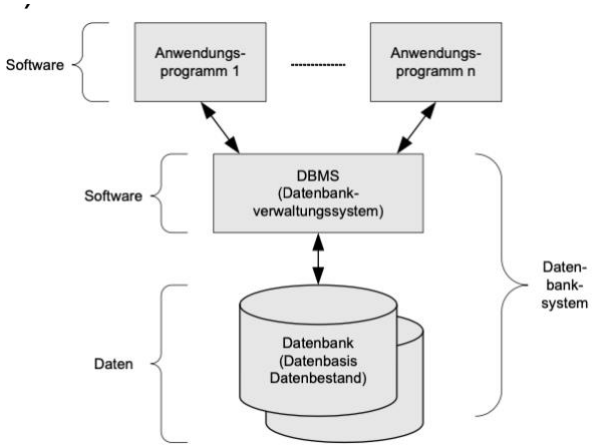
Datenbanksysteme



Datenbanksystem (DBS)

- DBMS (Datenbankverwaltungssystem)
- Datenbank

Informationssystem (IS)

= DBS + Anwendungsprogramme (AP)

Arten von Daten

Strukturierte Daten (relationale DB)

- tabellarisch
- fest vorgegebene Struktur

Unstrukturierte Daten (XML, JSON)

- unformatiert, keine explizite Struktur
- Texte, Bilder (Pixel), Filme

Semi-struktuierte Daten

- teilweise, unvollständige Struktur

3-Ebenen-Architektur

Konzeptionell (logische Ebene)
Datenbankschema, logische Gesamtsicht

Extern (Sichten)
Sicht einer einzelnen Anwendung bzw. Benutzergruppe

Intern (physische Ebene)
Speicherung, Datenorganisation, Physische Darstellung von Daten, Indizes

Logische Datenunabhängigkeit

Ebenso können bei (den meisten) Änderungen des logischen Schemas die externen Schemas unverändert weiterbestehen. Dies ist besonders deshalb erwünscht, weil dadurch Anwendungsprogramme nicht modifiziert oder neu übersetzt werden müssen.

Physische Datenunabhängigkeit

Das logische Schema kann demnach unverändert bleiben, wenn sich beispielsweise aus Gründen der Optimierung oder Reorganisation der Speicherort oder die Speicherform einzelner Daten ändern.

Aufgaben eines DBMS

1. Zentrale Kontrolle über die operationalen Daten
2. Hoher Grad an Datenunabhängigkeit
3. Hohe Leistung und Skalierbarkeit
4. Mächtige Datenmodelle und Abfragesprachen
5. Transaktionskonzepte
6. Ständige Betriebsbereitschaft

- effiziente und flexible Verwaltung grosser Menge persistenter Daten

Vorteile von Datenbanksystemen

- Datenkonsistenz einfacher sicherzustellen
- Mehrere Anwendungen können gleichzeitig auf dieselben Daten zugreifen

Unabhängig von:

- Physischer Datenstruktur «physische Datenunabhängigkeit»
- Erweiterung der Daten «logische Datenunabhängigkeit»

Relationales Datenmodell

Relation => mathematisches Objekt ≠ Tabelle

Begriff	**Beschreibung**	**Beispiel**
Relation	endliche Mengen ⇒ keine doppelten Elemente und sind ungeordnet! Relationsschema + Menge von Tupel	{<1,Meier,19.4.1992>, <2,Müller,23.8.1998>, <3,Huber,11.9.2001>} Studierende(Matrikelnummer,Name,Geburtstag)
Domäne	Datentyp	{red, green, blue}
Tupel	Zeile, Record, Eintrag => setzen sich aus Attributwerten zusammen	<8400, Winterthur, Nord>
Attribut	«Spalte» hat eine Bezeichnung und eine Domäne	Anrede / string[30]
Surrogatsschlüssel	«künstlicher» Schlüssel	
Entität	wie ein Tupel. Wird durch ein Objekt dargestellt	
Entitätstyp	wie eine Klasse. Wird durch eine Relation dargestellt.	
Bags	Multimengen	

Datenbankprogrammierung (pgSQL)

Unterschied Primärschlüssel / Unique key

Primärschlüssel:

- In einer Tabelle kann es nur einen Primärschlüssel geben
- In einigen DBMS kann es nicht NULL sein - z. MySQL fügt NOT NULL hinzu
- Primärschlüssel ist eine eindeutige Schlüsselkennung des Datensatzes

Unique key:

- Kann mehr als ein eindeutiger Schlüssel in einer Tabelle sein (1-1 Beziehung)
- Ein eindeutiger Schlüssel kann NULL Werte haben
- Eindeutiger Schlüssel kann NULL sein; Mehrere Zeilen können NULL Werte haben und werden daher möglicherweise nicht als "eindeutig" betrachtet.

