

DBI Zusammenfassung

Martin Linhard

May 26, 2022

Contents

1	Themenkorb 1 - Konzeptionelles Datenbankdesign	5
1.1	ER-Modell	5
1.2	ER-Diagramm (ERD)	5
1.2.1	Entity Typen	5
1.2.2	Beziehungen	5
2	Themenkorb - Information Retrieval	7
2.1	SQL	7
2.1.1	Reihenfolge der Ausführung	7
2.1.2	Befehle	7
2.1.3	Wichtige Funktionen	8
2.1.4	Joins	9
2.1.5	Subselects	12
2.1.6	Andere, wichtige Keywords	13
2.1.7	Indizes	14
2.1.8	Hierarchisches SQL	15
3	Themenkorb - Relationales Datenbankmodell	17
3.1	DDL	17
3.1.1	Datentypen	17
3.1.2	Constraints	17
3.1.3	Tabellen im Nachhinein bearbeiten	18
3.2	DML	19
3.2.1	Views	19
3.2.2	Sequences	19
3.2.3	MERGE	19
3.3	Normalisierung	19
3.3.1	Normalformen	19
3.3.2	Anwendung - Zirkelbezug	21
3.3.3	Abhängigkeitsdiagramm - Beispiele	21
4	Themenkorb - Entwurfsmuster in der Datenmodellierung	23
4.1	History	23
4.1.1	History eines Attributs	23
4.1.2	History einer 1:n Beziehung	24
4.1.3	History einer n:m Beziehung	24
4.2	Supertyp/Subtyp	25
4.2.1	Wann?	25
4.3	Reflexive Beziehungen	25
4.3.1	Hierarchie	25
4.3.2	Liste	26
4.3.3	Gerichteter Graph (Netzplan)	26
4.4	Mehrwertige Beziehungen	26
5	Themenkorb - Transaktionen und Concurrency	27
5.1	Transaktionen	27
5.1.1	Allgemeines	27
5.2	Anomalien im Einbenutzerbetrieb	28

5.3	Concurrency	28
5.3.1	Serialisierbarkeit	29
5.3.2	Lösungsmöglichkeiten	29
5.3.3	Deadlocks	30
5.3.4	Re-Read Methode	30
5.3.5	U-Lock	31
5.3.6	Isolation-Levels	31
5.4	Backup & Recovery	31
5.4.1	Backup	31
5.4.2	Recovery	32
5.5	Data Control Language	33
6	Themenkorb - Datenbankarchitektur und Datenbankverwaltung	35
6.1	Datenbankarchitektur	35
6.1.1	Allgemeines	35
6.1.2	System Global Area	35
6.1.3	Prozesse	36
6.1.4	Dateien	38
6.1.5	Fehlerbehandlung	38
6.2	Datenbankverwaltung	38
6.2.1	Allgemeines - Import	38
6.2.2	SQL Loader	38
6.2.3	Weitere Tools	39
8	Themenkorb - Aktuelle Datenmodelle	41
8.1	Allgemeines	41
8.1.1	BASE	41
8.1.2	Gliederung von NO-SQL Datenbanken	41
8.2	Redis	42
8.2.1	Allgemeines	42
8.2.2	Sharding	42
8.2.3	Replication	42
8.2.4	Cluster	42
8.2.5	Befehle	43
8.3	MongoDB	44
8.3.1	Struktur / Vergleich mit rel. Datenbank	44
8.3.2	Befehle	44
8.4	Neo4J	45
8.4.1	Befehle	45

1 Themenkorb 1 - Konzeptionelles Datenbankdesign

1.1 ER-Modell

- ER \Rightarrow Entity Relationship

1.2 ER-Diagramm (ERD)

1.2.1 Entity Typen

- Fundamental \Rightarrow Unabhängig von anderen
- Attributiv \Rightarrow Abhängig von genau einer anderen Entity
- Assoziativ \Rightarrow Abhängig von mindestens 2 anderen Entities

1.2.2 Beziehungen

- 1:1
- 1:n
- n:m

Übung macht den Meister!

2 Themenkorb - Information Retrieval

2.1 SQL

2.1.1 Reihenfolge der Ausführung

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT / ORDER BY
 - Es ist hier nicht ganz klar, was zuerst ausgeführt wird!

2.1.2 Befehle

ORDER BY

- Nicht angegeben \implies Reihenfolge ist nicht garantiert!

GROUP BY

- Wenn eine "normale" Spalte neben einer Gruppenfunktion im SELECT steht, muss diese "normale" Spalte im Group By enthalten sein!
 - Das Gruppen-Statement (z.B. MAX) wird dann für jeden unterschiedlichen Wert der "normalen" Spalte ausgeführt!
 - * z.B. für jede Abteilungsnummer, wenn danach gruppiert wird!

```
SELECT deptno AS "Department", AVG(sal) "Average"
FROM emp
GROUP BY deptno;
```

HAVING

- Wird verwendet, wenn man das Ergebnis einer Gruppenfunktion als Bedingung haben möchte
 - z.B. Durchschnittsgehalt aller Jobs, die ein durchschnittliches Gehalt > 1500 haben:

```
SELECT job, ROUND( AVG(sal),2 ) "Average Salary"
FROM emp
GROUP BY job
HAVING AVG(sal) > 1500;
```

2.1.3 Wichtige Funktionen

Case / Character

- LOWER / UPPER
- INITCAP \implies Erster Buchstabe wird groß geschrieben!
- SUBSTR(string, start, length)
 - Substring ab *start* mit Länge von *length*
- LENGTH \implies Länge des Strings
- LPAD / RPAD(column, length, 'ValueUsedForPadding')
- TRIM(string) \implies Löscht Whitespaces an beiden Enden
 - TRIM(string1, string2) \implies Trimmt string2 von string1 (am Anfang und am Ende)
- REPLACE(input, toBeReplaced, replaceWith) \implies Ersetzt in Input den 2. String mit dem 3.

Number

- ROUND(number, decimalPlaces) \implies Rundet *number* auf *decimalPlaces* Nachkommastellen
- TRUNC(number, decimalPlaces) \implies Schneidet *number* nach *decimalPlaces* Stellen ab
- MOD(number1, number2) \implies $\text{number1} \% \text{number2}$

Date

- MONTHS_BETWEEN(date1, date2) \implies Anzahl der Monate dazwischen
- ADD_MONTHS(date, numberOfMonths) \implies Fügt *numberOfMonths* Monate zu *date* hinzu
- NEXT_DAY(date, 'Day') \implies Gibt den nächsten Wochentag *nach* diesem Datum mit dem gewählten Namen zurück
- ROUND(date, ['MONTH' — 'YEAR'])
 - Rundet Auf das nächste / vorherige Jahr / Monat auf / ab
- TRUNC(date, ['MONTH' — 'YEAR'])
 - Setzt das Datum auf den 1. des Monats / Jahres

Conversion

- TO_CHAR(columnWithDate — columnWithNumber, 'Format')
- TO_NUMBER(input, 'Format')
 - String zu Zahl parsen
- TO_DATE()
 - String zu Datum parsen

YYYY	Full year in numbers
YEAR	Year spelled out
MM	Two-digit value for month
MONTH	Full name of the month
MON	Three-letter abbreviation of the month
DY	Three-letter abbreviation of the day of the week
DAY	Full name of the day of the week
DD	Numeric day of the month

HH24:MI:SS AM	15:45:32 PM
DD "of" MONTH	12 of October

DDspth	FOURTEENTH
Ddspth	Fourteenth
ddspth	fourteenth
DDD or DD or D	Day of year, month or week

Multi row

- MAX, MIN
- COUNT
- AVG
- SUM
- (STDDEV, VARIANCE)

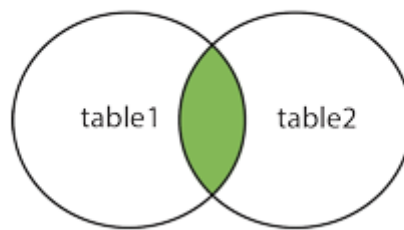
2.1.4 Joins

- Entweder mit ON oder mit USING
 - INNER JOIN DEPT D ON EMP.DEPTNO = D.DEPTNO; \implies Beide Spalten werden ausgegeben!
 - INNER JOIN DEPT D USING(DEPTNO); \implies Spalte muss in beiden Tables gleich heißen, wird nur 1x ausgegeben!

INNER JOIN

- Inkludiert nur Zeilen, die beiden Tables gleich sind!

