

## EINSATZ UND REALISIERUNG VON DATENBANKSYSTEME

Transaktionssysteme	<ul style="list-style-type: none"><li>- <i>Begin of transaction (BOT)</i></li><li>- <i>Commit</i>: alle Änderungen der Datenbasis werden durch diesen Befehl festgeschrieben (erfolgreichen Abschluss der Transaktion)</li><li>- <i>Abort</i>: Datenbanksystem muss sicherstellen, das die Datenbasis wieder in den Zustand zurückgesetzt wird (erfolglosen Abschluss der Transaktion)</li><li>- <i>Define savepoint</i>: DBMS muss sich dazu alle bis zu diesem Zeitpunkt ausgeführten Änderungen an der Datenbasis merken</li><li>- <i>Backup transaction</i>: die noch aktive Transaktion auf den jüngsten Sicherungspunkt zurücksetzen</li></ul> <p><b>Eigenschaften von Transaktionen (ACID)</b></p> <ul style="list-style-type: none"><li>- <i>Atomicity</i>: alle oder nichts</li><li>- <i>Consistency</i>: konsistenter Zustand der DB</li><li>- <i>Isolation</i>: jede Transaktion hat die DB für sich alleine</li><li>- <i>Durability</i>: Änderungen erfolgreicher Transaktionen können nie verloren gehen</li></ul> <p><b>Transaktionsverwaltung in SQL:</b></p> <ul style="list-style-type: none"><li>- <i>Commit work</i>: Änderungen werden festgeschrieben</li><li>- <i>Rollback work</i>: Änderungen sollen zurückgesetzt werden</li></ul>									
Fehlerbehandlung (Recovery)	<ul style="list-style-type: none"><li>- <i>R1 Recovery</i>: lokaler Fehler in einer noch nicht festgeschriebenen Transaktion (muss zurückgesetzt werden)</li><li>- <i>R2 Recovery</i>: Fehler mit Hauptspeicherverlust (Abgeschlossene TA müssen erhalten bleiben)</li><li>- <i>R3 Recovery</i>: Fehler mit Hauptspeicherverlust (Noch nicht abgeschlossene TA müssen zurückgesetzt werden)</li><li>- <i>R4 Recovery</i>: Fehler mit Hintergrundspeicherverlust</li></ul>									
Speicherhierarchie	<p>Ersetzung von Pufferseiten:</p> <ul style="list-style-type: none"><li>- <i>-steal</i>: Ersetzung von Seiten, die von einer noch aktiven Transaktion modifiziert wurden, ausgeschlossen</li><li>- <i>Steal</i>: jede nicht fixierte Seite ist prinzipiell ein Kandidat für die Ersetzung ((-) dreckige Seiten können in der DB geschrieben werden)</li></ul> <p>Einbringen von Änderungen abgeschlossener Transaktionen</p> <ul style="list-style-type: none"><li>- <i>Force</i>: Änderung werden zum Transaktionsende auf den Hintergrundsspeicher geschrieben</li><li>- <i>-force</i>: geänderte Seiten können im Puffer verbleiben ((-) geänderte Seiten sind möglicherweise noch nicht auf die Platte geschrieben)</li></ul> <table><tr><td></td><td>force</td><td>¬ force</td></tr><tr><td>¬ steal</td><td><ul style="list-style-type: none"><li>kein Undo</li><li>kein Redo</li></ul></td><td><ul style="list-style-type: none"><li>Redo</li><li>kein Undo</li></ul></td></tr><tr><td>steal</td><td><ul style="list-style-type: none"><li>kein Redo</li><li>Undo</li></ul></td><td><ul style="list-style-type: none"><li>Redo</li><li>Undo</li></ul></td></tr></table> <p><b>Update in Place</b>: alte Zustand der Seite wird überschrieben <b>Twin-Block-Verfahren</b>: Duplizierung der Seiten <b>Schattenspeicherkonzept</b>: nur geänderte Seiten werden dupliziert</p>		force	¬ force	¬ steal	<ul style="list-style-type: none"><li>kein Undo</li><li>kein Redo</li></ul>	<ul style="list-style-type: none"><li>Redo</li><li>kein Undo</li></ul>	steal	<ul style="list-style-type: none"><li>kein Redo</li><li>Undo</li></ul>	<ul style="list-style-type: none"><li>Redo</li><li>Undo</li></ul>
	force	¬ force								
¬ steal	<ul style="list-style-type: none"><li>kein Undo</li><li>kein Redo</li></ul>	<ul style="list-style-type: none"><li>Redo</li><li>kein Undo</li></ul>								
steal	<ul style="list-style-type: none"><li>kein Redo</li><li>Undo</li></ul>	<ul style="list-style-type: none"><li>Redo</li><li>Undo</li></ul>								
Log-Eintrag (Protokollierung von Änderungsoperationen)	<p>[LSN, TransaktionsID, PageID, Redo, Undo, PrevLSN]</p> <ul style="list-style-type: none"><li>- <i>LSN (Log Sequence Number)</i>: eindeutige Kennung des Log-Eintrags</li><li>- <i>TransaktionsID</i>: TA die die Änderung durchgeführt hat</li><li>- <i>PageID</i>: Kennung der Seite auf der die Änderung vollzogen wurde</li><li>- <i>Redo</i>: Information wie die Änderung nachvollzogen werden kann</li><li>- <i>Undo</i>: Information wie die Änderung rückgängig gemacht werden kann</li><li>- <i>PrevLSN</i>: Zeiger auf den vorhergehenden Log-Eintrag</li></ul> <p><b>Physische Protokollierung</b></p> <ul style="list-style-type: none"><li>- <i>Before-image</i>: enthält den Zustand vor Ausführung der Operation</li><li>- <i>After-image</i>: enthält den Zustand nach Ausführung der Operation</li></ul> <p><b>Logische Protokollierung</b>: das before-image wird durch Ausführung des Undo-codes aus dem after-image generiert, und das after-image durch Ausführung des Redo-codes aus dem before-image berechnet</p> <p>➔ Die Log-information wird zweimal geschrieben: Log-Datei für schnellen Zugriff (R1, R2, R3), Log-Archiv (R4)</p> <p><b>WAL-Prinzip (Write Ahead Log)</b></p>									