Stored procedures (explizit)

```
1.  - CREATE {TRIGGER|PROCEDURE|FUNCTION} Test ...
2.  - ALTER {TRIGGER|PROCEDURE|FUNCTION} Test ...
3.  - DROP {TRIGGER|PROCEDURE|FUNCTION} Test
```

```
DECLARE
  -- Deklarationsblock
  -- Der DECLARE Abschnitt ist optional
BEGIN
  -- Ausführungsteil
EXCEPTION
  -- Ausnahmeverarbeitung
  -- Der EXCEPTION Abschnitt ist optional
END;
```

Beispiel

```
1.  CREATE OR REPLACE FUNCTION IsCHPLZ(PLZ varchar(10))
2.    RETURNS bit
3.  AS $$
4.  BEGIN
5.    -- Prüfen, ob vierstellig numerisch
6.    IF PLZ NOT SIMILAR TO '%[0-9][0-9][0-9][0-9]%' THEN
7.      RETURN 0;
8.    END IF;
9.
10.   -- Prüfen, ob in der Referenztabelle vorhanden
11.   IF PLZ::integer IN (SELECT DISTINCT p.PLZ FROM PLZSchweiz p) THEN
12.    RETURN 1;
13.   END IF;
14.
15.   RETURN 0;
16. END; $$
17. LANGUAGE 'plpgsql';
18.
19. -- Testfälle
20.
21. SELECT IsCHPLZ('ac235');
22.
```

Trigger (implizit)

- Wird ausgelöst aufgrund von sich änderenden Datenbankzuständen:
  - INSERT, UPDATE, DELETE
- hängen an einer Tabelle

Auslösung

- BEFORE
- AFTER

Beispiele:

- Nachführen von Summen
- Realisieren von Integritätsbedingungen

Beispiel für einen Trigger

```
1.  CREATE OR REPLACE FUNCTION IsAdresseOK() RETURNS TRIGGER AS $IsAdresseOK$
2.  DECLARE
3.    RetCode integer;
4.  BEGIN
5.    IF (NEW.LandCode <> 'CH') AND (NEW.LandCode <> 'LI') THEN
6.      RETURN NEW; -- keine Prüfung nötig;
7.    END IF;
8.
9.    -- CH und LI: PLZ prüfen
10.   SELECT IsCHPLZ(NEW.PLZ) INTO RetCode;
11.
12.   IF RetCode = 0 THEN
13.     RAISE EXCEPTION 'Postleitzahl mussen numerisch und vierstellig und in der Referenztabelle
    vorhanden sein!'; -- nur zu Anschauungszwecken
14.   END IF;
15.
16.   RETURN NEW;
17. END; $IsAdresseOK$
18. LANGUAGE 'plpgsql';
19.
20. CREATE TRIGGER IsAdresseOK
21. BEFORE INSERT OR UPDATE ON Adressen
22. FOR EACH ROW EXECUTE PROCEDURE IsAdresseOK();
```

Stored procedures vs Triggers

Stored procedures

- Aufruf (explizit):
  - Durch Benutzer oder Anwendungsprogramm
  - Transaktionskontrolle
- Einsatzbereiche:
  - Kapselung von „business rules“
  - Optimierung von Abfragen, Reduktion des Netzwerkverkehrs -> Batch Processing
  - Erhöhte Sicherheit benötigt
  - ...
- Probleme:
  - Fehlerbehandlung

Trigger

- Aufruf (implizit):
  - Durch DBMS, in Abhängigkeit von Datenänderungen
  - Keine Transaktionskontrolle
- Einsatzbereiche:
  - Konsistenzsicherung (referentielle Integrität)
  - Logging
  - Nachführen von Tabellen
  - ...
- Probleme:
  - Kompliziert zu testen
  - Aufrufreihenfolge nicht determiniert

Index

```
1. CREATE INDEX start_year_idx ON title(start_year);
```

Abfragen werden damit schneller, allerdings wird das Einfügen von Daten langsamer, da auch der Index aktualisiert werden muss.

