TMS II

Begriffsdefinitionen

Konfigurationsmanagement

KM stellt sicher, dass Produkte eindeutig identifizierbar sind, Zusammenhänge und Unterschiede von verschiedenen Versionen einer Konfiguration erkennbar bleiben und Produktänderungen nur kontrolliert durchgeführt werden können.
Arten der Durchführung: - Automatisch (z.B. Git)
- Manuell
- Semi-automatisch (z.B. unterstützende Plugins)

Build-Management
Unter Build-Management wird diejenige Funktion verstanden, die alle Bauteile einer Konfiguration erzeugt, die nicht direkt vom Benutzer erstellt oder durch das System vorgegeben wurden. Zentrale Fragen: Wie ist die Software zusammengestellt? Welche Abhängigkeiten müssen berücksichtigt werden? Beispiele für klassische Abhängigkeiten: o Entwicklungsumgebung: Editor, Generatoren, Compiler, Linker, . . .
o Basissoftware: LAN, ...
o Betriebssystem
o Hardware

Change-Management
Im Änderungswesen muss definiert werden, wie Änderungswünsche (Change Requests) erfasst werden, wie und durch wen diese bewertet werden und wie die Durchführung von Änderungen zu erfolgen hat.
- Versions-Management (Wer hat wann was geändert?) Versionsmanagement befasst sich (in erster Linie mit der Verwaltung der zeitlich aufeinander folgenden Revisionen eines Dokuments.) **Release-Management**
Ein Release ist eine an Kunden ausgelieferte Konfiguration eines (Software-)Systems, bestehend aus ausführbaren Programmen, Bibliotheken, Dokumentation, Quelltexten, Installationsskripten und so weiter. Das Release-Managementdokumentiert ausgelieferte Konfigurationen und stellt deren Rekonstruierbarkeit sicher. Geplant

Produktzustände

In Bearbeitung
Das Produkt wird bearbeitet. Es befindet sich entweder im privaten Entwicklungsbereich des Entwicklers oder unter Kontrolle des Entwicklers innerhalb der Produktbibliothek.

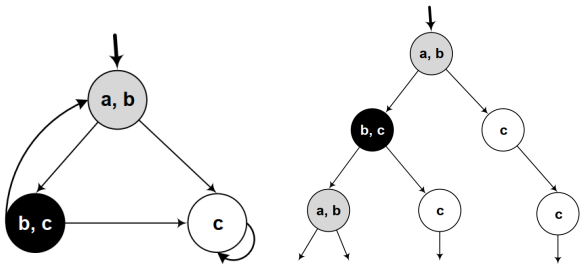
Vorgelegt
Das Produkt ist aus der Sicht des Erstellers fertig und wird der Konfigurationsverwaltung übergeben. Ab jetzt kann es einer Prüfung durch die Qualitätssicherung unterzogen werden. Wird das Produkt hierbei abgelehnt, so geht es wieder in den Zustand in Bearb. zurück, andernfalls rückt es in den Zustand akzeptiert vor. Ab dem Zustand vorgelegt an kann der Ersteller nur unter Fortschreibung der Versionsangabe Modifikationen durchführen.

Akzeptiert
Das Produkt wurde durch die QS überprüft und freigegeben. Es darf nur noch innerhalb einer neuen Version geändert werden.

CTL

Zustandsautomat als Zustandsraum

Zustandsraum endet mit Übergängen



Pfadoperatoren:

- $A\phi$ - auf allen Pfaden folgt ϕ (englisch: All)
- E ϕ - auf mindestens einem Pfad folgt ϕ (englisch: Exists)

Pfad-spezifische Operatoren:

- $X\phi$ - unmittelbar folgt ϕ (englisch: neXt state)
- $F\phi$-irgendwann folgt ϕ (englisch: some Future state oder Finally)
- $G\phi$ - auf dem folgenden Pfad folgt in jedem Zustand ϕ (englisch: Globally)
- $\phi U\psi - \phi$ folgt bis zum Erreichen des Zustands ψ (englisch: Until)
- $\phi W\psi - \phi$ folgt immer oder bis zum Erreichen des Zustands ψ (englisch: Weak Until)
- $EX\phi$ - in (mind.) einem nächsten Zustand gilt ϕ
- $EF\phi$ - in (mind.) einem der folgenden Zustände gilt ϕ (AG EF Terminal) stellt Terminal sicher
- EG ϕ - es gibt (mind.) einen Pfad, so dass ϕ entlang des ganzen Pfades gilt
- $E[\phi U\psi]$ - es gibt einen Pfad, für den gilt: bis zum ersten Auftreten von ψ gilt ϕ
- $AX\phi$ - in jedem unmittelbar nächsten Zustand gilt ϕ
- AF ϕ - man erreicht immer einen Zustand, in dem ϕ gilt
- $AG\phi$ - auf allen Pfaden gilt in jedem Zustand ϕ
- $A[\phi U\psi]$ - es gilt immer ϕ bis zum ersten Auftreten von ψ