	<div>1- Bevor eine Transaktion festgeschrieben wird, müssen alle relevante Logs ausgeschrieben werden</div> <div>2- bevor eine modifizierte Seite ausgelagert werden darf, müssen alle Logs in das temporäre und das Log-Archiv ausgeschrieben werden</div>																									
Phasen des Wiederanlaufs	<div>1- <i>Analyse</i>: Ermittlung der Winner (geschlossene Transaktionen) und Loser</div> <div>2- <i>Wiederholung der Historie</i>: alle protokollierten Änderungen werden in der Reihenfolge ihrer Ausführung in die Datenbasis eingebracht</div> <div>3- <i>Undo der Loser</i>: Loser-Transaktionen werden rückgängig gemacht (Compensating Log Record &lt;#, Ti, Pj, Redo, #, PreviousUndoLog&gt;)</div> <div>(!) Auch während der Recovery-Phase kann das System abstürzen</div>																									
Sicherungspunkte	<div>- <i>Transaktionskonsistent</i></div> <div>- <i>Aktionskonsistent</i>: Transaktionsausführung relativ zu einem aktionskonsistenten Sicherungspunkt und einem Systemabsturz</div> <div>- <i>Unschärf (fuzzy)</i>: modifizierte Seiten werden nicht ausgeschrieben (nur Kennung)<div><div>o MinDirtyPageLSN: minimale LSN, deren Änderung noch nicht ausgeschrieben wurde</div><div>o MinLSN: kleinste LSN der zum Sicherungspunkt aktiven Tas</div></div></div>																									
Mehrbenutzer-synchronisation	<div>Ausführung der Transaktionen im Mehrbenutzerbetrieb</div> <div>Fehler:</div> <div><div>- <i>Lost Update</i>: verlorengegangene Änderungen (Überschreibung der Werte)</div><div>- <i>Dirty Read</i>: read vor commit (commit wird nicht ausgeführt und wird abort)</div><div>- <i>Non Repeatable Read</i>: Lesen zwei verschiedener Werte in Transaktion (andere Transaktion hat eine Änderung durchgeführt)</div><div>- <i>Abhängigkeit von nicht freigegebenen Änderungen</i></div><div>- <i>Phantomproblem</i>: Transaktion sieht neue Daten nicht</div></div> <div>Serialisierbarkeit: Historie ist äquivalent zu einer seriellen Historie</div>																									
Isolation Levels	<table><tr><th></th><th>Lost Update</th><th>Dirty Read</th><th>Non Repeatable Read</th><th>Phantom Problem</th></tr><tr><td>read uncommitted</td><td>✓</td><td>✗</td><td>✗</td><td>✗</td></tr><tr><td>read committed</td><td>✓</td><td>✓</td><td>✗</td><td>✗</td></tr><tr><td>repeatable read</td><td>✓</td><td>✓</td><td>✓</td><td>✗</td></tr><tr><td>serializable</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td></tr></table> <div><div>- Kompromiss zwischen Performanz und Genauigkeit</div><div>-</div></div>		Lost Update	Dirty Read	Non Repeatable Read	Phantom Problem	read uncommitted	✓	✗	✗	✗	read committed	✓	✓	✗	✗	repeatable read	✓	✓	✓	✗	serializable	✓	✓	✓	✓
	Lost Update	Dirty Read	Non Repeatable Read	Phantom Problem																						
read uncommitted	✓	✗	✗	✗																						
read committed	✓	✓	✗	✗																						
repeatable read	✓	✓	✓	✗																						
serializable	✓	✓	✓	✓																						
Theorie der Serialisierbarkeit	<div>Falls <math>T_i</math> ein abort durchführt, müssen alle anderen Operationen vor <math>a_i</math> ausgeführt werden (analoges für das commit)</div> <div><div>- <i>Zwei Lese-Operationen</i>: Reihenfolge irrelevant</div><div>- <i>Lese- und Schreiboperation</i>: Konflikt</div><div>- <i>Zwei Schreiboperationen</i>: Konflikt (Reihenfolge muss festgelegt werden)</div><div>➔ Eine Historie ist serialisierbar wenn sie äquivalent zu einer seriellen Historie ist (zugehörige Serialisierbarkeitsgraph azyklisch)</div></div> <div>Abortete Transaktionen werden im Serialisierbarkeitsgraphen nicht berücksichtigt</div>																									
Rücksetzbare Historie	<div>Eine Historie heißt rücksetzbar, falls immer die schreibende Transaktion vor der lesenden Transaktion ihr commit durchführt. Eine Transaktion darf erst dann ihr committen, wenn alle Transaktionen , von denen sie gelesen hat, beendet sind</div>																									
Strikte Historie	<div>Eine Historie ist strikt, wenn für je zwei Tas <math>t_i</math> und <math>T_j</math> gilt: wenn <math>w_j &lt; o_i</math> dann <math>a_j &lt; o_i</math> oder <math>c_j &lt; o_i</math></div>																									

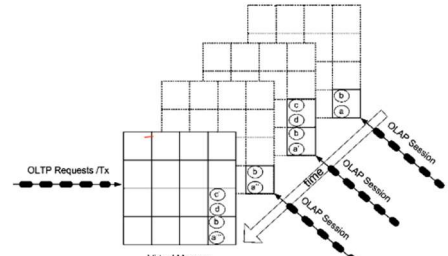
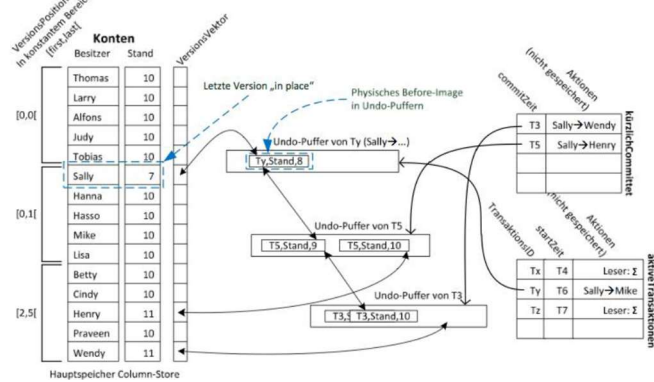
Klassen von Historien	<div></div> <div><ul style="list-style-type: none"><li>● <i>SR</i>: serialisierbare Historien</li><li>● <i>RC</i>: rücksetzbare Historien</li><li>● <i>ACA</i>: Historien ohne kaskadierendes Rücksetzen</li><li>● <i>ST</i>: strikte Historien</li></ul></div> <div><p><i>SR</i>: keine Zyklen <i>RC</i>: Schreiboperation muss vor Leseoperation committen <i>ACA</i>: Schreiber der Daten müssen vor dem lesen der Daten committen <i>ST</i>: Schreiber von Daten muss committen/aborten bevor Daten gelesen oder geschrieben werden</p></div>												
Sperrbasierte Synchronisation	<table><tr><td></td><td><i>NL</i></td><td><i>S</i></td><td><i>X</i></td></tr><tr><td><i>S</i></td><td>✓</td><td>✓</td><td>-</td></tr><tr><td><i>X</i></td><td>✓</td><td>-</td><td>-</td></tr></table> <ul style="list-style-type: none"><li>- <i>Zwei-Phasen-Sperrprotokoll</i><ul style="list-style-type: none"><li>○ Jedes Objekt, das von einer Transaktion benutzt werden soll, muss vorher entsprechend gesperrt werden</li><li>○ Eine TA muss die Sperren anderer TAs auf dem von ihr benötigten Objekt gemäß der Verträglichkeitstabelle beachten</li><li>➔ <b>Wachstum-Phase</b>: Sperren anfordern (keine freigeben)</li><li>➔ <b>Schrumpf-Phase</b>: Sperren freigeben (keine anfordern)</li><li>Bei EOT müssen alle Sperren freigegeben werden</li></ul></li><li>- <i>Strenges Zwei-Phasen Sperrprotokoll</i>: alle Sperren werden bis EOT gehalten</li></ul>		<i>NL</i>	<i>S</i>	<i>X</i>	<i>S</i>	✓	✓	-	<i>X</i>	✓	-	-
	<i>NL</i>	<i>S</i>	<i>X</i>										
<i>S</i>	✓	✓	-										
<i>X</i>	✓	-	-										
Deadlocks Vermeidung	<ul style="list-style-type: none"><li>- <i>Preclaiming</i> (+ S 2PS Protokoll): alle Sperren werden bei BOT angefordert</li><li>- <i>Zeitstempel</i>: jeder Transaktion wird ein eindeutiger Zeitstempel zugeordnet<ul style="list-style-type: none"><li>○ <b>Wound-Wait</b>: wenn T1* älter als T2 ist, wird T2 abgebrochen, sonst wartet T1 auf die Freigabe</li><li>○ <b>Wait-Die</b>: wenn T1* älter als T2 ist, wartet T1, sonst wird T1 abgebrochen und zurückgesetzt</li></ul></li><li>- <i>Multi-Granularity Locking (MGL)</i>: hierarchische Anordnung möglicher Sperrgranulate ➔ Kompatibilitätsmatrix</li><li>- <i>Erkennung durch Wartegraph</i></li></ul>												
Phantomproblem	Vor dem Löschen eines Objekts muss die Transaktion eine X-Sperre für dieses Objekt erwerben. Beim Einfügen eines neuen Objekts erwirbt die einfügende Transaktion eine X-Sperre												
Synchronisation	<ul style="list-style-type: none"><li>- <i>Optimistische Synchronisation</i><ol style="list-style-type: none"><li>1- Lesephase: alle Operationen der Transaktion werden ausgeführt (lokalen Variablen)</li><li>2- Validierungsphase: Entscheidung ob die Transaktion möglicherweise in Konflikt mit andern Transaktionen geraten ist (anhand Zeitstempel)</li><li>3- Schreibphase: wenn die Validierung positiv ist, daten in DB eingebracht<ul style="list-style-type: none"><li>➔ Es ist immer nur eine TA in der Validierungsphase</li><li>➔ Snapshot Isolation (SI): Elemente die geschrieben müssen überschneiden sich nicht (!) Garantiert nicht die Serialisierbarkeit)</li></ul></li></ol></li><li>- <i>Pessimistische Synchronisation</i><ul style="list-style-type: none"><li>○ Zeitstempel-basierende Synchronisation: die Transaktion muss zurückgesetzt werden wenn eine jüngere Transaktion eine Lese- oder Schreibsperre hat</li><li>○ Sperrbasiert<ul style="list-style-type: none"><li>▪ <i>Zwei-Phasen-Sperrprotokoll</i></li><li>▪ <i>Strenges Zwei-Phasen-Sperrprotokoll</i><ul style="list-style-type: none"><li>• <i>Multiple Granularity Locking</i></li></ul></li></ul></li></ul></li></ul>												
Sicherheitsaspekte	<ul style="list-style-type: none"><li>- <i>K Anonymität</i>: nur Aggregatanfragen sind erlaubt</li><li>- <i>SQL Injektion</i></li></ul>												
Discretionary Access Control	Zugriffsregeln mit Objekte, Subjekte, Zugriffsrechte, Prädikate und ein Boolean (der angibt ob der Subjekt das Recht hat an etwas) ➔ <b>Zugriffsmatrix</b> (-) Erzeuger der Daten ist der Verantwortlicher für deren Sicherheit Sichten: Realisierung des Zugriffsprädikats, Schutz von Individualdaten durch Aggregation												
Role Based Access Control	Mengen von Nutzer, Rollen und Rechte <ul style="list-style-type: none"><li>- Explizite / Implizite Autorisierung</li></ul>												

