1 – Analytische Datenbanken („Normal“ transaktional)

|  |  |  |
| --- | --- | --- |
| Anfragen | Transaktional | Analytisch |
| Fokus | Read/Write/Update/Delete | Read, periodic write |
| Transaktionsdauer/Typ | Short read/write operation | Lange Leseoperationen |
| Anfragestruktur | Einfach strukturiert | Komplex |
| Datenvolumen/Anfrage | Wenige Datensätze | Viele Datensätze |
| Datenmodell | Anfragenflexible | auswertungsbezogen |

**Datenqualität**: Glaubwürdig, Nützlich, Interpretierbar

**Datenqualitätsmängel**: Dopplung, Unterschiedliche Einheiten, Inkonsistenz

**Snowflakeschema**(Normal[isiert]) vs **Starschema** (Faktentabelle + Dimensionstabellen)

**Snowflake**: Langsam

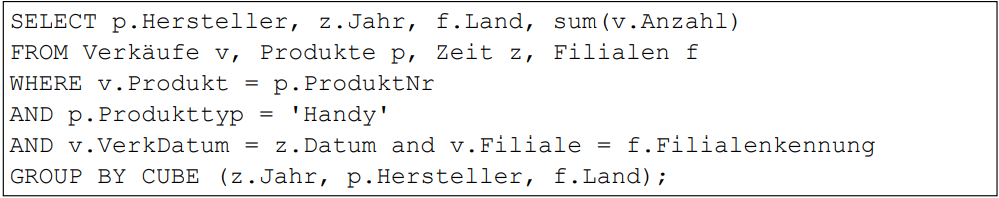
**Starschema**: Schnell, dafür Einfüge/Änderungs/- und Löschanomalien

Selecte KundeLand, ProduktLand, Umsatz. Produktland bleibt als Spalte stehen. Die Values DE, CH und AUS werden nach oben gebracht, mit ihrem jeweiligen Umsatz. SUM summiert hier nur über einen Wert (Workaround)

**Roll-Up**: Bei SELECT von Daten **wenige** Attribute in GROUP BY -> Daten verdichtet

**Drill-Down**: Bei SELECT von Daten **viele** Attribute in GROUP BY -> **hoher** detailgrad

Beispiel: Starschema, Faktentabelle Verkäufe wird mit Dimensionstabellen Produkt/Zeit/Filliale verbunden



SELECT \* FROM(

SELECT Land p, Land k, umsatz

FROM Verkauf\_3DC

WHERE jahr IS NULL

AND Land k IS NOT NULL

AND Land p IS NOT NULL

)

PIVOT(

SUM(umsatz)

FOR Land k

IN(

'Deutschland' AS 'Kunden Deutschland',

'Schweiz' AS 'Kunden Schweiz',

'Österreich' AS 'Kunden Österreich'

)

);

**create** **Table** Verkauf\_2DC

**AS**

**SELECT** **COALESCE**(p.produkt\_gruppe,'alle') **as** Gruppe, **COALESCE**(TO\_CHAR(k.kunde\_alter), 'alle')

**as** "ALTER", **SUM**(v.verkauf\_Anzahl) **as** anzahl

**FROM** Kunde k, Produkt p, Verkauf v

**WHERE** v.verkauf\_datum **>=** TO\_DATE('2020-01-01', 'YYYY-MM-DD')

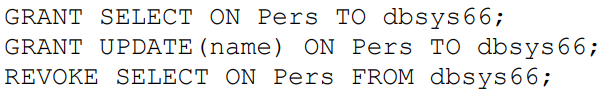
**AND** v.produkt\_ID **=** p.Produkt\_ID

