hamburg');
Commit;
```

Dabei wird davon ausgegangen, dass bei Abteilungen so gut wie nie der Ort geändert wird.