	<ul style="list-style-type: none"> <li>- Positive / Negative Autorisierung</li> <li>- Starke / Schwache Autorisierung</li> </ul> <p>Benutzer mit einem Zugriffsrecht auf einem Objekttypen haben auf die geerbten Attribute in den Untertypen ein gleichartiges Zugriffsrecht</p>
<b>Mandatory Access Control</b>	<p>Hierarchische Klassifikation von Vertrauenswürdigkeit und Sensitivität</p> <ul style="list-style-type: none"> <li>➔ Ein Subjekt darf ein Objekt nur lesen, wenn das Objekt eine geringere Sicherheitseinstufung besitzt</li> <li>➔ Ein Objekt muss mit mindestens der Einstufung des Subjektes geschrieben werden</li> </ul>
<b>Multilevel-Datenbanken</b>	Benutzer soll sich der Existenz unzugänglicher Daten nicht bekannt sein
<b>SQL Injection Attack</b>	<p>Aus den Eingabe-Parametern werden SQL-Anfragen generiert</p> <ul style="list-style-type: none"> <li>➔ Prepared Statements: bereitet die SQL Nachfrage mit Platzhalter vor <ul style="list-style-type: none"> <li>1- Prepare: bereitet die Anfrage vor und legt Datentyp fest</li> <li>2- QueryPrepare: ersetzt die Platzhalter als String</li> </ul> </li> <li>(+) Anfrage kann als Template benutzt werden</li> <li>➔ Input Sanitization: ersetzen/Entfernen der Sonderzeichen</li> </ul>
<b>Kryptographie</b>	<p>Nur die Verschlüsselung der gesendeten Information kann einen effektiven Datenschutz gewährleisten</p> <ul style="list-style-type: none"> <li>- <i>Geheim Schlüssel</i>: gleiches Schlüssel für ver- und Entschlüsselung</li> <li>- <i>Public Key</i>: verschlüsseln mit öffentlichen Schlüssel und entschlüsseln mit Geheim Schlüssel ➔ <b>RSA</b></li> </ul>
<b>Authentifizierung</b>	<ul style="list-style-type: none"> <li>- <i>Three-Way handshake</i></li> <li>- <i>Public Key Authentifizierung</i></li> <li>- <i>Digitale Signaturen</i></li> </ul>
<b>Ebenen des Datenschutzes</b>	<ol style="list-style-type: none"> <li>1- Legislative Maßnahmen</li> <li>2- Organisatorische Maßnahmen</li> <li>3- Authentisierung</li> <li>4- Zugriffskontrolle</li> <li>5- Kryptographie</li> </ol>
<b>Objektorientierte Datenmodellierung</b> <b>X</b>	<p>Wiederverwendbarkeit, Operationen direkt in Sprache des Objektmodells realisiert</p> <ul style="list-style-type: none"> <li>- <i>1 : 1 Beziehungen</i> (z.B. Professor ➔ Raum)</li> <li>- <i>1 : N Beziehungen</i> (z.B. Professor ➔ Vorlesungen)</li> <li>- <i>N : M Beziehungen</i> (z.B. Studenten ➔ Vorlesungen)</li> <li>- <i>Rekursive N : M Beziehungen</i> (z.B. Vorlesungen ➔ Voraussetzen)</li> <li>- <i>Ternäre Beziehungen</i> (z.B. Vorlesungen, Studenten, Professoren ➔ Prüfen)</li> </ul> <p><b>Eigenschaften von Objekten</b></p> <ul style="list-style-type: none"> <li>- <i>Identität</i>: wesentliche Charakteristikum objekt-orientierter Datenmodellierung <ul style="list-style-type: none"> <li>o Physische OIDs: enthalten den Speicherort des Objekts</li> <li>o Logische OIDs: unabhängig vom Speicherort der Objekte (Mapping)</li> </ul> </li> <li>- <i>Typ</i> <ul style="list-style-type: none"> <li>o Extensionen</li> <li>o Schlüssel</li> </ul> </li> <li>- <i>Wert/Zustand</i></li> </ul> <p><b>Modellierung des Verhaltens: Operationen</b></p> <ul style="list-style-type: none"> <li>- <i>Beobachter</i>: Objektzustand erfragen</li> <li>- <i>Mutatoren</i>: Änderungen am Zustand der Objekte (mutierbar)</li> <li>- <i>Konstruktoren und Destruktoren</i>: neue Objekte eines bestimmten Objekttyps zu erzeugen, existierendes Objekt auf Dauer zu zerstören</li> <li>➔ Man spezifiziert den Namen der Operation, die Anzahl und die Typen der Parameter, den Typ des Rückgabewerts der Operation und mögliche exceptions</li> </ul> <p><b>Vererbung</b>: Eigenschaften eines Obertyps an den Untertyp (in ODGM kann eine Klasse nur von einer Klasse erben)</p> <p><b>Subtypisierung</b>: eine Untertyp-Instanz ist überall dort einsetzbar, wo eine Obertyp-Instanz gefordert ist</p> <p>Objektorientierte Entwurfsmethode ➔ UML-Standard (Unified Modelling Language)</p>
<b>Large Objects</b> <b>X</b>	<ul style="list-style-type: none"> <li>- <i>CLOB</i> (Character Large Object): lange Texte werden gespeichert</li> <li>- <i>BLOB</i> (Binary Large Object): für die Archivierung</li> <li>- <i>NCLOB</i>: Texte mit 1-Byte Charakter-Daten beschränkt</li> </ul>
<b>Extensionale Datenbasis</b>	Die extensionale Datenbasis besteht aus einer Menge von Relationen und entspricht einer "ganz normalen" relationalen Datenbasis
<b>Intensionale Datenbasis</b>	Die intensionale Datenbasis besteht aus einer Menge von hergeleiteten Relationen. Die IDB wird durch Auswertung des Datalog-Programms aus der EDB generiert