Semantik

- $T(s_0) \models \neg\phi \quad \Leftrightarrow \quad T(s_0) \not\models \phi$
- $T(s_0) \models \phi \vee \psi \quad \Leftrightarrow \quad T(s_0) \models \phi \,\text{oder}\, T(s_0) \models \psi$
- $T(s_0) \models EX\phi \quad \Leftrightarrow \quad T(s_1) \models \phi$
- $T(s_0) \models EG\phi \quad \Leftrightarrow \quad \forall i : T(s_i) \models \phi$
- $T(s_0) \models \phi EU\psi \quad \Leftrightarrow \quad \exists k : T(s_k) \models \psi \wedge \forall i < k : T(s_i) \models \phi$

Transformationen

- $\neg A\phi \equiv E\neg\phi$
- $\neg AF\phi \equiv EG\neg\phi$
- $\neg EF\phi \equiv AG\neg\phi$
- $\neg AX\phi \equiv EX\neg\phi$
- $AG\phi \equiv \phi \wedge AXAG\phi$
- $AG\phi \equiv \neg EF(\neg\phi)$
- $EG\phi \equiv \phi \wedge EXEG\phi$
- $AF\phi \equiv \phi \vee AXAF\phi$
- $EF\phi \equiv \phi \vee EXEF\phi$
- $A[\phi U\psi] \equiv \psi \vee (\phi \wedge AXA[\phi U\psi])$
- $E[\phi U\psi] \equiv \psi \vee (\phi \wedge EXE[\phi U\psi])$

Wahrheitstabelle

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$
T	T	T	T	T
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

Aufwandsschätzung

Allgemein

Proportionalitätsfaktor: $k = \frac{1}{T}$

Zeitkonstante:

Entwicklungsleistung bekannt

Gesamtleistung $N_G(t) = f(t) + k \cdot \int f(t) \cdot dt + C$
Wartungsleistung $N_W(t) = k \cdot \int f(t) \cdot dt + C$
Entwicklungsleistung $N_E(t) = f(t)$
Gesamtarbeit $A_G(t) = A_E(t) + A_W(t)$
Entwicklungsarbeit $A_E(t) = \int f(t) \cdot dt + C$
Wartungsarbeit $A_W(t) = k \cdot \int \left[\int f(t) \cdot dt + C_1\right] \cdot dt + C_2$

Gesamtleistung bekannt

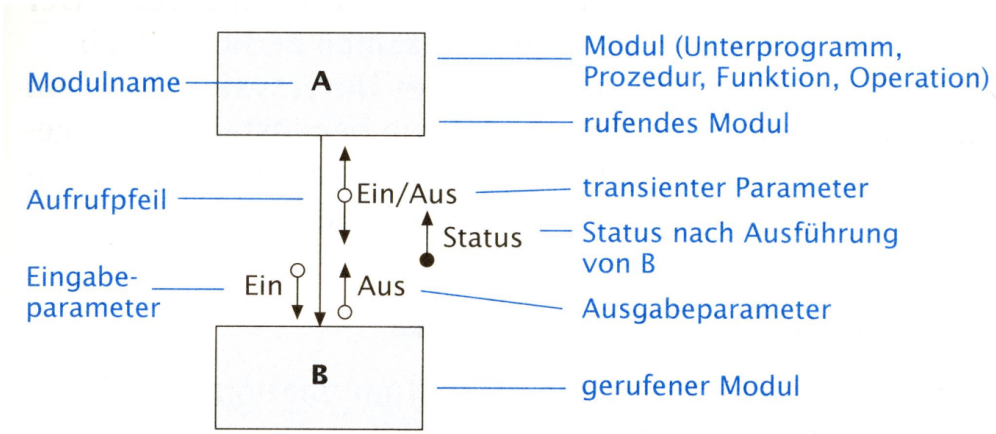
Gesamtleistung $g(t) = N_g = [Mann]$ Anzahl Leute
Wartungsleistung $N_W(t) = k \cdot e^{-kt} \left[\int e^{kt} \cdot g(t) dt + C\right] \overset{C=0}{=} k \cdot e^{-kt} \cdot N_G \cdot \int_0^t e^{kt} dt = N_G \left(1 - e^{-kt}\right)$
Entwicklungsleistung $N_E(t) = N_G - N_W(t) = N_G \cdot e^{-kt}$
Gesamtarbeit $A_G = N_G \cdot t$
Entwicklungsarbeit $A_E(t) = \frac{1}{k} \cdot N_G \left(1 - e^{-kt}\right)$
Wartungsarbeit $A_W(t) = A_G(t) - A_E(t) = N_G \cdot t - A_E(t) = N_G \left[t - \frac{1}{k} \cdot \left(1 - e^{-kt}\right)\right]$