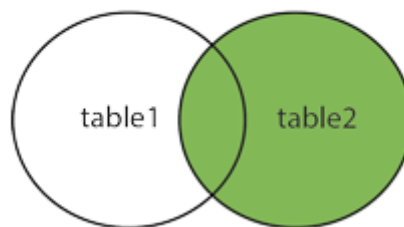
INNER JOIN



RIGHT OUTER JOIN

- Inkludiert alle Zeilen der rechten Tabelle (= die Tabelle, auf die gejoint wird) und Werte, die in beiden Tabellen gleich sind

RIGHT JOIN



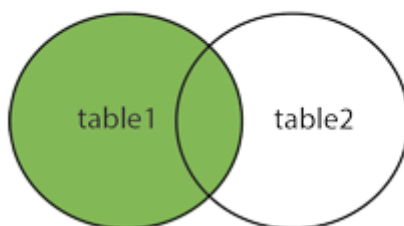
– Beispiel: Gib jene Abteilungen aus, die keine Mitarbeiter haben:

```
SELECT DISTINCT d.*  
FROM emp e  
RIGHT OUTER JOIN dept d ON e.DEPTNO = d.DEPTNO  
WHERE e.DEPTNO IS NULL;
```

LEFT OUTER JOIN

- Inkludiert alle Zeilen der linken Tabelle (= die Tabelle, von der weg gejoint wird) und Werte, die in beiden Tabellen gleich sind

LEFT JOIN



– Beispiel: Gib jene Abteilungen aus, die keine Mitarbeiter haben:

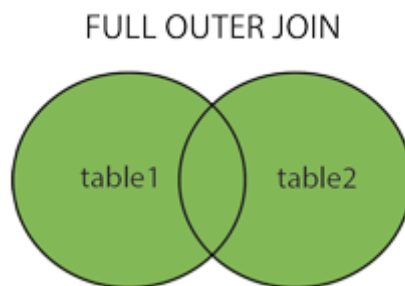
```

SELECT DISTINCT d.*
FROM dept d
LEFT OUTER JOIN emp e ON e.deptno = d.deptno
WHERE e.deptno IS NULL;

```

FULL OUTER JOIN

- Inkludiert alle Zeilen der linken Tabelle (= die Tabelle, von der weg gejoint wird) und alle Werte aus der rechten Tabelle



– Beispiel: Gib alle Mitarbeiter und Abteilungen aus

```

SELECT e.ename, d.deptno
FROM emp e
FULL OUTER JOIN dept d ON e.deptno = d.deptno;

```

CROSS JOIN

- Gibt jede Zeile in einer Tabelle mit jeder Zeile aus einer anderen aus
- Problem: Auch jede Zeile mit sich selbst!

```

SELECT a.teamname, b.teamname, c.teamname
FROM teamA a
CROSS JOIN teamB b
CROSS JOIN teamC c;


```

SELF JOIN

- Es wird nochmal auf den gleichen Table gejoint (z.B. um den Vorgesetzten zu bestimmen)

NATURAL JOIN

- Spalten, die beide Tabellen beinhalten werden nur 1x zurückgegeben!
- "Automatischer Inner Join" \implies Es werden nur Spalten zurückgegeben, die den gleichen Wert haben (kein NULL!)
- Es wird AUF ALLE GLEICH BENANNTEN SPALTEN IN BEIDEN TABELLEN gejoint!
 - Wenn eine neue Spalte hinzugefügt wird, welche zufällig so wie eine existierende heißt, werden nur Werte zurückgegeben, bei denen diese Spalten übereinstimmen!



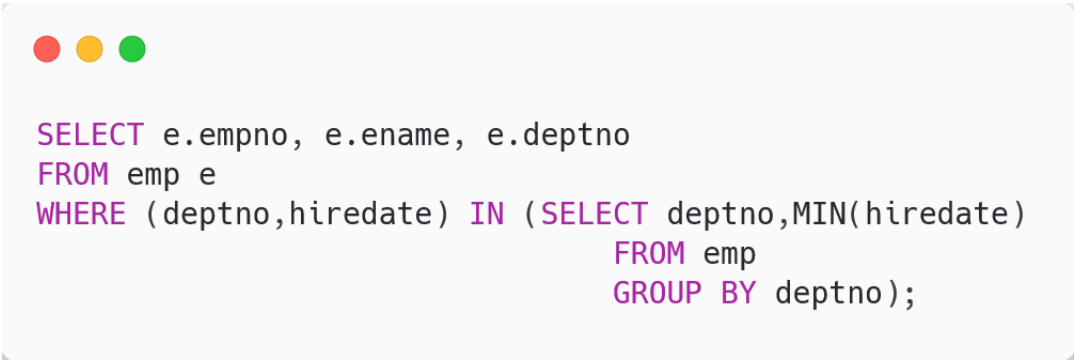
```
SELECT e.ename, d.loc
FROM emp e
NATURAL JOIN dept d;
```

EQUI / NON-EQUI Joins

- EQUI \implies =
- NON-EQUI (THETA) \implies Alles andere (Größer / Kleiner, Between and...)

2.1.5 Subselects

- Können in der WHERE, HAVING und FROM Klausel vorkommen
- Kann kein ORDER BY beinhalten
- Können eine (Single Row) oder mehrere (Multi-Row) Zeilen zurückliefern
 - Single Row \implies =, <, >, ...
- Wenn mehrere Werte aus dem Subselect zurückgegeben werden \implies IN muss verwendet werden:



```
SELECT e.empno, e.ename, e.deptno
FROM emp e
WHERE (deptno,hiredate) IN (SELECT deptno,MIN(hiredate)
                           FROM emp
                           GROUP BY deptno);
```

Multiple-Row Subselects


- Es müssen spezielle Operatoren verwendet werden:
 - IN \implies Es werden nur Zeilen zurückgegeben, dessen Wert in der Ergebnisliste des Subselects enthalten ist.

- ANY/SOME \implies Ein Wert muss $=$, $<$, $>$ als irgendein Wert in der Ergebnisliste sein
- ALL \implies Ein Wert muss $=$, $<$, $>$ als alle Werte in der Ergebnisliste sein
- Correlation \implies Es werden Werte von "Außen" in einer Subquery verwendet

2.1.6 Andere, wichtige Keywords

UNION


- Der Output von 2 SQL-Statements kann verbunden werden
- **UNION ALL** \implies Macht das gleiche, doppelte Werte werden allerdings angezeigt!
- Wichtig: Anzahl der Spalte + Datentypen müssen gleich sein, doppelte Werte werden ignoriert!



```
SELECT date
FROM store_info
UNION
SELECT date
FROM internet_sales;
```

INTERSECT


- Gibt nur Werte aus, die in beiden Statements vorhanden sind!



```
SELECT date
FROM store_info
INTERSECT
SELECT date
FROM internet_sales;
```

MINUS

- Gibt nur Werte aus, die in dem ersten Statement, nicht aber in dem 2. vorkommen!



```
SELECT date
FROM store_info
MINUS
SELECT date
FROM internet_sales;
```

2.1.7 Indizes

- Kann auf eine / mehrere (Composite Index) Spalten gleichzeitig angelegt werden
- Enthält den Wert + die zugehörige Spalte
- Muss bei jedem Insert / Delete / Update neu erstellt werden

Wann?


- Es werden aus einem großen Table nur wenige Ergebnisse erwartet
- Die Spalte enthält häufig NULL Werte

Wann nicht?

- Wenn die Tabelle oft bearbeitet / selten verwendet wird
- Wenn häufig mehr als 2-4% der Tabelle ausgegeben werden

Function based

- Die Werte im Index werden durch Funktionen berechnet:




```
CREATE INDEX upper_last_name_idx  
ON employees (UPPER(last_name));
```

- Es können auch selbst geschriebene Funktionen verwendet werden, diese müssen allerdings als "deterministic" markiert werden


Erstellen & Löschen

- Erstellen



```
CREATE INDEX index_name  
ON table_name(column...,column)
```

- Löschen



```
DROP INDEX upper_last_name_idx;
```

2.1.8 Hierarchisches SQL

- Parent \implies Wert über einer Node
- Child \implies Wert unter einer Node
- Sibling \implies Wert auf der gleichen Höhe
- Leaf \implies Node ohne Child

Abfragen

- Pseudospalten
 - LEVEL \implies Level ab Root (hat Level 1)
 - CONNECT_BY_ISCYCLE \implies Gibt 1 zurück, wenn das Element Grund für einen Loop ist (letzter in der Hierarchie, bevor es von vorne los geht!)
 - CONNECT_BY_ISLEAF \implies Gibt 1 zurück, wenn das Element ein Leaf ist
- Funktionen
 - SYS_CONNECT_PATH(column, char) \implies Pfad des Elements von der Root Node weg, getrennt durch *char*
- Operatoren
 - SYS_CONNECT_BY_ROOT \Leftarrow Gibt den Wert der Spalte der Root Node zurück
 - PRIOR \Leftarrow Um Parent Nodes zu verbinden
- Clauses
 - START WITH *condition* \implies Auswahl der Root-Zeile
 - CONNECT BY ...PRIOR \implies Gibt Verbindung zwischen Parent und Child an (mit PRIOR kann auf den Parent zugegriffen werden)
 - ORDER SIBLINGS BY \implies Sortiert die Siblings des Parents nach einer Spalte



```
SELECT e.ename, PRIOR e.ENAME, SYS_CONNECT_BY_PATH(e.ENAME, '/'), LEVEL
FROM EMP e
WHERE LEVEL >= 2
START WITH e.MGR IS NULL
CONNECT BY PRIOR e.EMPNO = e.MGR
ORDER SIBLINGS BY e.ENAME;
```


3 Themenkorb - Relationales Datenbankmodell

3.1 DDL

3.1.1 Datentypen

- CHAR(n) \implies Fixed-length, Rest wird mit Leerzeichen aufgefüllt bzw. abgeschnitten!
- VARCHAR(n) \implies Variable Länge (max. n)
- Date
- Timestamp
- NUMBER(s, p) \implies Einzige Zahlen-Datentyp in Oracle: s gibt die Gesamtstellen an, p die nach dem Komma
 - NUMERIC, DECIMAL sind nur die ANSI Name für diese Datentypen!
 - Float / Real / Double Precision steht in den Docs zwar als Subtyp von Number, wird aber (im Unterschied zu NUMERIC...) als FLOAT in Describe angezeigt!