	<p>Deduktionskomponente: besteht aus einer Menge von Herleitungsregeln (Datalog). Grundbestandteile der Regeln sind atomare Formeln oder Literale</p> <ul style="list-style-type: none"> <li>- Prädikate beginnen mit einem Kleinbuchstaben</li> <li>- Die zugehörige Relationen werden mit gleichem Namen, aber mit einem Großbuchstaben beginnend bezeichnet</li> </ul> <p>→ Ein Datalog-Programm ist rekursiv, wenn der Abhängigkeitsgraph einen (oder mehreren) Zyklen hat</p> <p>→ Eine Datalog-Regel ist sicher, wenn alle Variablen im Kopf beschränkt sind:</p> <ul style="list-style-type: none"> <li>- Variable im Rumpf der Regel in mindestens einem normalen Prädikat vorkommt (nicht nur in eingebauten Vergleichsprädikat)</li> <li>- Prädikat der Form <math>X = c</math> mit einer konstante <math>c</math> im Rumpf der Regel existiert oder</li> <li>- Prädikat der Form <math>X = Y</math> im Rumpf vorkommt, und man schon nachgewiesen hat, dass <math>Y</math> eingeschränkt ist</li> </ul> <p>→ Eine Regel mit einem negierten Prädikat im Rumpf kann nur dann sinnvoll ausgewertet werden, wenn <math>Q_i</math> schon vollständig materialisiert ist (Regeln im Kopf müssen zuerst ausgewertet werden) → Stratifiziert</p>
<b>Datalog non-rec</b>	<p>Bezeichnet die Datalog Sprache eingeschränkt auf nicht rekursive Programme aber erweitert um Negation. Diese Sprache hat genau die gleiche Ausdruckskraft wie die relationale Algebra</p>
<b>Verteilte Datenbanken</b>	<p>Sammlung von Informationseinheiten die aus mehreren Rechnern bestehen und die durch Kommunikationsnetzen verbunden sind</p> <ol style="list-style-type: none"> <li>1- Globales Schema</li> <li>2- Fragmentierungsschema</li> <li>3- Zuordnungsschema</li> <li>4- Lokales Schema</li> <li>5- Lokales DBMS</li> <li>6- Lokale DB (Station)</li> </ol> <p>Transparenz in verteilten Datenbanken: Grad der Unabhängigkeit den ein VDBMS dem Benutzer beim Zugriff auf verteilte Daten vermittelt</p> <ul style="list-style-type: none"> <li>- <i>Fragmentierungstransparenz</i></li> <li>- <i>Allokationstransparenz</i>: Benutzer müssen Fragmentierung kennen, aber nicht den Aufenthaltsorts eines Fragments</li> <li>- <i>Lokale-Schema Transparenz</i>: Benutzer muss auch noch den Rechner kennen, auf dem ein Fragment liegt (setzt voraus, dass alle Rechner dasselbe Datenmodell und dieselbe Anfragesprache verwenden)</li> </ul> <p><b>Join-Auswertung in VDBMS</b>: spielt kritischere Rolle als in zentralisierten Datenbanken, da Argumente können auf unterschiedlichen Stationen des VDBMS liegen</p> <ul style="list-style-type: none"> <li>- <i>Join-Auswertung ohne Filterung</i> <ul style="list-style-type: none"> <li>Nested Loops (zu aufwendig)</li> <li>Transfer einer Argumentrelation (Problematisch)</li> <li>Transfer beider Argumentrelationen</li> </ul> </li> <li>- <i>Merge-Join</i> (bei vorliegender Sortierung)</li> <li>- <i>Hash-Join</i> (bei fehlender Sortierung)</li> </ul> <p><i>Join-Auswertung mit Filterung</i> (e.g. mit Hashfilter/<b>Bloom-Filter</b>): Einsatz bei sehr voluminösen Joins</p> <ol style="list-style-type: none"> <li>1. Transfer der Unterschiedlichen Werte von R</li> <li>2. Auswertung des Semi-Joins</li> <li>3. Auswertung des Joins</li> </ol>
<b>Kommunikationsmedien</b>	<ul style="list-style-type: none"> <li>- LAN: local area network</li> <li>- WAN: wide area network</li> <li>- Telefonverbindungen</li> </ul>
<b>Fragmentierung</b>	<p>Fragmente enthalten Daten mit gleichem Zugriffsverhalten</p> <ul style="list-style-type: none"> <li>- <i>Horizontale Fragmentierung</i>: Zerlegung der Relation in disjunkte Tupelmengen</li> <li>- <i>Vertikale Fragmentierung</i>: Zusammenfassung von Attributen mit gleichem Zugriffsmuster. Um Rekonstruierbarkeit zu garantieren: <ul style="list-style-type: none"> <li>o Jedes Fragment enthält den Primärschlüssel (Verletzung Disjunktheit)</li> <li>o Jedem Tupel der Originalrelation wird ein eindeutiges Surrogat zugeordnet, welches in jedes vertikale Fragment des Tupels ist</li> </ul> </li> <li>- <i>Kombinierte Fragmentierung</i>: Anwendung horizontaler und vertikaler Fragmentierung auf dieselbe Relation</li> </ul> <p>Korrektheits-Anforderungen:</p>