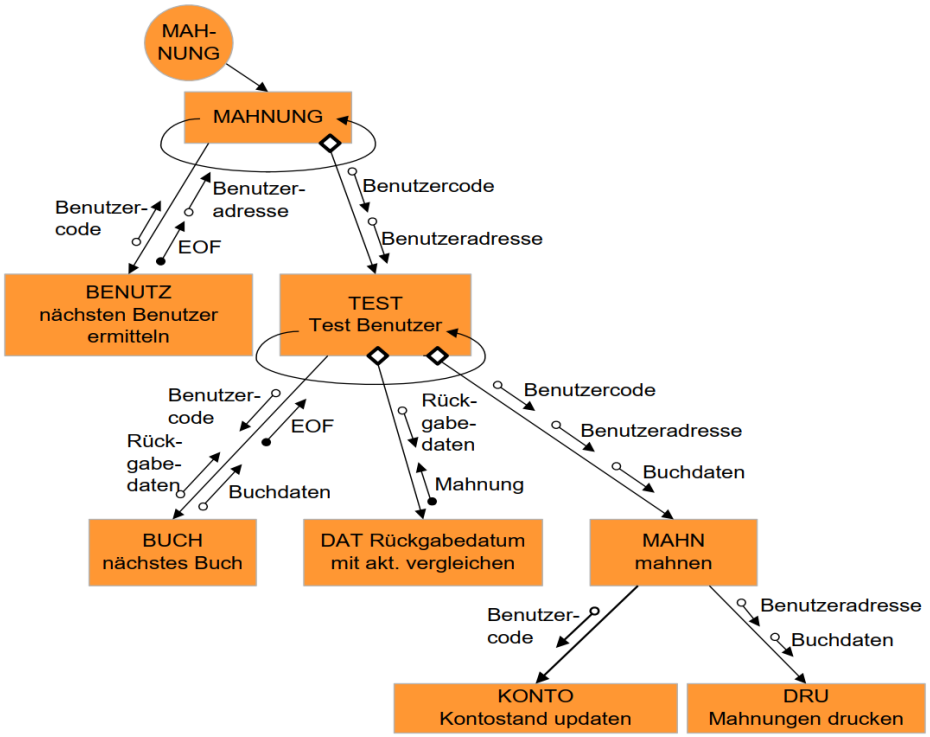
Entwicklungsarbeit bekannt

Gesamtleistung $N_G(t) = N_E(t) + N_W(t)$
Wartungsleistung $N_W(t) = k \cdot h(t) + C$
Entwicklungsleistung $N_E(t) = \frac{dh(t)}{dt}$
Gesamtarbeit $A_G(t) = h(t) + A_W(t)$
Entwicklungsarbeit $A_E(t) = h(t)$
Wartungsarbeit $A_W(t) = \int [k \cdot h(t) + C] \cdot dt + C_1$ mit $k = \left[\frac{Mann}{Mannjahr}\right]$ Wartungsintensität

Reverse Engineering

Structure Chart





Nesting tree

Indent tree
TODO Maybe add picture

Datenbanken

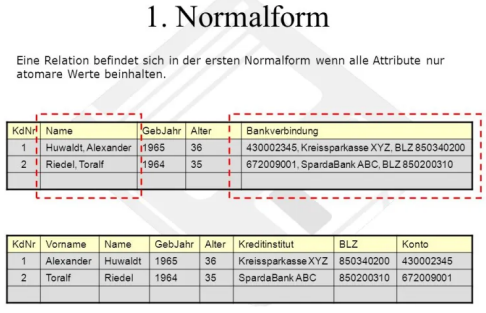
Allgemein

1. Vermeidung von Redundanz
2. Vermeidung von Insert-Anomalie
3. Vermeidung von Delete-Anomalie
4. Vermeidung von Update-Anomalie

Normalformen

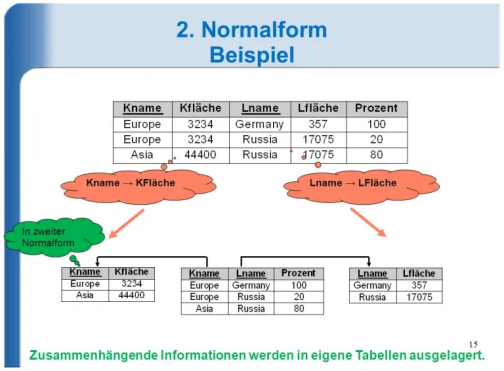
1. Normalform

Jedem Datenfeld eines Datensatzes darf höchstens ein Wert zugewiesen sein. D.h. es dürfen keine Mehrfacheinträge in einem Datenfeld vorliegen und Attribute müssen atomar sein.