3.1.2 Constraints

- NOT NULL \implies Null-Werte sind nicht erlaubt
- UNIQUE \implies Der Wert muss innerhalb der Spalte einzigartig sein
- PRIMARY KEY
 - Sofort nach dem Attribut, wenn er nur aus einem Attribut besteht
 - Am Ende des Tables, wenn er aus mehreren Attributen besteht!

```
CREATE TABLE bookLending
(
  isbn INTEGER,
  lendingDate DATE,
  CONSTRAINT pk_bookLending PRIMARY KEY (isbn, lendingDate)
);
```

- FOREIGN KEY
 - Am Ende des Tables

```
CREATE TABLE Orders
(
  O_Id INTEGER PRIMARY KEY,
  P_Id INTEGER,
  CONSTRAINT fk_PerOrder FOREIGN KEY (P_Id)
  REFERENCES Person(P_Id)
);
```

- CHECK \Rightarrow Um sicherzustellen, dass ein Wert ein gewisses Kriterium erfüllt

```
CREATE TABLE Persons
(
  P_Id INT NOT NULL,
  sal NUMBER,
  CONSTRAINT chk_Person CHECK (P_Id>0 AND sal > 0)
);
```

- DEFAULT \Rightarrow Default-Wert, falls dieser beim Insert weggelassen wird

3.1.3 Tabellen im Nachhinein bearbeiten

- Vor allem bei FKs relevant, da dann nicht mehr auf die Reihenfolge von Tabellen geachtet werden muss!
- Es können Constraints & Spalten bearbeitet werden!
 - Constraints

```
ALTER TABLE Orders
ADD CONSTRAINT fk_PerOrder FOREIGN KEY(P_Id)
REFERENCES Person(P_Id);
```

- Spalten

```
ALTER TABLE TABLE_NAME
ADD column_name datatype;
```

```
ALTER TABLE TABLE_NAME
RENAME COLUMN column_name TO new_column_name;
```

```
ALTER TABLE TABLE_NAME
MODIFY column_name NEW DATA TYPE;
```

- Es können sowohl einzelne Constraints, als auch Columns und Tables gedroppt werden!
 - Beim Droppen von Tables empfiehlt es sich, vorher die Foreign-Key-Constraints zu entfernen, damit im Falle von Cascade Constraints keine Daten aus Versehen gelöscht werden!

3.2 DML

3.2.1 Views

- Sind abgespeicherte Select-Statements

3.2.2 Sequences

- Erstellen
- Beim Inserten \Rightarrow sequence.NEXTVAL

```
CREATE SEQUENCE seqOne
START WITH 100
INCREMENT BY 1
[MAXVALUE 1000]
[CYCLE]
```

3.2.3 MERGE

- Inserted ein Item, falls das gesuchte nicht gefunden wurde
- Updated ein existierendes Item, falls es gefunden wurde

3.3 Normalisierung

3.3.1 Normalformen

Nullte Normalform

- Mehrere Werte stehen in einer Zeile:

PersNr	Name	Vorname	AbtNr	Abteilung	ProjektNr	Beschreibung	Zeit
1	Lorenz	Sophia	1	Personal	2	Verkaufspromotion	83
2	Hohl	Tatjana	2	Einkauf	3	Konkurrenzanalyse	29
3	Willschrein	Theodor	1	Personal	1,2,3	Kundenumfrage, Verkaufspromotion, Konkurrenzanalyse	140, 92, 110
4	Richter	Hans-Otto	3	Verkauf	2	Verkaufspromotion	67
5	Wiesenland	Brunhilde	2	Einkauf	1	Kundenumfrage	160

Erste Normalform

- Jede Zeile enthält nur einen Wert
- Es muss ein Primary Key gefunden werden (unterstreichen!), welcher jede **Zeile** eindeutig kennzeichnet!

<u>PersNr</u>	Name	Vorname	<u>AbtNr</u>	Abteilung	<u>ProjektNr</u>	Beschreibung	Zeit
1	Lorenz	Sophia	1	Personal	2	Verkaufspromotion	83
2	Hohl	Tatjana	2	Einkauf	3	Konkurrenzanalyse	29
3	Willschrein	Theodor	1	Personal	1	Kundenumfrage	140
3	Willschrein	Theodor	1	Personal	2	Verkaufspromotion	92
3	Willschrein	Theodor	1	Personal	3	Konkurrenzanalyse	110
4	Richter	Hans-Otto	3	Verkauf	2	Verkaufspromotion	67
5	Wiesenland	Brunhilde	2	Einkauf	1	Kundenumfrage	160

Zweite Normalform

- Die Relation befindet sich in der 1. Normalform + jedes Attribut ist vom Gesamtschlüssel der Relation abhängig, und nicht nur von einem Teil!
- Praxis: Ursprüngliche Tabelle in mehrere unterteilen, sodass oben genannte Anforderungen erfüllt sind!
 - Diese Tables dürfen nur Attribute enthalten, die vom gesamten PK abhängig sind \Rightarrow Es kann sein, dass eine Relation 2 Primary Key Attribute benötigt!

Relation Projekt

<u>ProjektNr</u>	Beschreibung
2	Verkaufspromotion
3	Konkurrenzanalyse
1	Kundenumfrage

Relation Personal

<u>PersNr</u>	Name	Vorname	<u>AbtNr.</u>	Abteilung
1	Lorenz	Sophia	1	Personal
2	Hohl	Tatjana	2	Einkauf
3	Willschrein	Theodor	1	Personal
4	Richter	Hans-Otto	3	Verkauf
5	Wiesenland	Brunhilde	2	Einkauf

Relation Firma

<u>PersNr</u>	<u>ProjektNr</u>	Zeit
1	2	83
2	3	29
3	1	140
3	2	92
3	3	110
4	2	67
5	1	160

Dritte Normalform

- Die Relation befindet sich in der 2. Normalform + Kein Attribut ist von einem anderen Nicht-Schlüssel-Attribut abhängig!

Relation Projekt

<u>ProjektNr</u>	Beschreibung
2	Verkaufspromotion
3	Konkurrenzanalyse
1	Kundenumfrage

Relation Personal

<u>PersNr</u>	Name	Vorname	<u>AbtNr.</u>
1	Lorenz	Sophia	1
2	Hohl	Tatjana	2
3	Willschrein	Theodor	1
4	Richter	Hans-Otto	3
5	Wiesenland	Brunhilde	2

Relation Firma

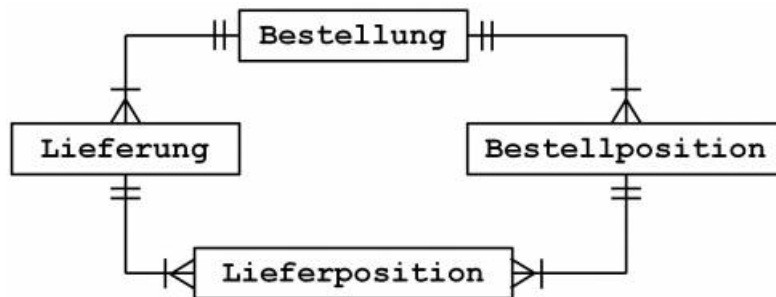
<u>PersNr</u>	<u>ProjektNr</u>	Zeit
1	2	83
2	3	29
3	1	140
3	2	92
3	3	110
4	2	67
5	1	160

Relation Abteilung

<u>AbtNr.</u>	Abteilung
1	Personal
2	Einkauf
3	Verkauf

3.3.2 Anwendung - Zirkelbezug

- Problem des Zirkelbezugs
 - Eine Entity kann von einer Ausgangsentity auf 2 verschiedene Wege erreicht werden:



```

Bestellung(BestellNr)
Lieferung(LieferNr, BestellNrFK)
Bestellposition(PositionsNr, BestellNrFK)
Lieferposition(LieferNr, BestellNr, PositionsNr, BestellNrFK)
  
```

- Je nachdem, ob über die Lieferung oder die Bestellposition auf die Bestellung zugegriffen wird, kann es zu unterschiedlichen Ergebnissen kommen!
- Lösung: Die doppelten Attribute werden zu einem zusammengezogen und in einen Foreign Key verpackt:

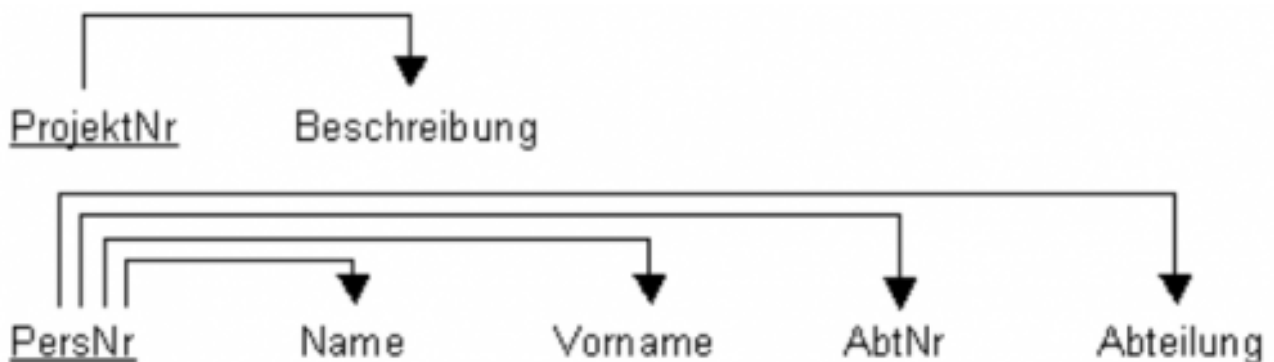
Solution:

```

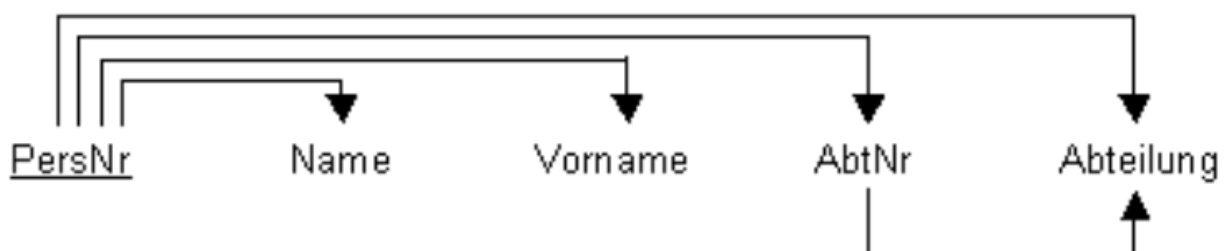
Lieferposition((LieferNr, PositionsNr, BestellNr)FK)
  
```

3.3.3 Abhängigkeitsdiagramm - Beispiele

2. Normalform



3. Normalform



4 Themenkorb - Entwurfsmuster in der Datenmodellierung

4.1 History

- Um die Werte eines Attributs nachvollziehbar zu machen
- z.B. bei Preisen, Mitarbeitergehältern...
- Es gibt lückenlose & lückenhafte Histories:
 - lückenlos \implies nur ein Datumswert(von / bis); muss teil des Primary-Keys sein
 - lückenhaft \implies zwei Datumswerte; einer muss teil des Primary-Keys sein

4.1.1 History eines Attributs

Allgemein

- z.B. um den Preis eines Produktes nachvollziehbar zu machen
- Es entsteht eine Extra-Entity mit folgenden Attributen (**PK**)
 - **Foreign Key** auf die Ursprungsentity
 - **GuelteigAb**
 - Tatsächlicher Wert (gleicher Datentyp wie im Ausgangsmodell)

Abfragen

Wert zu einem bestimmten Zeitpunkt bzw. aktueller Wert

- Um den aktuellen Preis zu bestimmen, muss die Datums-Klausel einfach "SYSDATE" enthalten.

```
SELECT Preis
FROM Artikelpreis
WHERE ArtikelID = 1 AND GuelteigAb =
    ( SELECT MAX(GuelteigAb)
      FROM Artikelpreis
      WHERE ArtikelID = 1 AND
        -- Bzw. <= SYSDATE
        GuelteigAb <= TO_DATE('12.02.2013','DD.MM.YYYY')
    );
```

4.1.2 History einer 1:n Beziehung

Allgemein

- z.B. um nachzuvollziehen, welcher Mitarbeiter wann in welcher Abteilung gearbeitet hat
- Es entsteht eine N:M Beziehung mit folgenden Attributen (**PK**)
 - Foreign-Key auf die fundamentale Entity
 - **Foreign-Key auf die attributive Entity**
 - **GueligAb**

Abfragen

Wert zu einem bestimmten Zeitpunkt bzw. aktueller Wert

- Um aktuelle Abteilung zu bestimmen, muss die Datums-Klausel einfach "SYSDATE" enthalten.

```
SELECT a.abteilung_name
FROM Mitarbeiter m
INNER JOIN MITARBEITERABTEILUNG Ma on m.mitarbeiter_id = Ma.MITARBEITER_ID
INNER JOIN ABTEILUNG A on Ma.abteilung_id = A.ABTEILUNG_ID
WHERE m.mitarbeiter_id = 2 AND ma.guelig_ab =
  (SELECT MAX(guelig_ab)
   FROM MitarbeiterAbteilung ma2
   WHERE ma2.mitarbeiter_id = m.mitarbeiter_id AND Ma.guelig_ab <= SYSDATE);
```

4.1.3 History einer n:m Beziehung

- z.B. um nachzuvollziehen, welcher Mitarbeiter wann an welchem Projekt gearbeitet hat
- Der bestehende Table wird um zwei Daten (von, bis) erweitert (**PK**)
 - **Foreign Key 1**
 - **Foreign Key 2**
 - **GueligAb**
 - GueligBis

Abfragen

```
SELECT a.abteilung_name
FROM Mitarbeiter m
INNER JOIN MITARBEITERABTEILUNG Ma on m.mitarbeiter_id = Ma.MITARBEITER_ID
INNER JOIN ABTEILUNG A on Ma.abteilung_id = A.ABTEILUNG_ID
WHERE m.mitarbeiter_id = 1 AND ma.guelig_ab =
  (SELECT MAX(guelig_ab)
   FROM MitarbeiterAbteilung ma2
   WHERE ma2.mitarbeiter_id = m.mitarbeiter_id
   AND Ma.guelig_ab <= SYSDATE AND ma.GUELTIG_BIS >= SYSDATE);
```


4.2 Supertyp/Subtyp

4.2.1 Wann?

- Wenn zwei Entities einige Attribute gemeinsame haben, sich aber auch in einigen unterscheiden
- Beispiel: Lehrer & Schüler
 - Beide haben Eigenschaften einer jeden **Person** (Vorname, Nachname)
 - Schüler haben außerdem eine Klasse, Lehrer ein Kürzel!
- Lösung: Es werden 3 Tabellen erstellt (Person, Schüler, Lehrer); der Primary Key in Schüler / Lehrer ist gleichzeitig ein Foreign Key auf die Person!
- Nur dann sinnvoll, wenn es eine endliche Anzahl an Subtypen gibt, sonst sind dynamische Eigenschaften (—Eine Entity hat Liste aus Eigenschaften, diese wiederum einen Wert für eine konkrete Entity) sinnvoller!

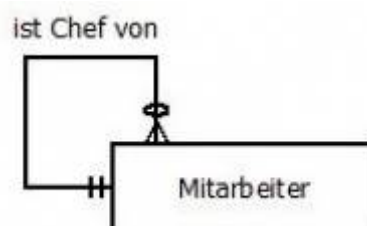
4.3 Reflexive Beziehungen

4.3.1 Hierarchie

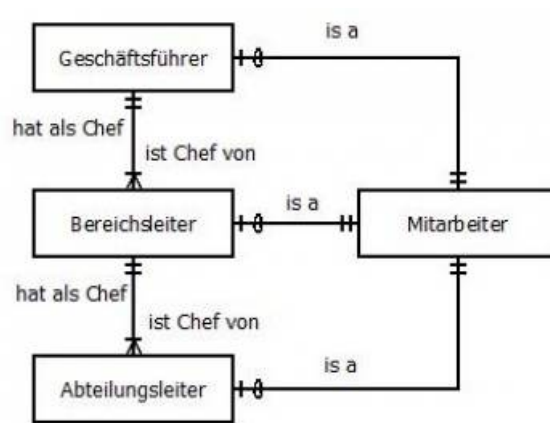
- Monohierarchie \implies ein Parent
- (Polyhierarchie \implies ggf. mehrere Parents)

Varianten

- Variante 1 und 2
 - Alle Ebenen haben identische Attribute
 - Tabelle enthält einen Foreign Key auf sich selbst
 - Je nach Umständen (Was ist Regel, was ist Ausnahme?) kann dieser Foreign Key optional (Variante 1) oder required (Variante 2) sein



- Variante 3
 - Die Ebenen haben verschiedene Attribute \implies Extra Table für jede Stufe, welcher einen Foreign Key auf den Parent beinhaltet
 - Problem: Anzahl an Ebenen ist fix vorgegeben
- Variante 4
 - Die Ebenen haben teilweise verschiedene Attribute \implies Extra Table für jede Stufe, welcher einen Foreign Key auf den Parent beinhaltet + Supertyp für die gemeinsamen Attribute
 - Problem: Anzahl an Ebenen ist fix vorgegeben



4.3.2 Liste

- Gleich wie eine Hierarchie, nur dass der Foreign Key **unique** ist (auf eine Task kann nur eine folgen bzw. kann nur eine davor kommen!)
- Abfragen sind auch hier mittels hierarchischem SQL möglich!