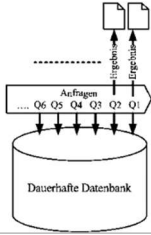
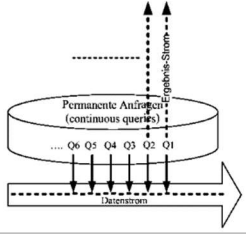
	<ul style="list-style-type: none"> <li>- <i>Rekonstruierbarkeit</i> (Vertikal --&gt; Join, horizontal --&gt; Vereinigung)</li> <li>- <i>Vollständigkeit</i></li> <li>- <i>Disjunktheit</i></li> </ul>
<b>Allokation</b>	<p>Fragmente werden den Stationen zugeordnet (selbe Fragment kann mehrmals zugeordnet werden)</p> <ul style="list-style-type: none"> <li>- <i>Mit Replikation</i></li> <li>- <i>Ohne Replikation</i></li> </ul>
<b>Anfrageübersetzer</b>	<p>Generierung eines Anfrageauswertungsplans auf den Fragmenten</p> <p><b>Parameter für die Kosten eines Auswertungsplan</b></p> <ul style="list-style-type: none"> <li>- Kardinalität von Argumentrelationen</li> <li>- Selektivität von Joins und Selektionen</li> <li>- Transferkosten für Datenkommunikation</li> <li>- Auslastung der einzelnen VDBMS-Stationen</li> </ul>
<b>Anfrageoptimierer</b>	<p>Generierung eines möglichst effizienten Auswertungsplanes</p> <p>➔ Um Selektionen und Projektionen über den Vereinigungsoperator hinweg nach unten zu drücken benötigt man folgende Regeln</p> <p>Effektive Anfrageoptimierung muss auf Basis eines Kostenmodells durchgeführt werden und soll mehrere Alternativen für unterschiedlichen Auslastungen erzeugen</p>
<b>Transaktionskontrolle in VDBMS</b>	<p>Transaktionen können sich über mehrere Rechnerknoten erstrecken</p> <p><b>Recovery:</b></p> <ul style="list-style-type: none"> <li>- <i>Redo</i>: wenn eine Station nach einem Fehler wieder anläuft, müssen alle Änderungen einmal abgeschlossener Transaktionen, an den an dieser Station abgelegten Daten wiederhergestellt werden</li> <li>- <i>Undo</i>: die Änderungen noch nicht abgeschlossener lokaler und globaler Transaktionen müssen auf den an der abgestürzten Station vorliegenden Daten rückgängig gemacht werden</li> </ul> <p>Eine globale Transaktion muss atomar beendet werden:</p> <ul style="list-style-type: none"> <li>- <i>Commit</i>: globale Transaktion wird an allen lokalen Stationen festgeschrieben</li> <li>- <i>Abort</i>: globale Transaktion wird gar nicht festgeschrieben</li> </ul>
<b>Zweiphasen-Commit-Protokoll</b>	<p>Gewährleistet die Atomarität der EOT-Behandlung (wird vom Koordinator überwacht)</p> <ol style="list-style-type: none"> <li>1. <i>Prepare</i>: Koordinator schickt an allen Agenten eine Prepare-Nachricht um herauszufinden ob sie Transaktionen festschreiben können</li> <li>2. <i>Ready/Failed</i>: Agent sendet entweder eine Ready- oder eine Failed Nachricht</li> <li>3. <i>Commit</i>: falls alle Agenten mit Ready geantwortet haben, sendet der Koordinator eine commit Nachricht an allen Agenten</li> <li>4. <i>Ack</i>: haben die Agenten ihre lokale EOT-Behandlung abgeschlossen, schicken sie eine Ack-Nachricht</li> </ol> <p><b>Fehlersituationen:</b></p> <ul style="list-style-type: none"> <li>- <i>Absturz eines Koordinators</i> <ul style="list-style-type: none"> <li>○ Vor dem Commit: versenden einer Abort Nachricht</li> <li>○ Nachdem Agenten mit Ready geantwortet haben: Blockierung der Agenten</li> </ul> </li> <li>- <i>Absturz eines Agenten</i> <ul style="list-style-type: none"> <li>○ Keine Antwort an der Prepare Nachricht: Koordinator schickt Abort</li> <li>➔ Kein Ready-Eintrag: Agent schickt Failed Nachricht</li> <li>➔ Ready-Eintrag aber keinen Commit: Agent fragt Koordinator</li> <li>➔ Commit-Eintrag vorhanden: Redo</li> </ul> </li> <li>- <i>Verlorengegangene Nachricht</i>: nach Timeout-Intervall geht Koordinator davon aus, dass betreffender Agent nicht funktionsfähig ist und sendet Abort</li> </ul>
<b>Mehrbenutzersynchronisation in VDBMS</b>	<ul style="list-style-type: none"> <li>- <i>Serialisierbarkeit</i>: lokale Serialisierbarkeit reicht nicht aus --&gt; globale Serialisierbarkeit</li> <li>- <i>Zwei-Phasen-Sperrprotokoll</i> <ul style="list-style-type: none"> <li>○ Lokale Sperrverwaltung: globale Transaktion muss vor Zugriff eines Datums A, das auf Station S liegt, eine Sperre von Sperrverwalter der Station S erwerben</li> <li>○ Globale Sperrverwaltung: alle Transaktionen fordern alle Sperren an einer einzigen, ausgezeichneten Station an (--&gt; Engpass)</li> </ul> </li> </ul> <p><b>Synchronisation bei replizierten Daten: ROWA (Read One Write All)</b></p> <ul style="list-style-type: none"> <li>- <i>Quorum-Consensus Verfahren</i>: Ausgleich der Leistungsfähigkeit zwischen Lese- und Änderungstransaktionen</li> </ul>

<b>Deadlock Vermeidung in VDBMS</b>	<ul style="list-style-type: none"> <li>- <i>Zentralisierte Deadlock Erkennung</i>: Stationen melden lokal vorliegenden Wartebeziehungen an neutralen Knoten, der daraus globalen Wartegraphen baut</li> <li>- <i>Dezentrale Deadlock Erkennung</i>: lokale Wartegraphen an den einzelnen Stationen + jeder lokale Wartegraph hat einen Knoten "External" der Stationenübergreifenden Wartebeziehungen modelliert</li> <li>- <i>Vermeidung von Deadlocks</i> <ul style="list-style-type: none"> <li>o Timeout: Transaktion wird zurückgesetzt und neu gestartet</li> <li>o Wound/Wait (Zeitstempel basiert); nur jüngere TAs warten auf ältere</li> <li>o Wait/Die (Zeitstempel basiert): nur ältere TAs warten auf jüngere</li> <li>o Optimistische Mehrbenutzersynchronisation</li> </ul> </li> </ul>
<b>Online Transaction Processing (OLTP)</b>	<p>Hoher Parallelitätsgraph, viele kurze Transaktionen → Hohe Verfügbarkeit muss gewährleistet werden</p> <ol style="list-style-type: none"> <li>1. Ein Datenbank-Server</li> <li>2. Mehrere Applikations-server (skalierbar)</li> <li>3. Sehr Viele Clients</li> </ol>
<b>Online Analytical Processing (OLAP)</b>	<p>Data Warehouse Anwendungen, Decision Support, Data Mining</p> <ol style="list-style-type: none"> <li>1. <i>Extract</i> (from OLTP)</li> <li>2. <i>Transform</i></li> <li>3. <i>Load</i> (in data Warehouse)</li> </ol> <p><b>Stern-Schema</b></p> <ul style="list-style-type: none"> <li>- Eine sehr große Faktentabelle</li> <li>- Mehrere Dimensionstabellen (relativ klein)</li> </ul> <p>→ Normalisierung führt zum Schneeflocken-Schema</p> <p><b>Roll-up Anfragen</b>: vergrößere</p> <p><b>Drill-down Anfragen</b>: verfeinere (reduziere)</p> <p><b>Slice and Dice</b>: Materialisierung von Aggregaten (Cube-Operator)</p> <p>→ Teilaggregate sind für eine Aggregation A wiederverwendbar wenn es einen gerichteten Pfad von T nach A gibt (Materialisierungshierarchie)</p>
<b>Bitmap-Indexe</b>	<p>Optimierung durch Komprimierung der Bitmaps</p> <ul style="list-style-type: none"> <li>- <i>Run length compression</i>: speichere Länge der Nullfolgen zwischen zwei Einsen</li> <li>- <i>Mehrmodus-Komprimierung</i>: speichere Länge der Null- und Einserfolgen</li> </ul>
<b>Decision-Support Anfragen</b>	<ul style="list-style-type: none"> <li>- <i>Top N Anfragen</i>: stop after ... <ul style="list-style-type: none"> <li>o Random Access</li> <li>o No Random Access (NRA)</li> </ul> </li> <li>- <i>Ranking</i>: RANK() over ...</li> <li>- <i>Window</i>: range between ... preceding and ... following <ul style="list-style-type: none"> <li>→ Window Functions werden nach group by und vor order by ausgewertet</li> </ul> </li> <li>- <i>Vorhergehendes Tupel in Frame</i>: Lag(...)</li> <li>- <i>Partitionieren/Sortieren</i>: partition by / order by</li> </ul> <p><b>Frame Begrenzungen</b>:</p> <ul style="list-style-type: none"> <li>- <i>Current row</i>: aktuelle Tupel inklusive seiner Peers angegeben</li> <li>- <i>Unbounded Preceding</i>: alle dem aktuellen Tupel vorausgehenden Tupel</li> <li>- <i>Unbounded following</i>: endet erst beim letzten Tupel der Partition</li> <li>- <i>Skyline</i>: bei etwas muss er besser sein</li> </ul>
<b>Data Mining</b>	<ul style="list-style-type: none"> <li>- <i>Klassifikation</i> (Entscheidungsbaume)</li> <li>- <i>Assoziationsregeln</i></li> <li>- <i>Clustering</i></li> </ul>
<b>Entscheidungsbaume</b>	<ol style="list-style-type: none"> <li>1- <i>Trainingsmenge</i>: dient als Grundlage für die Vorhersage von neuen Objekten</li> <li>2- <i>Rekursives Partitionieren</i>: fange mit einem Attribut an uns spalte die Tupelmenge</li> </ol>
<b>Assoziationsregeln</b>	<ul style="list-style-type: none"> <li>- <i>Confidence</i>: welchen Prozentsatz der Datenmenge, bei der die Voraussetzung erfüllt ist, die Regel auch erfüllt ist  <math>Confidence(L \rightarrow FI - L) = support(FI) / support(L)</math> <ul style="list-style-type: none"> <li>→ Vergrößern der linken Seite führt zur Erhöhung der Confidence</li> </ul> </li> <li>- <i>Support</i>: wie viele Datensätze gefunden wurden, um die Gültigkeit der Regel zu verifizieren <ul style="list-style-type: none"> <li>→ Finde alle Assoziationsregeln mit Support &gt; minsupp und Confidence &gt; minconf</li> </ul> </li> </ul> <p><b>A-Priori-Algorithmus</b>: alle Teilmengen eines Frequent Itemset müssen auch FI sein</p>
<b>Clustering</b>	<ul style="list-style-type: none"> <li>- <i>Greedy Heuristik</i> <ol style="list-style-type: none"> <li>a. Lese sequentiell alle Datensätze</li> </ol> </li> </ul>