Indexarten

- Index vs. Schlüssel:
  - Index: physische Datenstruktur, um Datensatz schnell zu finden
  - Schlüssel: eindeutige Identifizierung eines Datensatzes (relational Modellierung)
- Primärindex:
  - Zugriffspfad, der die Dateiorganisationsform nutzen kann
  - über Primärschlüssel oder eventuell Schlüsselkandidat (duplikatenfrei)
  - Maximal einer pro Tabelle
- Sekundärindex
  - Jeder Zugriffspfad ohne Nutzung der Dateiorganisationsform
  - Mehrere pro Tabelle
- Primärschlüssel (wichtiger Kandidat für Primär-/Sekundärindex):
  - Echter Schlüssel (d.h. identifizierend, ohne Duplikate)
  - Maximal einer pro Tabelle
- Sekundärschlüssel / Suchschlüssel (Kandidat für Sekundärindex):
  - Nicht zwingend Schlüssel
  - Mehrere pro Tabelle

Wann lohnt sich ein Index?

1. Attribute, die oft abgefragt werden, sollten indiziert werden.
2. Fremdschlüssel sollten indexiert werden, insbesondere dann, wenn über «Primär-Fremdschlüssel» gejoint wird (was häufig der Fall ist).
3. Attribute über die oft gejoint wird; wenn über mehrere Attribute gejoint wird dann muss ein zusammengesetzter Index verwendet werden.
4. Attribute mit niedriger Kardinalität (Extrembeispiele: Geschlecht, Ja/Nein- Flags u.ä.) sollten nicht indexiert werden (es gibt dafür spezielle Indexstrukturen, hier aber nicht behandelt).

Überindexierung kostet Ausführungszeit und Speicherplatz und kann den Optimizer in die Irre führen! Die Mustersuche LIKE `%E` verwendet keinen Index z.B.!

Transaktionen

ACID

1. Atomarität (Atomicity) / Unit of Work:
   Zusammengehörige Folge von Lese- und Schreibzugriffen (in sich geschlossene „Arbeitseinheit“), muss als Ganzes entweder erfolgreich abgeschlossen (Commit) oder rückgängig gemacht werden können (Rollback).
2. Consistency / Konsistenz: Alle Operationen hinterlassen die Datenbank in einem konsistenten Zustand.
3. Isolation / Nebenläufigkeit (Concurrency):
   „Gleichzeitiger“ Zugriff mehrerer Benutzer ermöglichen, so dass diese Transaktionen keinen unerwünschten Einfluss aufeinander haben (bei nur einer CPU auch möglich -> Gleichzeitigkeit wird durch Zeitscheibenverfahren „simuliert“).
4. Dauerhaftigkeit (Durability) / Recovery:
   Automatische Behandlung von Ausnahmesituationen (Fehlern) und schneller (möglichst automatischer) Wiederanlauf nach schwerwiegenden Fehlern. Wiederherstellung (Rollforward) verlorener Daten und Rücksetzten (Rollback) fehlerhafter Daten.

Integrität, Konsistenz

- Regeln, die die Daten einhalten müssen -> Konsistenz

Bereichsintegrität
Wert des Attributs muss in einem gewissen Wertebereich sein

Entitätsintegrität
Primärschlüssel muss eindeutig sein und immer vorhanden sein (NOT NULL)

Referentielle Integrität
Fremdschlüssel muss entweder leer sein oder genau ein Tupel mit einem solchen Schlüsselwert muss in der referenzierten Tabelle vorhanden sein

Transaktionsgrenzen

COMMIT: Transaktionsresultat wird permanent
ROLLBACK ABORT: Annullierung aller Veränderungen
Verlassen der Session ohne Commit ⇒ inkonsistenter Zustand ⇒ Recovery

Probleme konkurrenter Transaktionen

Lost Update

Überschreiben bereits getätigter Updates -> Update-Operation von Benutzer 1 geht verloren!

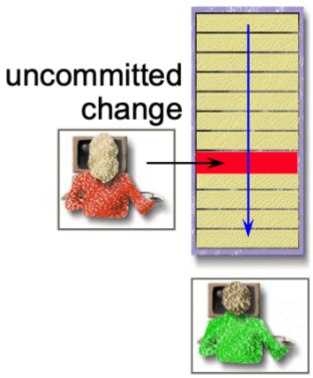
	Transaktion Benutzer 1	Transaktion Benutzer 2
1	SELECT Wert INTO W FROM Tbl	
2		SELECT Wert INTO W FROM Tbl
3	UPDATE Tbl SET Wert = 100	
4		UPDATE Tbl SET Wert = 200
5	SELECT Wert INTO W FROM Tbl	
6		SELECT Wert INTO W FROM Tbl

Isolationslevel Repeatable Read (RR) als Lösung.