4.3.3 Gerichteter Graph (Netzplan)

- Eine Ausgangsentity (z.B. Stadt) + einen Verbindungstable (von, nach) mit 2 Foreign Keys auf Ausgangsentity
- Bidirektional \implies View mithilfe von Union Erstellen, welcher von & nach umdreht!
- Reflexive N:M Beziehung!

4.4 Mehrwertige Beziehungen

- Wenn 3 fundamentale Entities in einem Satz vorkommen: Ein **Lehrer** unterrichtet eine **Klasse** in einem bestimmten **Fach**.
- Wenn viele N:M Beziehungen vorhanden sind
- Lösung: Eine verbindende Entity (z.B. Unterricht), welche mindestens 2 Foreign Keys im PK enthält
 - Je nach Gestaltung des PKs können unterschiedliche Regeln festgelegt werden (Ein Lehrer darf eine Klasse nur in einem Fach unterrichten...)

5 Themenkorb - Transaktionen und Concurrency

5.1 Transaktionen

”Eine Folge von Datenbankanweisungen, welche entweder ganz oder garnicht ausgeführt wird.”

5.1.1 Allgemeines

ACID Prinzip

- Atomicity \implies Transaktion ist die kleinste Arbeitseinheit, sie wird entweder ganz oder garnicht ausgeführt
- Consistency \implies Die Datenbank ist zu Beginn und Ende jeder Transaktion konsistent
- Isolation \implies Änderungen innerhalb einer Transaktion sind nur für diese sichtbar!
- Durability \implies Nach Beendigung einer Transaktion (successful commit) sind die Daten dauerhaft, auch im Fehlerfall, gespeichert.

Commit und Rollback

- Commit \implies Transaktion wird beendet, Änderungen werden dauerhaft gespeichert!
 - Änderungen sind nun für alle sichtbar!
- Rollback \implies Änderungen seit dem letzten Commit werden verworfen!
- AutoCommit \implies Nach jeder Anweisung wird ein Commit ausgeführt, sofern die Anweisung erfolgreich ausgeführt wurde
 - Nicht erfolgreich \implies Automatisches Rollback!
 - Modus wird deaktiviert, wenn explizite / implizite Transaktion gestartet wird!

DDL Statements - Implicit Commit

- Achtung: Sämtliche DDL Statements (Create Table...) führen automatisch zu einem Commit!
 - Zuvor ausgeführte Änderungen werden zuerst committed, DDL-Statements dann in einer neuen Transaktion!

Länge von Transaktionen

- so kurz als möglich, da:
 - Tabellen nicht so lang gesperrt bleiben müssen
 - Weniger Statements im Fehlerfall wiederholt werden müssen
 - Allgemein weniger Overhead entsteht!
- so lang als notwendig, damit die Daten konsistent sind!

5.2 Anomalien im Einbenutzerbetrieb

- Es kann beim Einfügen, Updaten und Löschen zu Problemen kommen, wenn die Daten nicht in die 3. Normalform gebracht wurden!

5.3 Concurrency

Lost Update

- Eine Transaktion überschreibt die Änderungen einer anderen:
- Es wird der Wert ausgelesen, bevor die 2. Transaktion beginnt!

Zeit	Transaktion 1	Transaktion 2
1	var a = read(Kontostand von Konto 1)	
2	a = a - 400;	
3		var b = read(Kontostand von Konto 1)
4		b = b + 2000;
5		write(Konto 1, b)
6		commit
7	write(Konto 1, a)	
8	commit	

Dirty Read

- Kommt nur in Zusammenhang mit Rollback vor!
- Eine Transaktion liest Werte von einer anderen, welche im Nachhinein wieder rückgängig (Rollback) gemacht wird!

Zeit	Transaktion 1	Transaktion 2
1		var b = read(Gehalt Mitarbeiter 1)
2		b = b + 400;
3		write(Gehalt Mitarbeiter 1, b)
4	var a = read(Gehalt Mitarbeiter 1)	
5	a = a * 2	
6	write(Gehalt Mitarbeiter 2, a)	
7	commit	
8		rollback

Non-Repeatable Read

- Entsteht dann, wenn lesende Vorgänge von einer anderen Transaktion unterbrochen werden!
- Beim nächsten Read liefert die Abfrage dann andere Ergebnisse, da hier keine 2. Transaktion "dazwischenpfuscht"!

Zeit	Transaktion 1 (Summenberechnung)	Transaktion 2 (Abbuchung)
1	var a = read(Kontostand von Konto 1)	
2	summe = summe + a;	
3		var b = read(Kontostand von Konto 1)
4		b = b - 8000;
5		write(Konto 1, b)
6		var c = read(Kontostand von Konto 2)
7		c = c - 6000;
8		write(Konto 2, c)
9		commit
10	var d = read(Kontostand von Konto 2)	
11	summe = summe + d;	
12	commit	

Phantom

- Kommt meist im Zusammenhang mit Aggregatfunktionen vor, wenn sich z.B. durch eine andere Transaktion die Anzahl an Record ändert!

Zeit	Transaktion 1 (Bonus)	Transaktion 2 (Neues Konto)
1	var kontenanz = SELECT COUNT(*) FROM konto	
2		INSERT INTO konto (kontonr, kundenid, betrag) VALUES (123, 91, 0)
3		commit
4	UPDATE konto SET betrag = 313373 / kontenanz	
5	commit	

5.3.1 Serialisierbarkeit

- Als serialisierbar wird ein Ausführungsplan (Gibt an, welche Transaktion ausgeführt wird) dann bezeichnet, wenn das Ergebnis das selbe als jenes eines seriellen Ablaufes ist
- Überprüfung von Serialisierbarkeit \implies Es wird ein Graph mit allen Operation aufgebaut; wenn dieser keinen Cycle enthält \implies Serialisierbar (In gewisser Reihenfolge)!

5.3.2 Lösungsmöglichkeiten

Sperrverfahren

- Pessimistisch
 - Es wird davon ausgegangen, dass Konflikte auftreten \implies Objekte werden von Anfang an gesperrt
- Optimistisch
 - Es wird davon ausgegangen, dass keine Probleme auftreten \implies Falls doch, muss die Datenbank reagieren!
- Timestamp
 - Jede Transaktion enthält Startzeitpunkt \implies Konflikt tritt auf, wenn jüngere Transaktion die gleichen Daten beschreibt!

Sperrebenen

- Je feiner, desto aufwändiger, aber höhere Parallelität
- Je gröber, desto leichter, aber geringere Parallelität
- Ebenen
 - Datenbank
 - Tabelle
 - Physischer Block / Seite
 - Zeile

Arten von Sperren

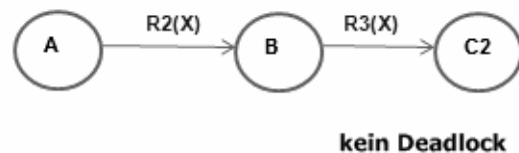
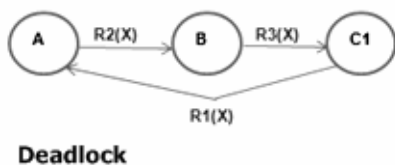
- X-Lock \Rightarrow Exklusiv, Read/Write erlaubt; es können keine weiteren Locks gesetzt werden!
- S-Lock \Rightarrow Shared, Read erlaubt; es können weitere S-Locks gesetzt werden!

5.3.3 Deadlocks

- Tritt dann auf, wenn 2 Transaktionen sich gegenseitig behindern (beide warten darauf, einen Lock auf gewisse Daten zu setzen!!)
- z.B. Wenn beide einen S-Lock auf einen Datensatz haben und dann jeweils einen X-Lock auf die anderen Daten setzen wollen

Behandlung

- Vermeidung
 - Eine Transaktion wird abgebrochen, wenn bei einer Sperranforderung die Gefahr auf einen Deadlock besteht; es werden sämtliche benötigte Objekte von Anfang an gesperrt!
- Erkennung
 - Es wird ein (gerichteter) Wartegraph geführt
 - * Knoten \Rightarrow Die einzelnen Transaktionen
 - * Kanten \Rightarrow Werden zwischen 2 Knoten gezeichnet, wenn einer auf den anderen warten muss
 - * Deadlock ist dann vorhanden, wenn im Graphen Zyklen enthalten sind!