**AND** v.kunde\_ID **=** k.Kunde\_ID

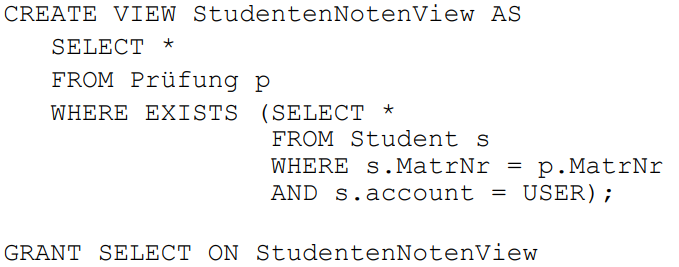
**GROUP** **BY** **CUBE** (p.produkt\_gruppe, k.kunde\_alter);

**Decode** als Alternative zu Coalesce

DECODE(GROUPING(region), 1, 'All Regions', 0, region)) AS Region

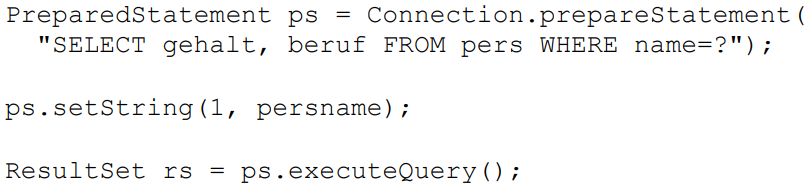
Ersteller von SQL Tabellen hat immer alle Rechte

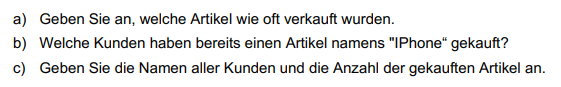
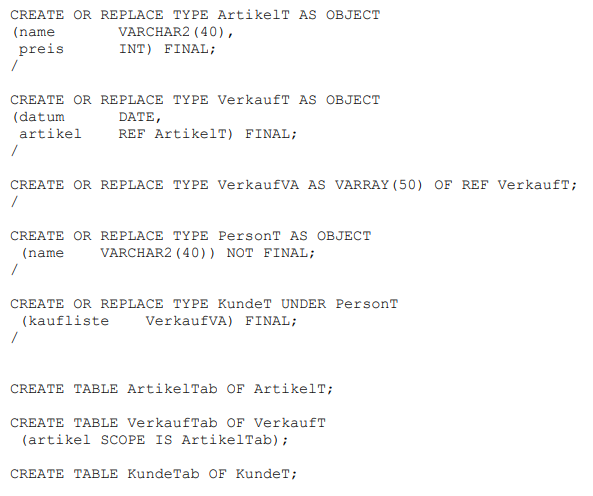
-> Software muss mit anderem Benutzer zugreifen

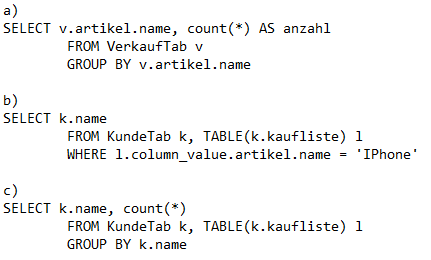


Workaround um Rechte nur auf eine Zeile zu geben:

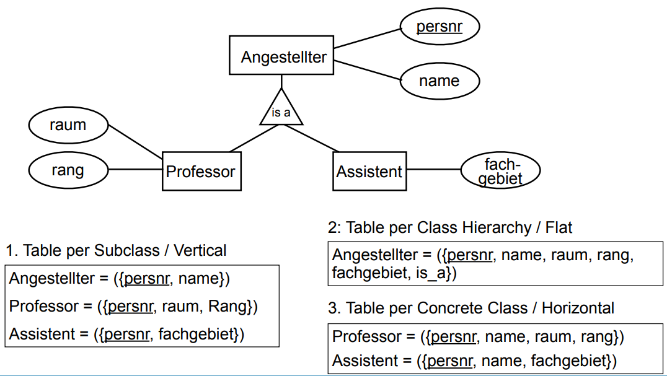
**Prepared Statements**, Input Validierung, Authentifizierung







**Kapitel 4**

**OR-Mapper**: Seperation of concerns

**Factory Method**: Genaue Objekt Instanz durch Factory gekapselt, kann ausgetauscht werden

Abbildung der Vererbung, 3 Möglichkeiten:

**Identitymap**: Keeps track of Objects that have been loaded from db.

If object is loaded again the object already in memory is used.