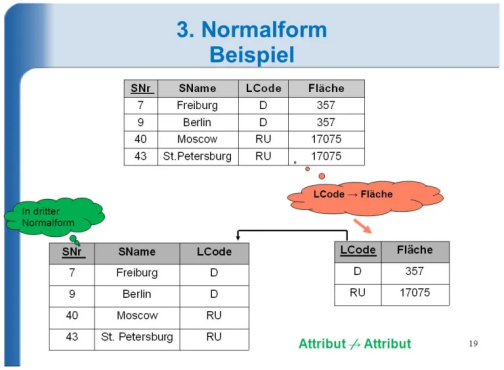
2. Normalform

Erfüllt 1. Normalform
Eine Tabelle enthält nur Daten eines Themen- bzw. Informationsbereiches.
Aufteilung in mehrere Tabellen nach Themen/ Informationsgebieten.
Jedes Nicht-Schlüsselfeld muss durch ein Schlüsselfeld identifizierbar sein und vom gesamten Schlüssel abhängen.
Überprüfen und ggf. neue Schlüsselfelder hinzufügen oder zusammengesetzte Schlüssel definieren.



3. Normalform

Aus keinem nicht Schlüsselattribut folgt ein anderes nicht Schlüsselattribut
Ist dann erfüllt, wenn die Tabelle die 2. Normalform erfüllt
Es dürfen keine transitiven (indirekten) Abhängigkeiten vorliegen.
Entfernung aller transitiven Abhängigkeiten durch Aufspalten der Tabelle in mehrere Tabellen, in denen alle Nicht-Schlüsselfelder direkt vom gesamten Schlüsselfeld abhängen



ERD

Kardinalitäten

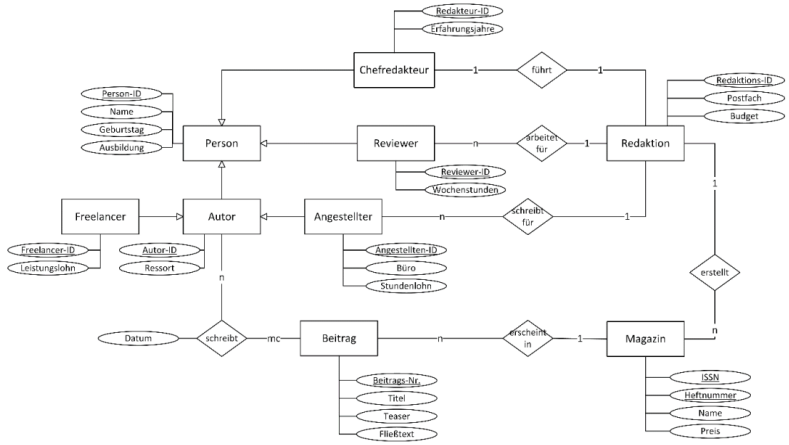
	1	C	M	CM
1	(1 : 1)	(1:C)	(1:M)	(1:MC)
C	(C : 1)	(C:C)	(C:M)	(C:MC)
N	(N : 1)	(N:C)	(N:M)	(N:MC)
NC	(NC : 1)	(NC:C)	(NC:M)	NC:MC

- **1:** Genau 1
- **C:** 1 oder kein
- **M und N** mehrere
- **MC** kein, ein, oder mehrere

Implementierung

Vererbung

Basis Entität bekommt foreign key ART-ID
Zusatztabelle: Basisentität ist ein mit primary key Art-ID und Attribut Art
Abgeleitete Entitäten bekommen Primary key der basis Entität als foreign key



BSP

Person	Person-ist-ein	Reviewer
Person-ID, Name, Birth, Person-Art-ID	Person-Art-ID, Art	Reviewer-ID, Person-ID, Stunden, Redaktions-ID

Chefredakteur
Red-ID, Person-ID, Jahre, Redaktions-ID

One-to-Many

One-Seite

```
CREATE TABLE Autoren ( AutorID INT PRIMARY KEY,
Name VARCHAR(255) NOT NULL,
Geburtsdatum DATE );
```

Many-Seite

```
CREATE TABLE Bücher (
BuchID INT PRIMARY KEY,
Titel VARCHAR(255) NOT NULL,
Veröffentlichungsdatum DATE,
AutorID INT,
FOREIGN KEY (AutorID) REFERENCES Autoren (AutorID) ON DELETE SET NULL );
```