5.3.4 Re-Read Methode

- Wird bei Änderungen an Daten im Mehruserbetrieb verwendet
- Ablauf
 1. Daten einlesen (ohne Sperre) \Rightarrow Record-Old
 2. Daten (oder Teile) von Record-Old kopieren \Rightarrow Record-Update

3. Daten in Record-Update (oder Teile davon) vom Benutzer ändern lassen
4. Datensatz erneut einlesen (mit X-Lock) \implies Record-Check
5. Record-Check mit Record-Old vergleichen:
 - a) Gleich \implies Update zulässig
 - b) Ungleich \implies Update unzulässig

5.3.5 U-Lock

- Kann bei Leseoperationen angegeben werden, um anschließend beabsichtigte Änderungs Operationen anzuzeigen

5.3.6 Isolation-Levels

- Read Uncommitted
 - kein Lock beim Lesen
 - Dirty Read, Non-Repeatable Read + Phantom sind möglich
- Read Committed
 - S-Lock auf Zeile beim Lesen, kein Two-Phase Locking
 - Non-Repeatable Read + Phantom sind möglich
- Repeatable Read
 - S-Lock auf Zeile beim Lesen bis Transaktionsende
 - Phantom ist möglich
- Serializable
 - S-Lock auf **Tabelle** beim Lesen bis Transaktionsende oder Predicate Locking
 - Nichts ist möglich

5.4 Backup & Recovery

- Backup \implies Kopie der Daten in einer Datenbank, um sie später wiederherzustellen
- Recovery \implies Das Wiederherstellen der Daten selbst (im Fehlerfall)

5.4.1 Backup

Arten von Backups

Die Einteilung kann nach Menge der Daten / Häufigkeit der Backups und nach dem Zustand des Systems zum Backupzeitpunkt eingeteilt werden.

Menge der Daten, Häufigkeit der Backups

- Full Backup \implies Es werden die gesamten Daten gesichert
 - **Vorteil:** Restore-Zeit ist gering, hohe Redundanz \implies sehr sicher
 - **Nachteil:** Viel Speicherplatz wird benötigt, Anfertigung des Backups braucht viel Zeit
- Partial Backup \implies Es werden nur die Daten gesichert, die sich geändert haben
 - **Vorteil:** Weniger Speicherplatz wird benötigt, Anfertigung des Backups braucht nicht so viel Zeit
 - **Nachteil:** Hohe Restore-Zeit, Daten sind nur bedingt redundant \implies geringere Sicherheit

- Unterarten
 - * Differential Backup \implies Es werden nur Daten gespeichert, die sich seit dem letzten **Full Backup** geändert haben
 - **Vorteil:** Restore-Zeit ist gering, sicherer da eine gewisse Redundanz gegeben ist
 - **Nachteil:** Viel Speicherplatz wird benötigt, Anfertigung des Backups braucht viel Zeit
 - * Incremental Backup \implies Es werden nur Daten gespeichert, die sich seit dem letzten **Partial Backup** geändert haben
 - **Vorteil:** Weniger Speicherplatz wird benötigt, Anfertigung des Backups braucht nicht so viel Zeit
 - **Nachteil:** Restore dauert länger, es gibt wenige Redundanzen \implies Nicht sehr sicher

Zustand des Systems zum Backupzeitpunkt

- Online (Hot) Backup
 - Wird während dem laufenden Betrieb ausgeführt
 - Achtung: Während der Erstellung des Backups können Änderungen geschehen \implies Es müssen vor Beginn des Backups die Änderungen mitprotokolliert werden
 - * Um konsistenten Zustand einzuspielen \implies Redo Logs müssen ausgeführt werden
- Offline (Cold) Backup
 - Wird ausgeführt, wenn die Datenbank offline ist

5.4.2 Recovery

Arten von Fehlersituationen

Transaktionsfehler (Lokaler Fehler)

- Transaktion wurde nicht ordentlich beendet; Daten sind nun in inkonsistentem Zustand
- Auslöser
 - Runtime-Fehler
 - Deadlock
 - Time-Out
 - Manuelles Rollback...
- Maßnahmen
 - Alle Änderungen bis hin zum Abbruch müssen zurückgenommen werden (Rollback / Transaction Recovery) \implies Backward Recovery

Systemfehler (Soft Crash)

- Mehrere Transaktionen konnten nicht ordnungsgemäß beendet werden
- Auslöser
 - Stromausfall
 - Fehler im Betriebssystem
- Maßnahmen
 - Alle Änderungen der Transaktionen, die beim Absturz in Progress waren, müssen zurückgenommen werden (Crash Recovery) \implies Backward Recovery

Mediumfehler (Hard Crash)

- Daten sind physikalisch zerstört / nicht mehr lesbar
- Auslöser
 - Irrtümliches Löschen von Daten
 - Fehler in der Dateiverwaltung des Betriebssystems
 - Fehler im Disk Controller
- Maßnahmen
 - Sicherungsstand wird eingespielt, Änderungen seit Sicherung müssen nachvollzogen werden (Media Recovery, Disaster Recovery, Crash Recovery) \implies Forward Recovery

Techniken für Recovery**Backward Recovery**

- Vor sämtlichen Änderungen innerhalb der Datenbank wird eine Kopie von den alten Werten (= Before Image) erstellt & in Undo-Log-Dateien gesichert \implies Nicht abgeschlossene Transaktionen können so rückgängig gemacht werden!
- Undo Log enthält unter Anderem:
 - Identifikation der Transaktion
 - Art der Operation (Insert...)
 - Before Image
- Logs werden meist zu Recovery-Zwecken fortlaufend geführt
- Transaction Recovery
 - Der Undo-Log wird bis zum Beginn der Transaktion gelesen
- Crash Recovery
 - Der Undo-Log wird bis zum Beginn gelesen, um alle Before-Images von nicht-beendeten Transaktion zu finden
 - Undo-Log wird in Checkpoints unterteilt (=zu diesem Zeitpunkt aktive Transaktionen werden gespeichert)
 - Undo-Log wird bis zum jüngsten Checkpoint gelesen \implies Alle Transaktion dieses Checkpoints, welche keine Endmarke haben, werden rückgängig gemacht!

Forward Recovery

- 2 Strategien
- Logging
 - Sämtliche Änderungen werden nach Ende der Transaktion als After Images in Redo-Log Dateien gespeichert
- Gespiegelte Platten
 - RAID

5.5 Data Control Language

- Wird verwendet, um gewissen Usern bestimmte Berechtigungen zu geben

6 Themenkorb - Datenbankarchitektur und Datenbankverwaltung

6.1 Datenbankarchitektur

6.1.1 Allgemeines

- Instanz \Rightarrow Der Datenbankprozess + sämtliche Hintergrundprozesse, befindet sich im Hauptspeicher
- Datenbank \Rightarrow Besteht aus den Datenbankdateien
- SGA (System Global Area) \Rightarrow Wird von allen Prozessen bzw. von allen Usern geteilt
- PGA (Program Global Area) \Rightarrow Pro Prozess

6.1.2 System Global Area

- Besteht aus
 - Database Buffer Cache
 - Shared Pool
 - Redo Log Buffer
 - Data Dictionary Cache

Database Buffer Cache

- Speichert zuletzt verwendete Daten, um einen schnellen Zugriff gewährleisten zu können
- Befindet sich im RAM
- Besteht aus zwei Listen:
 - Write-List: Enthält Daten, die modifiziert, aber noch nicht geschrieben wurden
 - LRU-List: Enthält Adressen von zuvor modifizierten / freien Daten, welche lange nicht benutzt wurden und damit wieder zum Beschreiben verfügbar sind

S

- Drei Arten von Blöcken:
 - Frei (schwarz)
 - Belegt (weiß)
 - Undo (für Rollback) (grau)

Shared Pool

- Enthält wichtige Daten von bereits ausgeführten SQL- und PL/SQL Anweisungen + das Data Dictionary
 - **Data Dictionary** \Rightarrow Metadaten zu Tabellen, Usern...
- Dient dazu, um bei gleichen Anfragen schnell Ergebnisse liefern zu können
- Funktioniert nach dem **LRU** Prinzip

Redo-Log Buffer

- Protokolliert sämtliche Änderungen im Database Buffer Cache \Rightarrow Werden für die Wiederholung von Statements (Forward Recovery) benötigt
- Geschriebene Daten beschränken sich auf die wesentlichsten Informationen: Delete \Rightarrow PK + Tabellenname...
- Hat eine fixe Größe (Ringbuffer) + ist dreigeteilt: Wenn das erste Drittel voll ist, beginnt der Logwriter (LGWR) das erste Drittel asynchron in die Online Redo-Log Dateien wegzuschreiben
 - Wenn der Buffer voll ist + das erste Drittel noch nicht vollständig weggeschrieben wurde, muss gewartet werden!