Metadaten für Mapping als xml Datei oder Annotation im Java Code

Vorgehensweise Hibernate: Topw-down/Bottom-up/Middle-out/Meet in the middle

**Kapitel 5**

**Data Dictionary**: Speicherung aller Informationen, die zur Verwaltung der Objekte in der Datenbank benötigt werden

TAB – SELECT \* FROM Tab; – Alle eigenen Tabellen, Views

USER\_TABLES – Tabellen eines Benutzers

USER\_TAB\_COLUMNS – Attributes einer Tabellen eines Benutzers

USER\_TRIGGERS – Trigger eines Benutzers

USER\_TAB\_PRIVS – Zugriffsrechte auf eine Tabelle eines Benutzers

USER\_VIEWS – Views eines Benutzers

– Schema-Informationen, also Tabellen, Attribute, etc.

– Statistiken

– Verwendeter Speicherplatzbedarf

– Name von Datenbank Benutzern, Rollen, Zugriffsrechte

**Kapitel 6**

**Dirty Read/Write**: Transaktion 1 liest/schreibt Daten, die von Transaktion 2 verändert aber noch nicht committet wurden => Problem bei Rollback

**Lost Update**: Lesen von Daten, die eine andere Transaktion bereits gelesen hat und im Anschluss bearbeiten möchte.

**Incorrect summary**: Erstellen einer Statistik (z.B. Gesamtsumme aller Konten) während der Daten geändert werden.

**Phantom-Problem**: Erstellen einer Statistik, während neue Datensätze eingefügt werden

**ACID**-Prinzip bei Transaktionen: Atomicity, Consistency, Isolation, Durability

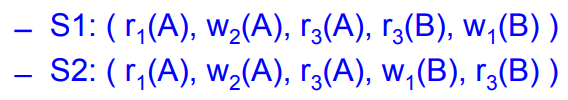
Bei Systemausfall Recovery: Wiederherstellen des jüngsten transaktionskonsistenten Zustands

**Physisches** Logging: Zustand merken (Before und After-Image)

**Logisches** Logging: Protokollierung der Änderungsoperationen mit ihren Parametern

**Seriell**: Transaktionen werden vollständig nacheinander ausgeführt.

**Serialisierbar**: Schedule liefert dasselbe Ergebnis wie mindestens ein **serieller** Schedule



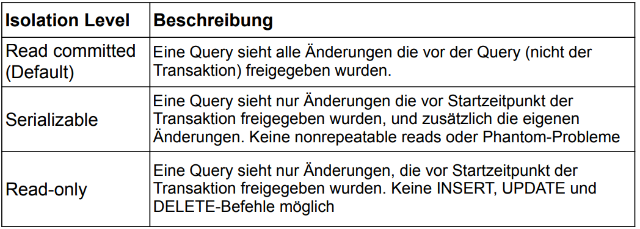
Einfügen von Kante T1→T2, falls eine Operation einer Transaktion T1 vor einer Operation einer Transaktion T2 ausgeführt werden muss (muss vorher ausgeführt werden, wenn ansonsten ein anderes Ergebnis entsteht)

**Physische Sperre**: Sperre von DB Objekt, keine Lösung zu Phantom Problem, da neu eingefügte Objekte nicht gesperrt sind

**Logischer Sperre** (Prädikatssperre): Sperren einer Menge von Tupeln(z.B. P1=(Gehalt>50000), P2=(Jahrg<1960))

* Nur bei Lesezugriff möglich, da bei überlappenden Mengen 2x Write Locks angefordert werden

Ganze Tabelle sperren:

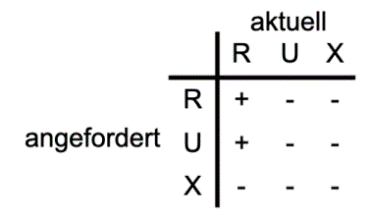
LOCK TABLE name IN EXCLUSIVE MODE; LOCK TABLE name IN SHARE MODE;