Many-to-Many

```
 CREATE TABLE BuchAutoren (
BuchID INT,
AutorID INT,
PRIMARY KEY (BuchID, AutorID),
FOREIGN KEY (BuchID) REFERENCES Bücher (BuchID) ON DELETE CASCADE,
FOREIGN KEY (AutorID) REFERENCES Autoren (AutorID) ON DELETE CASCADE );
```

Anzahl auftreten aus Relation

BSP: Autorenanzahl pro Buch

```
 SELECT b.BuchID, b.Titel, COUNT(ba.AutorID) AS AnzahlAutoren
FROM Bücher b
LEFT JOIN BuchAutoren ba ON b.BuchID = ba.BuchID
GROUP BY b.BuchID, b.Titel
ORDER BY b.BuchID;
```

SQL

Befehle

DISTINCT: Gibt nur eindeutige Werte ohne Duplikate

Select from n based on limit in n

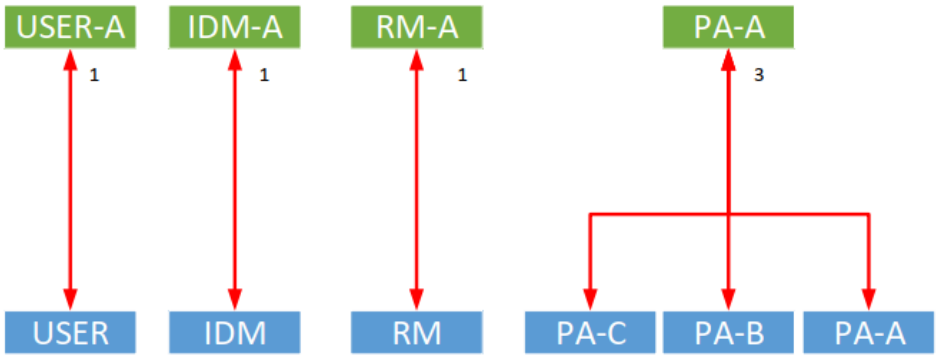
Alle Tests aus den 5 Laboren mit dem höchsten Budget

```
 SELECT Test.ID, Test.Datum, Labor.Name
FROM Test
INNER JOIN Labor ON Test.LaborID = Labor.LaborID
WHERE Test.Risiko > 2
AND Labor.LaborID IN (
SELECT LaborID
FROM Labor
ORDER BY Budget DESC
LIMIT 5
);
```

BSB: Create table from ERD

```
CREATE TABLE Spielmaterial ( Material-Nr INT NOT NULL,
Art VARCHAR(50) NOT NULL,
Farbe VARCHAR(15) NOT NULL,
Beschriftung VARCHAR(50) NOT NULL,
Gewicht DOUBLE NOT NULL,
Spiel-ID INT NOT NULL,
PRIMARY KEY (Material-Nr),
FOREIGN KEY (Spiel-ID) REFERENCES Spiel(Spiel-ID));
```

GAIA



Role Model Template

Role Schema:	
Description:	
Protocols and Activities:	
Permissions:	- reads: - changes: - generates:
Responsibilities:	- Liveness: - Safety:

- Protokolle und Aktivitäten (Protocols and Activities)
  - Protokolle legen fest, wie verschiedene Rollen miteinander interagieren;
  - Aktivitäten sind Aufgaben, die von einer Rolle ohne Interaktion erledigt werden.
- Rechte (Permissions)
  - Hier werden die Rechte zum Lesen, Ändern und Erzeugen von Datenelementen spezifiziert.
- Verantwortlichkeiten (Responsibilities)
  - Verantwortlichkeiten werden in zwei Kategorien unterteilt: Lebendigkeit (Liveness) und Sicherheit (Safety).
    * Liveness beschreibt die Reihenfolge der Aufgaben, die von der Rolle realisiert werden müssen. Sie setzen sich aus den im ersten Bereich beschriebenen Protokollen und Aktivitäten zusammen.
    * Safety sind Bedingungen (Invarianten), die eingehalten werden müssen, um unerwünschte oder sogar gefährliche Situationen zu vermeiden.

BSB Role Description

Role Schema: S1

Description: S-Bahnverbindung zwischen Landbahnhof, Stadtbahnhof und zurück. Stellt eine Transportkapazität von 360 Personen pro Stunde zur Verfügung (180 Personen pro Stunde in jede Richtung der Verbindung).