Data Dictionary Cache

- Prüft vor Ausführung von Statements, ob die Spaltennamen existieren

6.1.3 Prozesse

Database Writer (DBWR)

- Schreibt Daten aus dem Database Buffer Cache in die Daten-Dateien (standardmäßig alle 3 Sekunden, asynchron) (auch Tablespace genannt)
- Wählt die Daten aus, die am längsten nicht mehr verändert wurden (LRU)
- Schreibt, wenn:
 - Database Buffer Cache erreicht gewisse Größe
 - DBWR-Checkpoint wird erreicht
 - ein Time-Out auftritt
 - kein freier Buffer zur Verfügung steht
 - eine Liste zum Abarbeiten vom LGWR kommt

Log Writer (LGWR)

- Schreibt Änderungen vom Redo Log Buffer in Redo Log Dateien \Rightarrow Es kommen auch Undo-Informationen in die Redo-Logs!
- Schreibt sequentiell und wahlweise synchron / asynchron. Synchron ist kein Problem, da:
 - Teile wurden bereits asynchron weggeschrieben
 - Daten sind komprimiert
 - LGWR schreibt schneller als DBWR
 - Zusammenhängender Speicherplatz \Rightarrow schreibt sequentiell
- Commit \Rightarrow LGWR schreibt gesamten Redo Log Buffer synchron; Commit ist erst fertig, wenn Redo Log Buffer leer ist

Log File Switch

- Eine Redo-Log Datei ist voll \Rightarrow LGWR wechselt zu anderer Datei
- Zwei Modi:
 - **archive log** \Rightarrow **ARCH** Prozess kopiert die vollen Online Redo Logs in die Offline Redo Logs

- **noarchivelog** \implies Alte Dateien werden einfach überschrieben, falls der LGWR in seiner Rotation wieder bei einem vollen Log-File ankommt
- Änderung des Modus \implies DBWR muss Blöcke so wegschreiben, dass die zu überschreibenden Redo-Logs nur Informationen enthalten, die bereits auf die Daten-Dateien übertragen wurden

Check Point (CKPT)

- Am Ende eines Checkpoints wird Header der Data- + Control-Files geupdated \implies Datenbank kann so herausfinden, auf welchem Stand sich die Dateien befinden

Archiver (ARCH)

- Archiviert automatisch die Online Redo-Log- Dateien in die Offline-Redo Logs

System Monitor (SMON)

- Wird automatisch gestartet, wenn neue Datenbank-Instanz gestartet wird
- Schreibt Redo Logs von Änderungen, die noch nicht auf die Festplatte geschrieben wurden, damit bei Absturz die Instanz wiederhergestellt werden kann
- Führt automatischen Rollback aus, wenn Transaktionen nicht committed wurden
- Außerdem \implies Alle 5 Minuten werden Wartungen durchgeführt, die Speicherplatz / temporäre Segmente freigeben

Process Monitor (PMON)

- Zuständig, wenn ein User-Prozess versagt / abgebrochen wird \implies reinigt Cache + gibt vom Prozess benutzte Ressourcen frei

Recoverer (RECO)

- Zuständig, um Fehler, welche in ausgelagerten Transaktionen auftreten, zu beheben

Lock Prozess (LCKn)

- Zuständig für das Sperren von Ressourcen zwischen verschiedener Instanzen in einem Server

Dedicated Server Prozess

- Fertigt genau einen User ab
- Sinnvoll, wenn dieser eine User besonder viel macht
- Bei SELECT werden Daten direkt aus Database Buffer Cache / aus den Daten-Dateien geholt

Shared Server-Prozess

- Kann mehrere User-Prozesse abfertigen

Dispatcher

- Sorgt für Zuteilung zwischen User und Server

Listener

- Verbindet Client zu Datenbank-Server

6.1.4 Dateien

Daten-Dateien

- Enthalten Benutzer- und Systemdaten der Datenbank

Control-Dateien

- Enthalten eine Beschreibung der Datenbankstruktur + ermöglichen Überprüfung und Sicherstellung der Integrität

Redolog-Dateien

- Online / Offline
- Enthalten Befehle, welcher der User eingegeben hat, in einer komprimierten Form

6.1.5 Fehlerbehandlung

Lokaler Fehler

- Rollback wird mithilfe der Undo-Blöcke im Database Buffer Cache durchgeführt

Rollback

- Stromausfall \implies SMON stellt aus Redo-Log Dateien wieder konsistenten Zustand her

6.2 Datenbankverwaltung

6.2.1 Allgemeines - Import

- ...wenn Daten aus
 - .csv
 - .txt
 - Excel-Files
 - einem Datenbankbackup
- importiert werden
- Herausforderung: Daten sind oft unnormalisiert, enthalte NULL-Werte...

6.2.2 SQL Loader

- Wird dazu verwendet, um Daten in eine Oracle-DB zu importieren
- Verwendet Control-Dateien, die den Aufbau der Ausgangsdateien beschreiben
 - Können auch vom SQL-Developer erstellt werden!
- Beim Import werden sowohl erfolgreiche als auch gescheiterte Zeilen geloggt, welche dann später analysiert werden können

Control-Dateien

- Beschreibt den Aufbau der zu importierenden Daten
- Es können Felder + Datentyp definiert werden, das Characterset festgelegt werden...

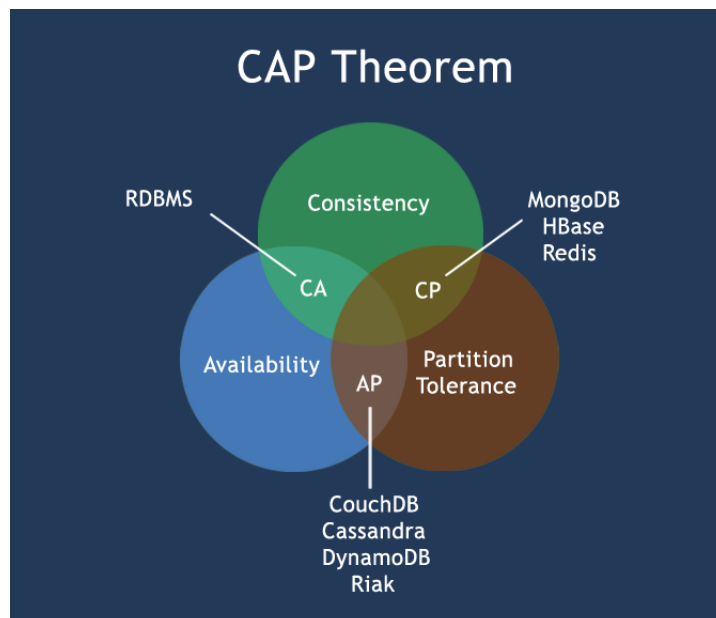
6.2.3 Weitere Tools

- Oracle Import-Utility
 - Es können Daten aus einer Exportdatei (dump) importiert werden \implies Müssen zuvor mit der Export-Utility exportiert worden sein
- Data Pump
 - Neuer, schneller und flexibler als die Import-Utility; es kann eine PL/SQL API verwendet werden!

8 Themenkorb - Aktuelle Datenmodelle

8.1 Allgemeines

- NO-SQL \Rightarrow Not-Only-SQL
- Nicht relational, keine fixe Tabellenstruktur, erlauben horizontale Skalierung (= mehr Geräte statt stärkerer Hardware)
- Relationale Datenbanken kämpfen teils mit häufigen Änderungen an bestehenden Daten bzw. gigantischen Datenmengen
- Hauptattribute jeder Datenbank \Rightarrow Konsistenz, Verfügbarkeit, Ausfalltoleranz \Rightarrow CAP Theorem: Nicht alles kann zu 100% erfüllt sein!



8.1.1 BASE

- NO-SQL Datenbanken arbeiten oft nach dem BASE-Prinzip
- Steht für Basically Available, Soft State, Eventual Consistency
- Gibt absolute Konsistenz der Daten auf, um dafür die Verfügbarkeit des Systems zu verbessern \Rightarrow Kann zwischendurch in nicht konsistentem Zustand sein!

8.1.2 Gliederung von NO-SQL Datenbanken

- Document-Store
 - Kleinste Informationseinheit ist ein Document; wird mit einer eindeutigen ID identifiziert (meist automatisch generiert) \Rightarrow z.B. MongoDB
- Key-Value-Store
- Graph

- Daten werden in Knoten gespeichert, welche durch Kanten verbunden werden (können ebenfalls Informationen enthalten; z.B. Kosten der Verbindung)
- Speziell auf gewisse Queries ausgerichtet (Kürzester Pfad...)

8.2 Redis

8.2.1 Allgemeines

- Schemafrei
- Daten werden grundsätzlich In-Memory (Schnell!) gespeichert & können wahlweise auch auf die Festplatte übertragen werden
- Zu einem gewissen Key können ein oder mehrere Werte abgespeichert werden
- Eventual Consistency \implies Daten werden nicht sofort auf allen Servern/Partitionen geschrieben, sondern erst nach einer gewissen Zeit
 - Dadurch kann es bei gleichen Abfragen teilweise zu unterschiedlichen Ergebnissen kommen!
- Transaktionen \implies Optimistic Locking \implies Alle User haben Lesezugriff, im Falle einer Änderung werden alle benachrichtigt

8.2.2 Sharding

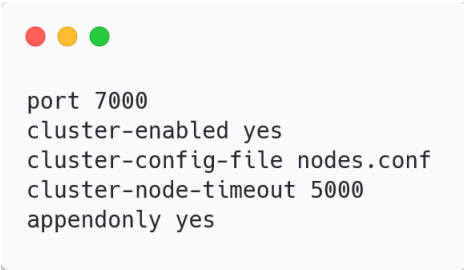
- Auch bekannt als Partitioning \implies Die Daten werden anhand ihrer Keys aufgeteilt und dann auf verschiedenen Maschinen gespeichert

8.2.3 Replication

- Die Daten werden über mehrere Maschinen hinweg gespiegelt \implies Erhöht Lesegeschwindigkeit