**Multiversion Concurrency Control** in Oracle:

Beim Lesen Überschriebener Daten wird durch UNDO-Daten

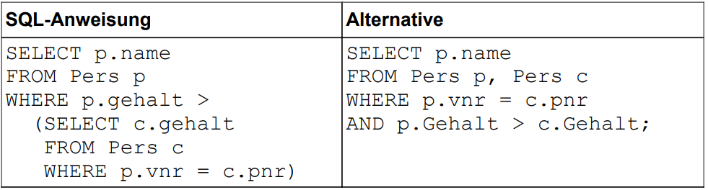
der Alte Stand wiederhergestellt/simuliert

**U**(pdate)-**Sperre** -> Kann angefordert werden wenn andere R-Locks aktiv sind.

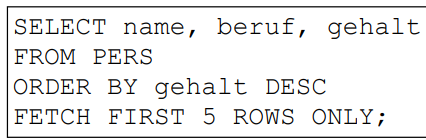
 Wenn U-Sperre vorhanden werden keine neuen R-Locks vergeben.

Wenn alle R-locks freigegeben wurden Update zur X-Sperre

**Kapitel 7**

****Korrelierte Unterabfrage

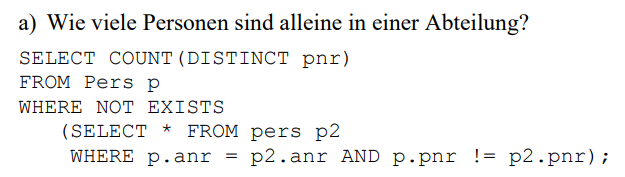
DISTINCT nur wenn notwendig

Top-N-Analyse

Ermittlung der Top-Verdiener der Firma

ALternativen:

FETCH FIRST 10 PERCENT ROWS ONLY

FETCH FIRST 3 ROWS WITH TIES

**SELECT** **COUNT**(**DISTINCT** p.pnr) **AS** AnzahlAlleineInAbteilung

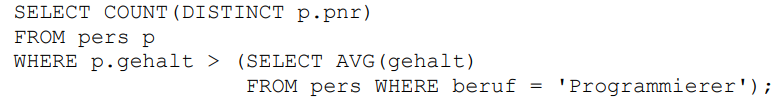
**FROM** Pers p

**LEFT** **JOIN** Pers p2 **ON** p.anr **=** p2.anr **AND** p.pnr !**=** p2.pnr

**WHERE** p2.pnr **IS** **NULL**;

**CREATE** MATERIALIZED **VIEW** ProgGehalt **AS**

**SELECT** **AVG**(gehalt) **AS** durchschnitt

 **FROM** pers

**WHERE** beruf **=** 'Programmierer';

**SELECT** **COUNT**(**DISTINCT** p.pnr)

**FROM** pers p, ProgGehalt pg

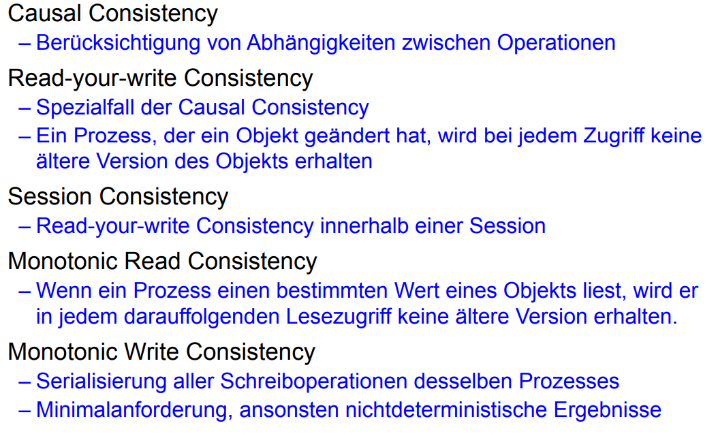
**WHERE** p.gehalt **>** pg.durchschnitt;