Protocols and Activities

DriveTrack, NegotiateGoals, NegotiateRole

Permissions

Reads: TransportCapacity; Changes:TimetableS1

Responsibilities - Liveness:

S1 = ((DriveToStadt · DriveToLand) | NegotiateGoals | NegotiateRole)^{w}

Template for Interaction Model

[Protocol's name]		[Inputs]
[Initiator]	[Responder]	
[Description]		[Outputs]

- Description: Beschreibung
- Initiator: Rolle
- Responder: Rolle
- Inputs: Informationen des Auslösers
- Outputs: Informationen des Beantworters

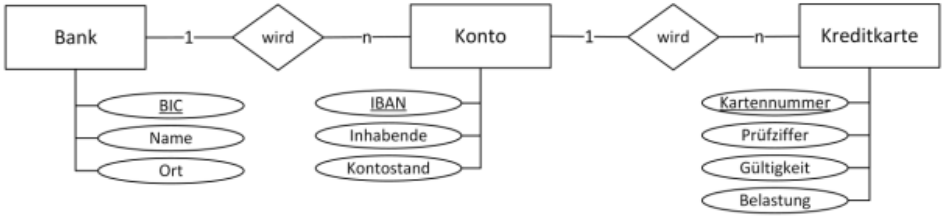
Operatoren Ablaufbeschreibung

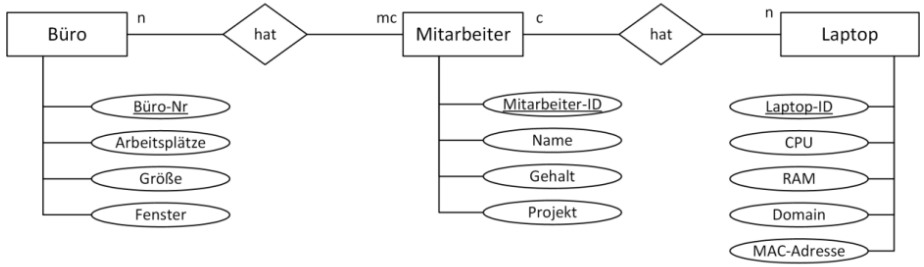
Operator	Interpretation
$x.y$	x followed by y
$x\|y$	x or y occurs
x^*	x occurs 0 or more times
$X+$	x occurs 1 or more times
x^ω	x occurs innately often
$[x]$	x is optional
$x\|\|y$	x and y are interleaved(verschachtelt)

Service Model

Service	Inputs	Outputs	Precondition	Post-condition

SQL Beispiele





Erstellen Sie eine SQL-Anweisung, welche die Tabelle Konto mit ihren zugehörigen Attributen der Datenbank hinzufügt.:

CREATE TABLE Konto (
IBAN Varchar(22) NOT NULL,
Inhabende Varchar(50) NOT NULL,
Kontostand INT,
BIC Varchar(11) NOT NULL,
PRIMARY KEY (IBAN),
FOREIGN KEY (BIC) REFERENCES Bank(BIC));

Erstellen Sie eine SQL-Anweisung, welche die Kontodaten (IBAN, BIC, Bankname und Inhabende) aller Konten von Stuttgarter Banken aufsteigend sortiert nach ”Inhabende” ausgibt.:

SELECT Bank.Name, Bank.BIC, Konto.IBAN, Konto.Inhabende
FROM Konto
JOIN Bank USING (BIC)
WHERE Bank.Ort = “Stuttgart”
ORDER BY Konto.Inhabende ASC;

Erstellen Sie eine SQL-Anweisung, welche die IBAN und das gesparte Geld gruppiert nach Konten mit weniger als 10.000€ und aufsteigend sortiert nach dem gesparten Geld auflistet. Dabei berechnet sich das gesparte Geld als Differenz des Kontostandes und der Summe aller Kreditkartenbelastungen. Es sollen nur die ersten 100 Treffer ausgegeben werden.

SELECT Konto.IBAN,
Konto.Kontostand-SUM(Kreditkarte.Belastung) AS Geld
FROM Konto LEFT JOIN Kreditkarte USING(IBAN)
GROUP BY Konto.IBAN HAVING Geld ¡ 10000
ORDER BY Geld
LIMIT 100;

Erstellen Sie eine SQL-Anweisung, welche den 100 ärmsten Konten der Konten in Stuttgart die einen Kontostand unter 10.000 € aufweisen 5.000€ gutschreibt.

UPDATE Konto
LEFT JOIN Bank USING(BIC)
SET Konto.Kontostand = Konto.Kontostand + 5000
WHERE Konto.Kontostand ¡ 10000 AND Bank.Ort = “Stuttgart“
ORDER BY Konto.Kontostand
LIMIT 100;

Erstellen Sie eine SQL-Anweisung, welche die Tabelle Laptop mit ihren zugehörigen Attributen der Datenbank hinzufügt.