Dirty Read

Lesen von Veränderungen noch nicht abgeschlossener Transaktionen

	Transaktion Benutzer 1	Transaktion Benutzer 2
1	SELECT Wert INTO W FROM Tbl	
2	UPDATE Tbl SET Wert = NeuerWert	
3		SELECT Wert INTO W FROM Tbl
4	ROLLBACK	
5		UPDATE Tbl SET Wert = W + 1
6		SELECT Wert INTO W FROM Tbl
7	SELECT Wert INTO W FROM Tbl	



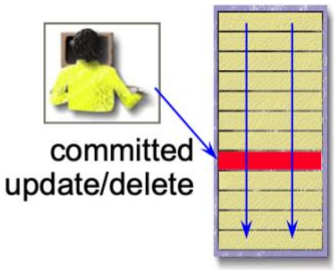
➔ Keine Isolation der Transaktionen beider Benutzer

Lösung: nur Updates bestätigter Transaktionen lesen!

Non-Repeatable-Read

Lesen von zwischenzeitlich von anderen Transaktionen durchgeführten Veränderungen

	Transaktion Benutzer 1	Transaktion Benutzer 2
1	SELECT Wert INTO W FROM Tbl	
2		UPDATE Tbl SET Wert = Wert + 5
3		COMMIT
4	SELECT Wert INTO W FROM Tbl	

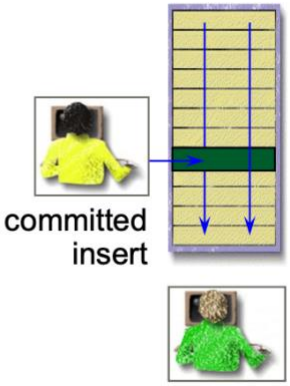


➔ Keine Isolation der Transaktionen beider Benutzer

Phantom Read

Lesen von zwischenzeitlich von anderen Transaktionen durchgeführten Veränderungen

	Transaktion Benutzer 1	Transaktion Benutzer 2
1	SELECT count(*) INTO cnt FROM Tbl	
2	N = cnt	
3		INSERT INTO Tbl VALUES (…)
4		COMMIT
5	SELECT count(*) INTO cnt FROM Tbl	
6		



➔ Keine Isolation der Transaktionen beider Benutzer

Lösung: Nur den Datenbankzustand sehen, der bei Beginn einer Transaktion vorliegt

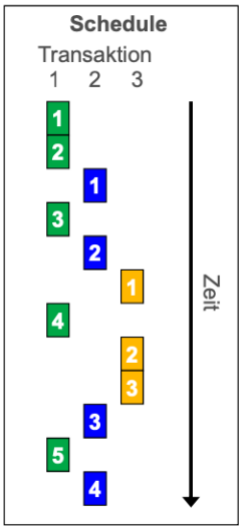
Praxis

- Lost-Update: kann in einem Transaktionssystem fast nie toleriert werden.
- Andere: bei konkurrentem Lesen oft zu einem gewissen Ausmass tolerierbar

Isolationsebene	Dirty Read	Non-Repeatable Read	Phantom Read	Lost Update
READ UNCOMMITTED	~~möglich~~ (nicht in Postgres)	möglich	möglich	~~möglich~~ (nicht in Postgres)
READ COMMITTED Häufig in der Praxis	verhindert	möglich	möglich	verhindert
REPEATABLE READ	verhindert	verhindert	~~möglich~~ (nicht in Postgres)	verhindert
SERIALIZABLE	verhindert	verhindert	verhindert	verhindert

Höherer Isolationslevel (Serializable z.B.) => weniger, konkurrente Transaktionen

Schedules



Schedule ist ein Ablaufplan. Also eine Folge von Lese- bzw. Schreiboperationen für die Ausführung einer oder mehrerer Transaktionen. Können die ACID-Eigenschaften verletzen.

Eine vollständiger Schedule sind sämtliche Schritte aller anstehenden Transaktionen inklusive Terminierung. Für jede Transaktion ist festgehalten, ob sie erfolgreich endet oder abbricht.

Bei einem seriellen Schedule kommen alle Transaktionen nacheinander. Serielle Schedules werden als konsistenzerhaltend betrachtet. Der Scheduler ist der Transaktionsmanager. Er erzeugt einen serialisierbaren Ablaufplan für parallel auszuführende Transaktionen. Er verwaltet auch notwendige Synchronisationsinformationen, z.B. die Lese- und Schreibsperren.