	<div><div>b. Bestimme den niedrigsten Abstand zum Zentrum existierender Clusters</div><div>c. Falls Abstand grösser Epsilon, füge neuen Cluster ein</div><div><div>- <i>K-Means</i>: minimiere die Summe der Abstände der Datenpunkte x zum Mittelpunkt „ihres“ Clusters</div><div><div>1- Initialisierung: wähle zufällig k Mittelwerte</div><div>2- Zuordnung zu Clustern</div><div>3- Neuberechnung der Mittelwerte</div></div><div>- <i>DBScan</i>: Density Based Clustering</div></div></div>
<div>Leistungsengpässe Klassischen DBS</div>	<div><div><div><div>- 35 % Pufferverwaltung</div><div>- 30 % Mehrbenutzersynchronisation</div><div>- 12 % Logging</div><div>- 16 % Optimierungspotential</div><div>- 7 % Nützlich</div></div><div><div>Speicherhierarchie:</div><div><div>1- Register (1-8 Byte)</div><div>2- Cache (8-128 Byte)</div><div>3- Hauptspeicher (4-64 KB)</div><div>4- Plattenspeicher (Faktor 10<sup>5</sup>)</div><div>5- Archivspeicher</div></div></div></div><div>➔ Einsatz von Hauptspeicher Datenbanksystemen: HTAP (Hybrid Transactional Analytical Processing)</div></div>
<div>Logischer Row-Store</div>	Transaktionen bekommen jeweils eine eigene Zeile
<div>Logischer Column-Store</div>	Transaktionen werden in mehreren Spalten gespatet (-) Transaktionen verlieren 20 % an Performance (++) Analysen verdienen Faktorweise an Performance
<div>Snapshot für Anfragen</div>	<div><div>➔ Zweiteilung von OLAP und OLTP</div><div>Kurz und viele Transaktionen (OLTP) ➔ Hauptdatenbank</div><div>Komplex und lange Transaktionen (OLAP) ➔ Snapshot der Hauptdatenbank</div><div><div>- <i>Update Staging</i>: merge erfolgt periodisch</div><div>- <i>Scan-Only Datenbanken</i>: liest die geänderten Daten (nur für sehr einfache TAs)</div><div>- Snapshotting via forking</div><div>- Snapshot Maintenance (copy on write)</div></div><div></div></div>
<div>Kompaktifizierung der Datenbank</div>	<div><div>- <i>Heiße Objekte</i>: Working Set, unkomprimiert, kleine Seiten</div><div>- <i>Abkühlenden Objekte</i>: heiße und kalte Objekte gemischt, unkomprimiert, kleine Seiten</div><div>- <i>Kalte Objekte</i>: unkomprimiert, kleine Seiten</div><div>- <i>Gefrorene Objekte</i>: kalte, komprimierte Objekte, große Seiten, geänderte Objekte werden als ungültig markiert ➔ Read Only</div></div> <div>➔ Geänderte Objekte werden heiße Objekte</div>
<div>Transaktionsverwaltung</div>	<div><div>- <i>Serielle Ausführung auf Partitionen</i>: Zuordnung zu einer geeigneten Partition (bei falscher Zuordnung ➔ Abort, und TA wird neu gestartet mit einer Sperre)</div><div>- <i>Isolation von OLAP und OLTP</i></div><div>- <i>Tentative Ausführung langer Transaktionen</i></div><div>- <i>Precision Locking</i>: Prädikatsprüfung</div></div>
<div>Multi-Version Concurrency Control</div>	<div><div></div><div><div>Undo-Buffer:</div><div>[Undo-ID, TID, NextUID, Zeitstempel, Attribut, Undo]</div><div>Mit Precision Locking</div><div><div>- Lesende Anfragen sind immer erlaubt</div><div>- Falls schreibende Anfragen: Überlappender Prädikatbereich?<div><div>○ Falls ja, BOT und commit Reihenfolge beachten</div></div></div></div></div></div>