CREATE TABLE Laptop (
Laptop-ID INT NOT NULL,
CPU DOUBLE NOT NULL,
RAM INT NOT NULL,
Domain Varchar(30) NOT NULL,
MAC-Adresse Varchar(17) INT NOT NULL,
Mitarbeiter-ID INT NOT NULL,
PRIMARY KEY (Laptop-ID),
FOREIGN KEY (Mitarbeiter-ID) REFERENCES
Mitarbeiter(Mitarbeiter-ID));

Erstellen Sie eine SQL-Anweisung, die ermittelt wie viele Laptops mit 4GB RAM oder weniger in den einzelnen Projekten existieren.

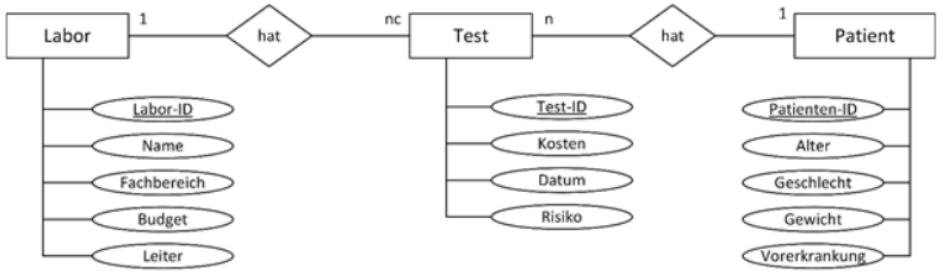
SELECT COUNT(Laptop-ID), Mitarbeiter.Projekt FROM Laptop
INNER JOIN Mitarbeiter ON Mitarbeiter.Mitarbeiter-ID =
Laptop.Mitarbeiter-ID
WHERE Laptop.RAM ¡= 4GB
GROUP BY Mitarbeiter.Projekt;

Erstellen Sie eine SQL-Anweisung, welche die Anzahl der Mitarbeiter pro Büro unterteilt in die Projekte auflistet. Die Büro-Nr. sollen absteigend und die Projekte aufsteigend sortiert sein.

SELECT COUNT(Mitarbeiter.Mitarbeiter-ID),
Mitarbeiter.Projekt, Büro.Büro-Nr FROM Büro
LEFT JOIN Büro-Mitarbeiter-Rel rel ON rel.Büro-Nr =
Büro.Büro-Nr
LEFT JOIN Mitarbeiter ON Mitarbeiter.Mitarbeiter-ID =
rel.Mitarbeiter-ID
GROUP BY Büro.Büro-Nr, Mitarbeiter.Projekt
ORDER BY Büro.Büro-Nr ASC, Mitarbeiter.Projekt DESC;

Erstellen Sie eine SQL-Anweisung, welche die Laptops mit 4GB RAM oder weniger auf 8GB aufrüstet, wenn der zugeordnete Mitarbeiter ein Gehalt über 30.000 hat.

UPDATE Laptop
INNER JOIN Mitarbeiter ON Laptop.Mitarbeiter-Nr =
Mitarbeiter.Mitarbeiter-Nr
SET Laptop.RAM=8
WHERE (Laptop.RAM ¡= 4 AND Mitarbeiter.Gehalt ¿ 30000)



Erstellen Sie eine SQL-Anweisung, welche die Tabelle Test mit ihren zugehörigen Attributen der Datenbank hinzufügt.

```
CREATE TABLE Test (
Test-ID INT NOT NULL,
Kosten DOUBLE NOT NULL,
Datum DATE NOT NULL,
Risiko INT NOT NULL,
Labor-ID INT NOT NULL,
Patienten-ID INT NOT NULL,
PRIMARY KEY (Test-ID),
FOREIGN KEY (Labor-ID) REFERENCES Labor(Labor-ID),
FOREIGN KEY (Patienten-ID) REFERENCES Patient(Patienten-ID));
```

Erstellen Sie eine SQL-Anweisung, welche die Gesamtkosten aller Tests der Patienten mit einem Alter unter 50 ermittelt, die in den Monaten Januar bis Juni 2020 durchgeführt wurden.

```
SELECT SUM(Kosten) FROM Test
JOIN Patient ON Patient.Patienten-ID = Test.Patienten-ID
WHERE Patient.Alter ¡ 50 AND
Test.Datum BETWEEN 2020-01-01 AND 2020-06-30;
```

Erstellen Sie eine SQL-Anweisung, welche alle Test-IDs, das Testdatum und den zugehörigen Labornamen der fünf Labore mit dem größten Budget ausgibt. Dabei sollten nur Ergebnisse gelistet werden, die Tests mit einer Risikostufe größer 2 betreffen.

```
SELECT Test.ID, Test.Datum, Labor.Name FROM Test
INNER JOIN (
SELECT DISTINCT Labor.Name FROM Labor
ORDER BY Labor.Budget DESC
LIMIT 5
) AS Laborliste ON Laborliste.Labor-ID = Test.Labor-ID
INNER JOIN Labor ON Labor.Labor-ID=Laborliste.Labor-ID
WHERE Test.Risiko ¿ 2;
```