Hints (Auf Index): SELECT /\*+ INDEX(p Pers IndexPersGehalt) \*/ \* FROM Pers p

Clustering: Speicherung von Tabellen, die häufig verknüpft werden als physisch vorberechneter Join

Partitionierung: Aufteilung einer Tabelle in physisch mehrere Teile (z.B. Kunden Partition nach Land)

**Kapitel 8**

Vertikale Skalierung: System durch Leistungsfähigeres Ersetzen

Horizontale Skalierung: Verteilung der Arbeit durch zusätzliche Knoten

CAP – Consistency, Availability, PartitionTolerance

BASE: Basically Available, Soft State, Eventual Consistent (Auflockerung von ACID)

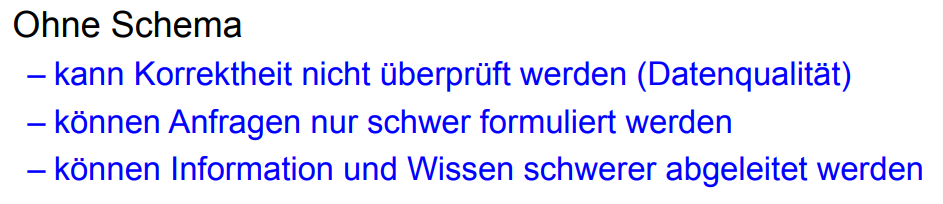
Unterschied Sharding und Partitionierung

– Sharding ist Verteilung von Daten

auf mehrere verschiedene Computer

– Partitionierung ist Verteilung von

Daten auf demselben Computer

Sharding:

Verhinderung von Bottlenecks

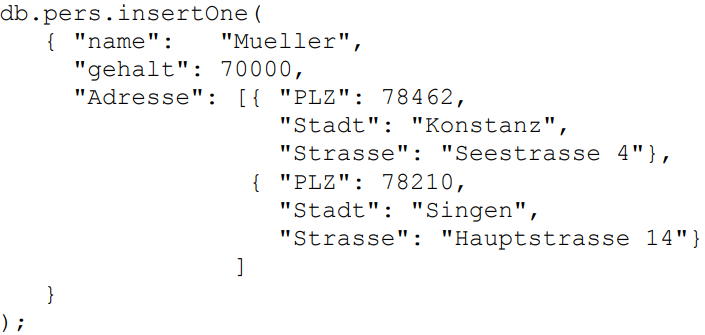
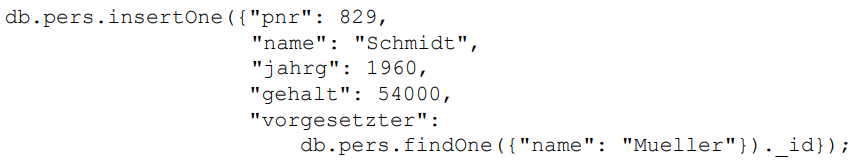
Keine single points of failure

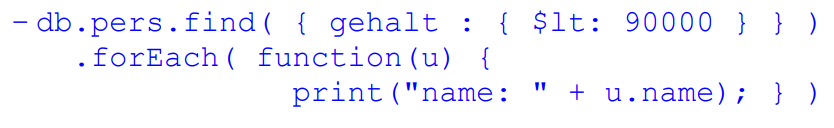
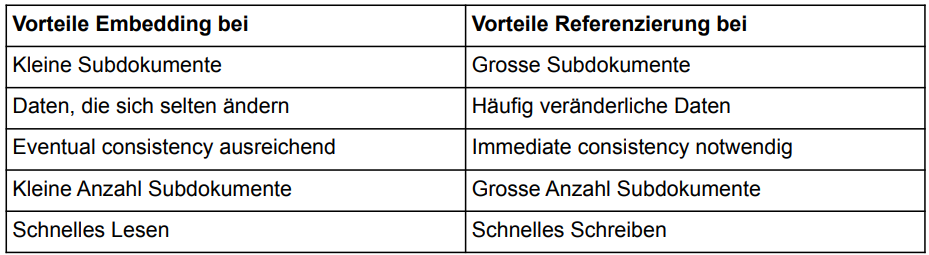
Geografische Verteilung