<b>Indexstrukturen für Hauptspeicher-DB</b>	<ul style="list-style-type: none"> <li>- <i>Binärer Suchbaum</i> (<math>O(\log N)</math>)</li> <li>- <i>Radix-Baum / Trie / Praefixbaum</i> (<math>O(k)</math>)</li> <li>- <i>ART-Baum</i>: Dynamisch, so schnell wie Hashtabellen, sortiert</li> </ul>
<b>Hochparallelen Sort/Merge-Joins</b>	<ul style="list-style-type: none"> <li>- <i>Bereichspartitionierung</i> (e.g. Partitionierung nach Größen)</li> <li>- <i>Radix-Bereichspartitionierung</i> (binär)</li> <li>- <i>Paralleler Radix-Join</i></li> <li>- <i>Mehrfache Partitionierung des Radix-Joins</i> → Cache-Lokalität (um keine Cache Fehler mehr zu haben)</li> <li>- <i>Hash-Join-Teams</i>: Globale Hashtabelle</li> <li>- <i>NUMA-Lokale Arbeitszuteilung</i> (morsel driven) → Lastbalancierung = Work stealing</li> <li>- <i>Adaptive Dynamische Parallelisierung</i></li> </ul>
<b>Algorithmen fuer sehr grossen Datenmengen</b>	<ul style="list-style-type: none"> <li>- <i>Nested Loop</i> (<math>O(N^2)</math>): nicht skalierbar</li> <li>- <i>Sortieren</i>: SORT-MERGE Join (<math>O(N \log N)</math>)</li> <li>- <i>Partitionieren und Hashing</i> (<math>O(N)</math>) <ul style="list-style-type: none"> <li>o <b>Build</b>: create Hash table</li> <li>o <b>Probe</b>: Hash and search</li> </ul> </li> </ul>
<b>HTML</b>	Datenmodell ohne Schema (nur Insider können Tags verstehen) → wenig geeignet als Datenaustauschformat
<b>Relationales Datenmodell</b>	Schema ist vorgegeben und man kann nur Schemakonforme Daten einfügen → Kein Datenaustauschformat
<b>XML (Extensible Markup Language)</b>	<p>Datenmodell liegt zwischen HTML und das relationale Datenmodell. Semi-strukturierte Daten (teilweise Schematisch). Wenn ein Schema vorgegeben ist, muss dieses eingehalten werden → Datenaustausch Format</p> <ul style="list-style-type: none"> <li>- <i>Tags</i>: geben die Bedeutung der Elemente an (immer paarweise: <math>\langle \dots \rangle \langle / \dots \rangle</math>)</li> </ul> <p><b>XML-Daten mit Schema</b></p> <ul style="list-style-type: none"> <li>- <i>Wurzelement</i> (e.g. Buch)</li> <li>- <i>Attribut</i> (e.g. Jahr)</li> <li>- <i>Unterelemente</i> (Reihenfolge ist relevant) (e.g. Titel, Autor, ...)</li> </ul> <p>XML erlaubt rekursive Strukturen</p> <p><b>IDREF(S)</b>: Menge von Referenzen (e.g. referenzieren von Vater UND Mutter in Stammbaum → XML ist nicht geeignet für N : M Beziehungen)</p> <p>→ XML ist sehr gut für die Modellierung von Hierarchien (entsprechen den geschachtelten Elementen) → Funktionale Beziehungen</p> <pre> graph LR     Uni[Universität] -- 1 --&gt; Fak1[Fakultäten]     Fak1 -- N --&gt; Fak2[Fakultät]     Fak2 -- 1 --&gt; Prof1[Professoren]     Prof1 -- N --&gt; Prof2[Professorin]     Prof2 -- 1 --&gt; Vor1[Vorlesungen]     Vor1 -- N --&gt; Vor2[Vorlesung] </pre> <p>→ XML-Anfragesprache ist XQuery</p>
<b>XQuery</b>	<p>Basiert auf Xpath, einer Sprache für Pfadausdrücke. Ein Lokalisierungspfad besteht aus einzelnen Lokalisierungsschritten, die aus drei Teilen bestehen:</p> <p>Achse::Knotentest[Prädikat]</p> <p><b>Xpath Achsen:</b></p> <ul style="list-style-type: none"> <li>- <i>Self</i>: Referenzknoten</li> <li>- <i>Attribute</i>: alle Attribute des Referenzknotens</li> <li>- <i>Child</i>: alle direkten Unterelemente bestimmt</li> <li>- <i>Descendant</i>: alle direkten und indirekten Unterelemente</li> <li>- <i>Descendant-or-self</i>: direkten und indirekten Unterelemente mit Referenzknoten</li> <li>- <i>Parent</i>: Vaterknoten des Referenzknotens</li> <li>- <i>Ancestor</i>: alle Knoten auf dem Pfad vom Referenzknoten zur Wurzel</li> <li>- <i>Ancestor-or-self</i>: alle Knoten auf dem Pfad vom Referenzknoten zur Wurzel mit Referenzknoten</li> <li>- <i>Following-sibling</i>: Dokumentreihenfolge nachfolgenden Kinder des Elternknotens von self</li> <li>- <i>Preceding-sibling</i>: Dokumentreihenfolge vorangehenden Kinder des Elternknotens von self</li> <li>- <i>Following</i>: alle Knoten die nach dem Referenzknoten aufgeführt sind</li> <li>- <i>Preceding</i>: alle Knoten die vor dem Referenzknoten vorkommen</li> </ul> <p><b>Verkürzte Syntax</b></p> <ul style="list-style-type: none"> <li>- <code>.</code> → Aktueller Referenzknoten (self::)</li> <li>- <code>..</code> → Vaterknoten (parent::)</li> <li>- <code>/</code> → Abgrenzung einzelner Schritte oder Wurzel</li> <li>- <code>//</code> → descendant-or-self::node()</li> <li>- <code>@AttrName</code> → Attributzugriff</li> </ul>

	<p><b>XQuery Anfragesyntax (FLWOR)</b></p> <ul style="list-style-type: none"> <li>- For: Schleifen <span style="float: right;">&lt;MäeutikVoraussetzungen&gt;</span></li> <li>- Let: Variable definieren <span style="float: right;">{ for \$m in doc("Uni.xml")//Vorlesung[Titel= "Mäeutik"],</span></li> <li>- Where: selektieren <span style="float: right;">\$v in doc("Uni.xml")//Vorlesung</span></li> <li>- Order by: sortieren <span style="float: right;">where contains(\$m/@Voraussetzungen,\$v/@VorlNr)</span></li> <li>- Return: neuer XML <span style="float: right;">return \$v/ Titel}</span></li> </ul> <p style="text-align: right;">&lt;/MäeutikVoraussetzungen&gt;</p> <p>➔ Anfragen immer in {}</p> <p><b>XML in relationalen Datenbanken:</b> Speicherung von XML Dokumenten in Relationen</p> <ul style="list-style-type: none"> <li>- <i>BLOB</i> (Binary Large Objects): keine Anfragemöglichkeit, keine Strukturierung, nur Archivierung</li> <li>- <i>Shreddern</i>: Speicherung aller Kanten des XML-Baums (Relationale Darstellung)</li> <li>- <i>Objektrelationale Speichermodelle</i>: Ausnutzung von Schemainformationen</li> </ul> <p>➔ Auswertung von Pfadausdrücken: e.g. #Buch#Autoren#Autor#Nachname</p> <p><b>XML Nachteile</b></p> <ul style="list-style-type: none"> <li>(-) Verbos</li> <li>(-) Nettoinformation wird aufgebläht</li> <li>(-) Unterscheidung Attribut und Element</li> </ul> <p>➔ <b>JSON</b></p>
<b>JSON</b>	<p>Basiert auf Key/Value Paare (ohne Reihenfolge, oder Arrays mit Reihenfolge)</p> <p>➔ Keine Attribute</p> <p>➔ NoSQL-DBMS</p>
<b>Web Services</b>	<p>SOAP (Simple Object Access Protocol): basiert auf XML und ermöglicht entfernte Prozeduraufrufe</p>
<b>Resource Description Framework (RDF)</b>	<p><b>Triple-Datenmodell:</b> (Subjekt, Prädikat, Objekt)</p> <ul style="list-style-type: none"> <li>- Subjekte und Objekte sind Knoten</li> <li>- Prädikate sind gerichtete Kanten (von Subjekten nach Objekten)</li> </ul> <p>Es gibt mehrere leicht unterschiedliche textuelle RDF-Tripeldarstellungen, die man unter dem Namen N3, N-Triples oder Turtle findet</p> <p><b>Kurzformen</b></p> <ul style="list-style-type: none"> <li>- ; ➔ Subjekt wie zuvor</li> <li>- , ➔ Subjekt und Prädikat wie zuvor</li> <li>- _ ➔ Namenlosen Knoten</li> </ul> <p><b>SPARQL:</b> ist die RDF Anfragesprache <span style="float: right;">PREFIX ex: &lt;http://www.example.org&gt;</span></p> <ul style="list-style-type: none"> <li>- UNION <span style="float: right;">SELECT ?KempersBuecherTitel WHERE</span></li> <li>- OPTIONAL <span style="float: right;">{ ?KempersBuecher ex:Autor ?k.</span></li> <li>- FILTER <span style="float: right;">?k ex:NachName "Kemper".</span></li> <li>- SELECT COUNT <span style="float: right;">?KempersBuecher ex:Titel ?KempersBuecherTitel.</span></li> </ul> <p style="text-align: right;">}</p> <p>Implementierung einer RDF-Datenbank ➔ so viele B-Bäume wie möglich</p> <p><b>Kompressionstechnik: Dictionary und Präfix</b></p> <ul style="list-style-type: none"> <li>- Jedes Tripel wird genau 6 mal repliziert abgelegt [(p,s,o), (o,s,p), ...]</li> <li>- Aggregierte Indexe geben die Anzahl der Vorkommen des jeweiligen Musters</li> <li>- In den Blättern der B-Bäume wird eine Präfix-Komprimierung durchgeführt</li> <li>- Es wird nur die Differenz zum code des Vorgängers gespeichert</li> </ul>
<b>Datenbanken vs. Datenströme</b>	<ul style="list-style-type: none"> <li>- <i>Bereichsanfrage</i>: between ... and ...</li> <li>- <i>Fenster-Anfrage</i>: window(range ...)</li> <li>- <i>Sliding Windows</i>: window(range ... slide ...)</li> </ul> <div style="display: flex; justify-content: space-around; align-items: center;">   </div>
<b>Information Retrieval</b>	<p>➔ Ranking von Dokumenten um relevante Informationen zu finden</p> <p>➔ Ähnlichkeit von Dokumenten zu erkennen</p> <p><b>TF-IDF: Term Frequency – Inverse Document Frequency</b></p> <p>V: Vokabular mit  V  Worten</p> <p><math>f_{ij}</math>: Frequenz von Wort i in j</p> <p><math>TF_{ij}</math>: Term Frequency Normalisiert</p> <p>➔ Desto seltsamer ein Wort ist, desto grösser die Bedeutung</p> <p><b>Page Rank:</b> normalisierte Reputation der Verweise</p> $r(A) = \frac{\alpha}{N} + (1 - \alpha) \left( \frac{r(B_1)}{ B_1 } + \dots + \frac{r(B_n)}{ B_n } \right)$ <p style="text-align: right;">Alpha: Dämpfungsfaktor</p> <div style="display: flex; justify-content: space-between; align-items: center;"> <math display="block">TF_{ij} = f_{ij} / \sum_{i=1.. V } f_{ij}</math> <math display="block">IDF_i = \log(N/n_i)</math> </div> $rel(D_j, Q) = \sum_{i \in Q} TF_{ij} * IDF_i$