<u>Serialisierbarer Schedule ⇒ keine Zyklen</u>

Scheduling-Verfahren:

1. Ein Scheduler arbeitet aggressiv, wenn er Konflikte zulässt und dann versucht, aufgetretene Konflikte zu erkennen und aufzulösen:
   - Ziel: Maximiert die Parallelität von Transaktionen
   - Risiko: Transaktionen werden erst am Ende ihrer Ausführung zurückgesetzt
   - Grenzfall: Im Extremfall ist keine Transaktion mehr erfolgreich
   - Beispiel: Postgres, Oracle («Multiversion Concurrency Control»)
2. Ein Scheduler arbeitet konservativ, wenn er Konflikte möglichst vermeidet, dafür aber Verzögerungen von Transaktionen in Kauf nimmt:
   - Ziel: Minimiert den Rücksetzungsaufwand für abgebrochene Transaktionen
   - Risiko: Erlauben eine geringere Parallelität von Transaktionen
   - Grenzfall: Im Extremfall findet keine Parallelisierung von Transaktionen mehr statt, d.h. die Transaktionen werden sequentiell ausgeführt
   - Beispiel: SQL Server (Scheduling-Verfahren: Sperrverfahren; nutzt 7 Haupt-Lock-Modi und diverse Untermodi zur Optimierung)

Sperrverfahren

Blocking

Eine gesperrte Ressource zwingt andere Prozesse zu warten, bis diese wieder freigegeben wird → Reduktion des Durchsatzes an Transaktionen.

Livelock (Verhungern)

Eine Transaktion kommt nie dran, weil immer wieder andere vorher berücksichtigt werden.

Deadlock (Verklemmung)

Eine Menge von Transaktionen sperren sich gegenseitig, wenn jede Transaktion der Menge auf ein Ereignis wartet, das nur durch eine andere Transaktion der Menge ausgelöst werden kann.

⇒ RDBMS macht Zyklen-Suche (”wer wartet auf wen”)
⇒ Objekte immer in derselben Reihenfolge abfragen

Recovery

Alle Massnahmen zur Wiederherstellung verloren gegangener Datenbestände. Der Transaktionsmanager/Scheduler wahrt die Isolation- und Konsistenzeigenschaft einer Transaktion. Der Recovery-Manager sichert die Atomaritäts- und Dauerhaftigkeitseigenschaft einer Transaktion.

1. **Recovery-Manager**: Er sorgt dafür, dass nach einem Fehler alle Änderungen committeter Transaktionen im stabilen Speicher abgelegt werden und keine Änderungen von aktiven oder abgebrochenen Transaktionen im stabilen Speicher verbleiben.
2. **Puffer-Manager:** Verwaltet den Puffer (DB- und Log-Puffer) und holt Daten vom stabilen Speicher in den Puffer, schreibt Daten vom Puffer in den stabilen Speicher und ersetzt Daten im Falle eines "Pufferüberlaufs".

Es gibt 3 Klassifikationen der Fehler, die nach dem betroffenen Bereich des DBMS auftreten können.

1. **Transaktionsfehler:** Haben den Abbruch der jeweiligen Transaktion zur Folge, keinen Einfluss auf den Speicher des Systems.
   Typische Transaktionsfehler: Fehler im Anwendungsprogramm (z.B. Division durch 0), Transaktionsabbruch durch den Benutzer (z.B. unzulässige Dateneingabe), Transaktionsabbruch durch das System (z.B. Deadlock)
2. **Systemfehler:** Zerstörung der Daten im Hauptspeicher (flüchtige DB), betreffen jedoch nicht den Hintergrundspeicher (permanente DB)
   Typische Systemfehler: DBMS-Fehler (z.B. Konfigurationsfile fehlerhaft), Betriebssystemfehler (z.B. Seitenfehler, ungültiger Befehl), Hardware-Fehler (z.B. Stromausfall)
3. **Mediafehler:** Verlust von Daten der stabilen Datenbank
   Typische Mediafehler: "Head-Crashes", Controller-Fehler, Naturgewalten wie Feuer oder Erdbeben, Operator-Fehler, Fehler im Programm

Logging

Zur Wiederherstellung muss eine Historie aller Änderungen protokolliert werden.
Das Write Ahead Log-Prinzit (WAL-Prinzip) fordert zwei Regeln, die eingehalten werden müssen, um korrekte Log Einträge zu machen.

1. Vor dem COMMIT einer Transaktion müssen alle zugehörigen Log-Einträge auf stabilen Speicher ausgelagert werden. Diese Regel ist notwendig, um ein REDO durchführen zu können.
2. Vor dem Auslagern einer modifizierten Seite in die persistente Datenbank müssen alle zugehörigen Log-Einträge zur Seite auf stabilen Speicher ausgelagert werden. Diese Regel ermöglicht das UNDO bei abgebrochenen Transaktionen.