Erstellen Sie eine SQL-Anweisung, welche die ältesten 10.000 Testergebnisse löscht, die ein Risiko kleiner gleich 2 besitzen und vor Januar 2020 erstellt wurden oder vor dem Jahr 2016 erstellt wurden.

```
DELETE FROM Test WHERE Test.Risiko ¡ 3 AND Test.Datum ¡ 2020-01-01
OR Test.Datum ¡ 2016-01-01
ORDER BY Test.Datum ASC
LIMIT 10000;
```

CTL-Beispiele

!S0–!S3–!S7 $\rightarrow$ EX S7	Es soll außer von S0, S3 und S7 immer der "Chat mit dem Servicemitarbeiter" direkt gestartet werden können
EX S4	Da Daten zum Einloggen auch gespeichert werden können, soll es optional möglich sein, nach dem Start des Webservice direkt in den Zustand Einloggen (S4) zu gelangen.
$\mathrm{S7} \rightarrow \mathrm{AXS1}$	Da Daten zum Einloggen auch gespeichert werden können, soll es optional möglich sein, nach dem Start des Webservice direkt in den Zustand Einloggen (S4) zu gelangen.
AG EF S3	Am Ende wird der Webservice immer geschlossen.
EG EX a	Auf mindestens einem Pfad folg in jedem Zustand, dass entlang mindestens einem Pfad in im unmittelbar nächsten Zustand a gilt.
$\mathbf{AG}(\mathrm{b} \rightarrow \mathbf{AX}(\mathrm{n} \mid \mathrm{m}))$	Auf allen Pfaden gilt in jedem Zustand, wenn b gilt dann gilt auf allen Pfaden im unmittelbar nachfolgenden Zustand n oder m.
$\mathrm{AG}\ (\mathrm{c} \rightarrow \mathrm{EF!(c\&u)})$	Entlang aller Pfade in allen Zuständen gilt: wenn c gilt, dann gilt auf mindestens einem Pfad irgendwann auch in einem Zustand c und u nicht zusammen.
$\mathbf{AG}(\mathrm{d}\mathbf{U}\mathrm{v}) \mid \mathbf{AX}\mathrm{y}$	Entlang aller Pfade in allen Zuständen gilt: Auf allen Pfaden gilt d bis zum Erreichen von y oder auf allen Pfaden folgt unmittelbar ein Zustand in dem y gilt.
AG „Wasserst. überw."	In allen Zuständen wird "Wasserstand überwachen" durchgeführt.
$\mathrm{AG(S2\mid\ S3\mid S4)} \rightarrow$ EX S5	Sobald Wasser in der Maschine ist muss es immer möglich sein in den "Wasser abpumpen" Zustand zu wechseln.
$\mathrm{AG(S5} \rightarrow \mathrm{AXS1)}$	Wasser abpumpen wird erst beendet, wenn kein Wasser mehr in der Maschine ist
AG(!S6 U S1)	Nur wenn kein Wasser mehr in der Waschmaschine ist, kann der Waschvorgang abgeschlossen werden.
AG AX a	Auf allen Pfaden in jedem Zustand gilt, dass entlang aller Pfaden in jedem nächsten Zustand a gilt (= a gilt immer! solche Formulierung nur halber Punkt!).
$\mathbf{EG}(\mathrm{b} \rightarrow \mathbf{AX}(\mathrm{r} \mid \mathrm{s})\mathbf{\&AX}(\mathrm{s} \rightarrow \mathrm{q}))$	Auf mindestens einem Pfad gilt in jedem Zustand, wenn b gilt dann gilt auf allen Pfaden im unmittelbar nachfolgenden Zustand r oder s und wenn s gilt dann gilt auch q.
$\mathbf{EG}(\mathrm{c} \rightarrow \mathbf{AF}(\mathrm{c\&z}))$	Es existiert mindestens ein Pfad auf dem, wenn c gilt dann gilt, auf allen Pfaden irgendwann auch c und z zusammen.
$\mathrm{:x} \rightarrow \mathrm{A}(\mathrm{d}\mathbf{W}\mathrm{y}) \mid \mathbf{EX}\mathrm{y}$	Wenn nicht x gilt dann gilt d immer oder bis zum Erreichen von y oder auf mindestens einem Pfad folgt unmittelbar y.
$\mathbf{EX}(\mathrm{!e} \rightarrow \mathbf{A}(\mathrm{v}\mathbf{U}\mathrm{w}))$	Entlang mindestens einem Pfad folgt unmittelbar dass wenn nicht e gilt, dann gilt immer v bis zum ersten Auftreten von w und w muss auftreten