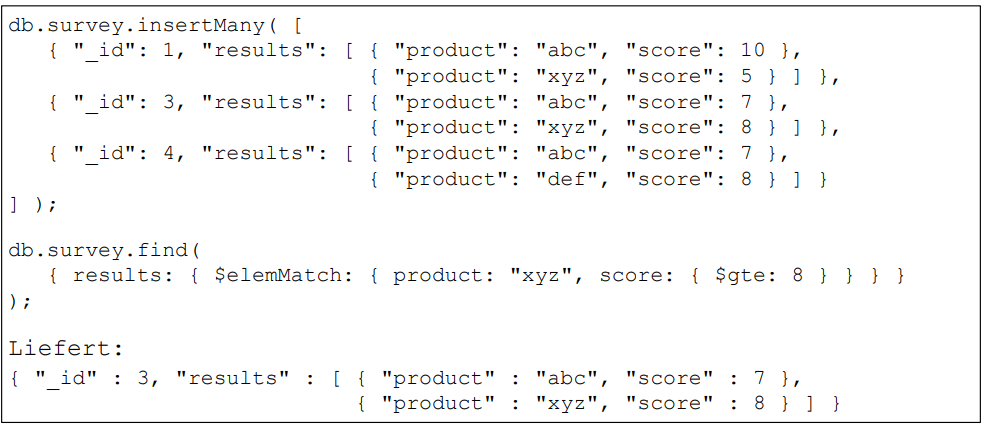
No-SQL: Key Value Stores, Spaltenorientiert, Dokumentenorientiert

Entscheidung für No-SQL: Viel temporäre, dokumentenorientierte Daten (z.B Warenkorb)

**Polyglott Persistence**: Mehrere Datenbanksysteme, jeweils für spezifischen Anwendungsbereich

**Kapitel 9**



$elemMatch: Kriterien werden auf ein Element angewendet.

Ohne würde id:1 zurückgeliefert werden, da ein Element

product:xyz und ein anderes score: {$gt:8} erfüllt

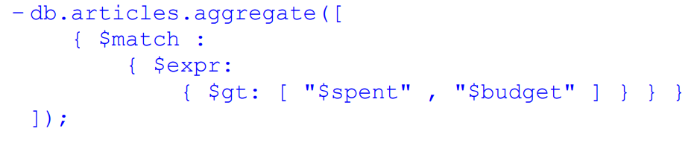
Distinct: db.student\_data.distinct("semester", { note: 1.0 })

Sort: db.pers.find().sort( { name: 1 }) -> Aufsteigend: 1, absteigend: -1

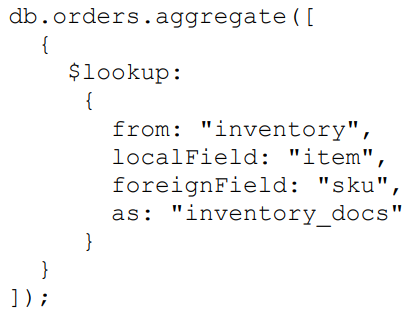
Count: db.pers.find( { name : "Müller" } ).count()

Limit: db.pers.find( { name : "Müller" } ).limit(10)

Expression vergleicht Werte von Feld spent und budget

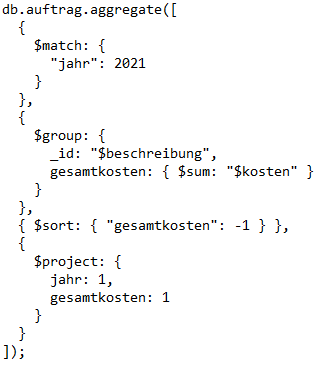
db.person.find({"geburtsdatum": new Date("2014-02-16")})

db.pers.updateOne({\_id: 100}, {$set: {gehalt: 120000}})

db.pers.deleteOne({jahrg: 1960})