8.2.4 Cluster

- Daten können auf mehrere Nodes aufgeteilt werden; Es kann weitergearbeitet werden, wenn Nodes ausfallen
- Für jede Node im Cluster muss ein Config-File erstellt werden, welches wie folgt aussieht (Ports müssen entsprechend angepasst werden):

A terminal window with a light gray background and three colored window control buttons (red, yellow, green) in the top left corner. It displays the following configuration for a Redis cluster node:

```
port 7000
cluster-enabled yes
cluster-config-file nodes.conf
cluster-node-timeout 5000
appendonly yes
```

Weitere Konfigurationsparameter

- **cluster-slave-validity-factor:** Gibt, multipliziert mit cluster-node-timeout, die maximale Zeit an, in der der Slave versucht für den Master zu übernehmen
 - Wenn 0 \implies Slave probiert immer, Master zu ersetzen
- **cluster-migration-barrier:** Minimum an Slaves, die einem Master erhalten bleiben müssen und somit nicht zu anderen Masters migriert werden können

- **cluster-require-full-coverage:** Gibt an, ob die gesamte Slot-Range durch einen Master gecovered sein muss

Data Sharding

- Ein Redis-Cluster enthält 16484 Hash-Slots (Keys werden nach Formel dem Slot zugewiesen)
- Jede Node ist für einen gewissen Teil dieser Hash-Slots zuständig
- Multiple Key Operations \Rightarrow Alle Keys müssen Teil des selben Hash-Slots sein!

Master & Slave

- Um Ausfallssicherheit zu garantieren \Rightarrow Ein Master kann N Slaves haben, welche die gleiche Hash-Slot-Range abdecken!
- Node Timeout gibt an, wie lange gewartet wird, bis eine Node als inaktive gilt & ein Slave übernimmt

8.2.5 Befehle

Standard

SET [key] [value]	Set the string value of a key
GET [key]	Get the value of a key
MGET [key] ...	Get the values of all the given keys
MSET [key] [value] [key value...]	Set multiple keys to multiple values
GETSET [key] [value]	Set the string value of a key and return its old value
DEL [key]	Delete a key
EXISTS [key]	Determine if a key exists
GETRANGE [key] [start] [end]	Get a substring of the string stored at a key
RENAME [key] <u>[newkey]</u>	Rename a key
DECR [key]	Decrement the integer value of a key by one
DECRBY [key] [decrement]	Decrement the integer value of a key by the given number
INCR [key]	Increment the integer value of a key by one
INCRBY [key] [increment]	Increment the integer value of the key by the given amount
EXPIRE [key] [seconds]	Set a key's time to live in seconds
PERSIST [key]	Remove the expiration from a key
ECHO [message]	Echo the given string
TIME	Return the current server time
KEYS [pattern]	Find all keys matching the given pattern
STRLEN [key]	Get the length of the value stored in a key
HDEL [key] [field] [field...]	Delete one or more hash fields
HEXISTS [key] [field]	Determine if a hash field exists
HGET [key] [field]	Get value of a hash field
HGETALL [key]	Get all fields and values in a hash
HINCRBY [key] [field] [increment]	Increment the integer value of a hash field by the given number
HKEYS [key]	Get all the fields in a hash
HLEN [key]	Get the number of fields in a hash
HMGET [key] [field] [field...]	Get the values of all the given hash fields
HMSET [key] [field] [value] ...	Set multiple hash fields to multiple values
HSET [key] [field] [value]	Set the string value of a hash field
HSTRLEN [key] [field]	Get the length of the value of a hash field
HVALS [key]	Get all the values in a hash

Geo

```

GEOADD --> Um einen neuen Eintrag (ggf. zu einem bestehenden Key) hinzuzufügen (longitude, latitude & name können wiederholt werden um mehrere zu einem Key hinzuzufügen!):

GEOADD {key} {longitude} {latitude} {name}

GEODIST --> Um die Entfernung zwischen 2 Einträgen innerhalb eines keys zu finden:

GEODIST {key} {member1} {member2} [m|km|ft|mi]

GEOPOS --> Um die Werte für einen Eintrag innerhalb eines Keys auszulesen:

GEOPOS {key} {member} [member ...]

GEOSEARCH --> Um Objekte innerhalb eines Keys für z.B. einen gewissen Radius etc. zu finden:

GEOSEARCH {key} [FROMMEMBER member] [FROMLONLAT longitude latitude] [BYRADIUS radius m|km|ft|mi] [BYBOX width height m|km|ft|mi] [ASC|DESC] [COUNT count [ANY]] [WITHCOORD] [WITHDIST] [WITHHASH]

GEOSEARCHSTORE --> Wie GEOSEARCH, nur dass vor dem Key noch einen Destination angegeben werden kann, in welche das Ergebnis der Query gespeichert wird.

GEOSEARCHSTORE {destination} {source} [FROMMEMBER member] [FROMLONLAT longitude latitude] [BYRADIUS radius m|km|ft|mi] [BYBOX width height m|km|ft|mi] [ASC|DESC] [COUNT count [ANY]] [STOREDIST]

GEOHASH --> Um die Geohashes eines/mehrerer Einträge eines Keys zu finden:

GEOHASH key member [member ...]

```

8.3 MongoDB

8.3.1 Struktur / Vergleich mit rel. Datenbank

- Database \Rightarrow Database
- Table \Rightarrow Collection
- Row \Rightarrow Document
- Column \Rightarrow Field

8.3.2 Befehle

Daten einspielen

```

db.schueler.insertOne({
  klasse: "5CHIF",
  vorname: "Maximilian",
  nachname: "Musterschüler",
  vertWunsch: ["MANE", "NOSQL"]
})
db.schueler.insertMany([
  {
    klasse: "3BHIF",
    vorname: "Günther",
    nachname: "Schwammkopf",
    vertWunsch: ["SNVS", "MANE"],
  }, {
    klasse: "1CHIF",
    vorname: "Andreas",
    nachname: "Kindermann",
    geburtsdatum: new Date("2000-04-19"),
    vertWunsch: ["SAP", "SNVS", "SIMF"],
    eigenberechtigt: false
  }
])

```

Daten updaten

```

db.schueler.updateMany( { klasse:"5CHIF" },
  { $set: { klasse:"90CHIF" } } );

db.schueler.updateOne( { vorname:"Andreas", nachname:"Kindermann" },
  { $unset: { fehlstunden: 0 } } );

```

Daten löschen

```

db.schueler.deleteOne({ nachname:"Johnlock" });
db.schueler.deleteMany({ klasse:"3CHIF" });

```

Daten anzeigen

```

db.schueler.find({"vertWunsch.0": "NOSQL"});
db.schueler.find({vertWunsch: ["BI", "SAP"]});
// Nur Vorname + Nachname werden ausgegeben!
db.schueler.find({}, {vorname: 1, nachname: 1});

```

8.4 Neo4J

- Daten werden in Form von Knoten gespeichert, die von einem gewissen Typen sein können (z.B. Person)
- Knoten werden über Kanten verbunden, welche ebenfalls einem Typen + gewisse Attribute haben können

8.4.1 Befehle

Daten einspielen

```

CREATE ( p1 : Person { name : "Max" } ),
// Knoten
( p2 : Person { name : "Maria" } ),

// Unidirektionale Verbindung;
// vor DP könnte Name stehen!
( p1 ) - [ : wohnt_bei ] -> ( p2 ),
( p1 ) <- [ : pflegt ] - ( p2 );

// Erstellen von Beziehung zwischen
// bestehenden Knoten
MATCH ( p1 : Person { name : "Max" } ),
( p2 : Person { name : "Maria" } )
CREATE ( p1 ) - [ : liebt ] -> ( p2 )

```

Daten updaten

```
MATCH (p:Person {name: 'Jennifer'})
SET p.birthdate = date('1980-01-01');
```

Daten löschen

```
MATCH ( p : Person { name : "Max" } )
DETACH DELETE p;
```

Daten anzeigen

```
match (c:Stadt) WHERE c.population > 200000 RETURN c;
match (c:Stadt) WHERE c.population < 200000 return c;
match (s:Stadt {name : "Graz"}) --> (d:Stadt) WHERE d <> s return d;
match (s:Stadt { name: "Graz"}) -[:direkt_nach] -> (d:Stadt) -[:liegt_an] -> (f:Fluss { name: "Donau"}) return d;
match (s:Stadt) -[:direkt_nach]-> (s2:Stadt) -[:direkt_nach] -> (s) return s;
match (s:Stadt {name: "Graz"}) -[:direkt_nach*2]->(s2:Stadt) WHERE s2 <> s return s2;
match (s:Stadt {name: "Graz"}) -[:direkt_nach*1..2]->(s2:Stadt) WHERE s2 <> s return s2;
match (s:Stadt {name: "Bregenz"}), (s2:Stadt { name: "Wien"}), p = SHORTESTPATH((s)-[:direkt_nach*]->(s2)) UNWIND
nodes(p) AS n return count(*);
MATCH (s1:Stadt{name:"Wien"}), (s2:Stadt{name:"Bregenz"}), p = shortestPath((s1)-[:direkt_nach*]->(s2)) RETURN p;
```