	<p>➔ Mathematisches Modell des PageRank mit Matrix (sparse Matrix)</p> $M_{ij} = \begin{cases} 1/ P_j  & \text{falls } P_j \text{ auf } P_i \text{ verweist} \\ 0 & \text{sonst} \end{cases}$ <p>Man berechnet dann iterativ die Vektoren</p> $p_1 = M * p_0, p_2 = M * p_1 = M * (M * p_0) = M^2 * p_0, \dots, p_i = M^i * p_0$ <p><b>HITS-Algorithmus: Hubs und Autoritäten</b></p> <p>Hub-Wert einer Seite i wie folgt definiert</p> $h_i = \delta \sum_{j=1 \dots N} A_{ij} a_j \quad A_{ij}: \text{Adjazenzmatrix (Verweise von i nach j)}$ <p><i>Hubs</i>: Verweisen auf relevante Seiten (Autoritäten)</p> <ol style="list-style-type: none"> <li>1- Berechne die Hub-Werte jeder Seite indem man die Summe der Autoritäts-Werte aller Seiten ermittelt, die auf die Seite verweist</li> <li>2- Berechne die Autorität der Seite p durch Summierung der Hub-Werte der Seiten, die auf der Seite verweisen</li> <li>3- Normalisiere die Autoritäts-Werte indem man sie mit <math>\lambda = 1/\max</math> multipliziert (Maximalwert aller gerade neu berechneten Autoritäts-Werte)</li> </ol>
<b>Graph Exploration</b>	<p>Analyse grösser sozialer Netzwerke</p> <p>➔ Benutzung einer Adjazenzmatrix um direkte Verweise der Knoten zu verwalten (bidirectional verwaltet)</p> <p>Compressed Sparse Rows (CSR)</p> <ul style="list-style-type: none"> <li>- <i>Werte</i> (Label)</li> <li>- <i>SpaltenIndex</i></li> <li>- <i>ZeilenPtr</i> (Anzahl Einträge in Zeilen)</li> </ul> <p>(-) Nicht Update fähig</p>
<b>Graph-Mining (Zentralitätsmasse)</b>	<p>Charakterisiert ganze Graphen oder Teilstrukturen</p> <ul style="list-style-type: none"> <li>- Verbindungs-zentralität (degree centrality): wie viele ausgehenden Kanten ➔ Normierung mit dem Maximalwert für den sternförmigen Graphen (1 = Sehr ähnlich)</li> <li>- Nähe-Zentralität (closeness centrality): Distanz der Kanten (man kann auch nähere Kanten stärker gewichten)</li> <li>- Pfad-Zentralität (betweenness centrality): wie viele kürzeste Wege laufen durch den Knoten ➔ Normierung</li> </ul>
<b>Map Reduce</b>	<p>Massiv paralleler Datenverarbeitung</p> <ul style="list-style-type: none"> <li>- <i>Mapper</i></li> <li>- <i>Reducer</i></li> </ul> <p><b>Verbesserung nach Ullman</b>: Relationen werden mehrmals in Zeilen und Spalten repliziert sodass alle Join Partner nur innerhalb einer Zelle sind</p> <p><b>PigLatin</b>: Map Reduce Skriptsprache</p>
<b>Peer-to-Peer Informationssysteme</b>	<ul style="list-style-type: none"> <li>- <i>Seti@Home</i>: P2P number crunching (Suche nach außerirdischen)</li> <li>- <i>Napster</i>: P2P file sharing (stealing) ➔ Hat einen zentralen Verzeichnis mit Adressen der Peers (2-stufigen Vorgehen)</li> <li>- <i>Gnutella-Architektur</i>: verzeichnisloses File Sharing ➔ Wellenweise Suche (sehr aufwendig)</li> <li>- <i>Distributed Hash Table (DHT)</i>: basieren auf „consistent hashing“ Vollständige Dezentralisierung der Kontrolle (dennoch Zielgerichtete Suche) ➔ CHORD: Kreisförmige Architektur mit Fingertable</li> </ul>
<b>No-SQL Datenbanken</b>	<p>No-SQL Datenbanksysteme verteilen die Last innerhalb eines Clusters/Netzwerks</p> <p><b>CAP-Theorem</b>: nur 2 von 3 Wünschen erfüllbar</p> <ul style="list-style-type: none"> <li>- <i>Konsistenz</i> (wurde verzichtet)</li> <li>- <i>Zuverlässigkeit / Verfügbarkeit</i></li> <li>- <i>Partitionierungstoleranz</i></li> </ul> <p><b>Relaxiertes Konsistenzmodell</b></p> <ul style="list-style-type: none"> <li>- Replizierte Daten haben nicht alle den neuesten Zustand (kein 2-Phasen-Commit-Protokoll)</li> <li>- Transaktionen könnten veraltete Daten zu lesen bekommen</li> <li>- <i>Eventual Consistency</i>: Würde man das System anhalten, würden alle Kopien irgendwann in denselben Zustand übergehen</li> <li>- <i>Read your writes Garantie</i>: Transaktion liest auf jedem Fall eigenen Änderungen</li> <li>- <i>Monotonic Read-Garantie</i>: Transaktionen würde beim wiederholten Lesen keinen älteren Zustand als den vorher mal sichtbaren lesen</li> </ul>

**Multi-Tenancy / Cloud  
Datenbankem**

- IaaS (Infrastructure as a Service): virtuelle Maschinen zur Verfügung gestellt, auf denen dann beliebige Software installiert werden können
- PaaS (Platform as a Service): reichhaltige Schnittstellen für die Neuentwicklung von Web-Applikationen
- SaaS (Software as a Service): Systeme stellen komplexe, anwendungsspezifische Funktionalität zur Verfügung

**Multi-Tenancy Datenbankenarchitekturen**

- *Gemeinsame Maschine (shared machine)*
- *Gemeinsames Datenbanksystem (shared process)*
- *Gemeinsame Relationen (shared tables)*: Applikationssoftware muss sicherstellen, dass jede Anfrage entsprechen umgeformt wird, je nachdem welchen Benutzer sie empfangen wurde