Lookup von Orders(localField: item) Left Outer Join mit (from) inventory und

ForeignField: sku. As “inventory\_docs” => Neues Feld in orders, welches eingebettetes

 (gejointes) Document aus inventory enthält

Auftrag aus 2021, nach Beschreibung gruppieren, für jede group

Gesamtkosten berechnen (Sum of „kosten“), nach Gesamtkosten

db.auftrag.find({

"jahr" **:** 2021,

**$or :** [{"beschreibung":"Service"},

{"beschreibung":"Support"}]

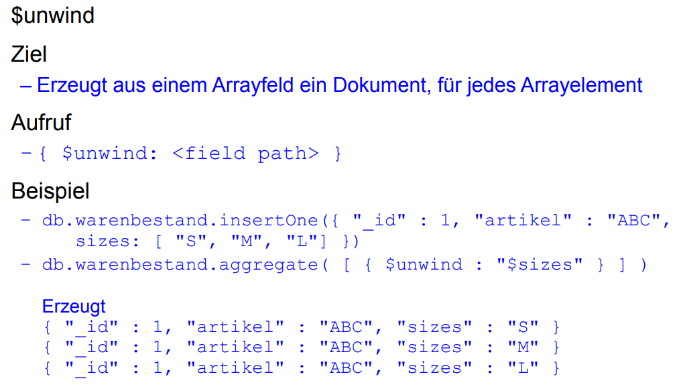
}

**{ \_id:** 0, **beschreibung:** 1 }

);

Absteigend, Jahr und Gesamtkosten weitergeben.

Da $project das letzte ist werden diese Felder ausgegeben.

Replica Sets:

Mehrere DB Server. 1 Primary Node erhält alle Schreibaufrufe

Und repliziert diese auf die Secondary Nodes. Leseaufrufe gehen

Je nach Read Preference Mode an Primary oder Secondary Node.

Wenn Primary Node ausfällt wählen Secondarys eine Neue.

Read Preference Modes:

Primary Mode

– Lesen beim Primary Node

– Falls dieser nicht verfügbar: Fehler oder Exception

PrimaryPreferred Mode

– Lesen beim Primary Node

– Falls dieser nicht verfügbar: Lesen bei Secondary Node

– maxStalenessSeconds Option: maximale Verzögerung bei Lesen,

wenn Verzögerung diesen Wert überschreitet, wird Client nicht mehr verwendet DBSYS2

Secondary Mode

– Lesen beim Secondary Node

– Falls keiner verfügbar: Fehler

– Ebenfalls Verwendung: maxStalenessSeconds

SecondaryPreferred Mode

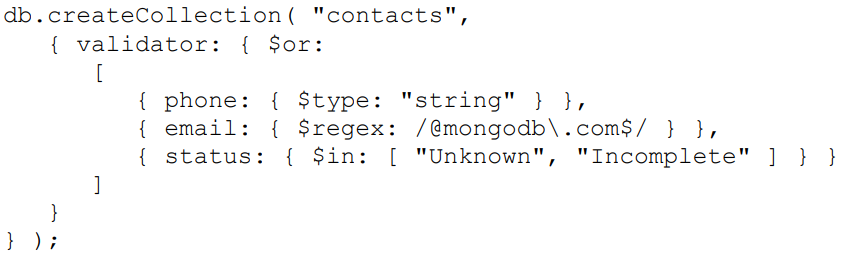
– Lesen beim Secondary Node

– Falls keiner verfügbar: Lesen beim Primary Node

Nearest Mode

– Lesen beim Knoten mit Node geringer Latenzzeit

– Primary und Secondary Nodes werden gleich verwende



No-SQL Schema Validierung (Beim Erzeugen der Collection anlegen)

Seit Version 4 sind auch MultyDocument-Transactions teilweise ACID

{$divide: [ "$kosten", "$stunden" ] }