Konzeption und Realisierung eines Constraint-basierten Datenbank-Cache

Diplomarbeit von
Christian Merker

Gliederung

- Motivation
- Constraint-basiertes Datenbank-Caching
- Existierende Prototypen
- CBCS: Unser Prototyp
- Ausblick

Motivation

Situation:

Aus Sicht der Unternehmen:

- Internet als neue Verkaufsplattform
- Individuelles Ansprechen der Kunden durch Web-Anwendungen

Aus Sicht der Benutzer:

- Zahl der Breitbandinternetanschlüsse wächst stetig

  ➔ ***Immer mehr Internetbenutzer***

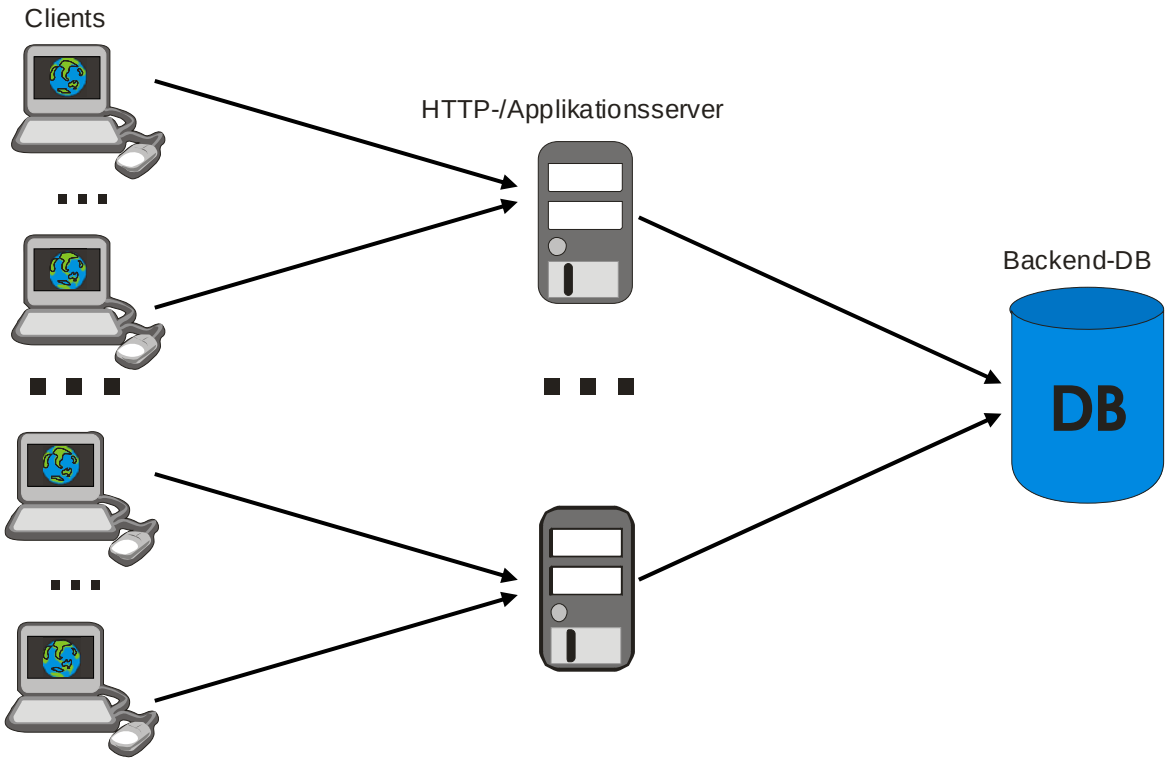
- Hohe Erwartungen an Web-Anwendungen:
  - Kurze Reaktionszeit,
  - Aktuelle Informationen,
  - Anpassung an persönliche Bedürfnisse,
  - …

  ➔ ***Neue Herausforderungen für Anbieter von Web-Anwendungen***

Typische Mehrschichtenarchitektur

- Viele Clients stellen an mehrere Applikationsserver Anfragen
- Mehrere Applikationsserver generieren Web-Seiten aus den gespeicherten Informationen einer Datenbank

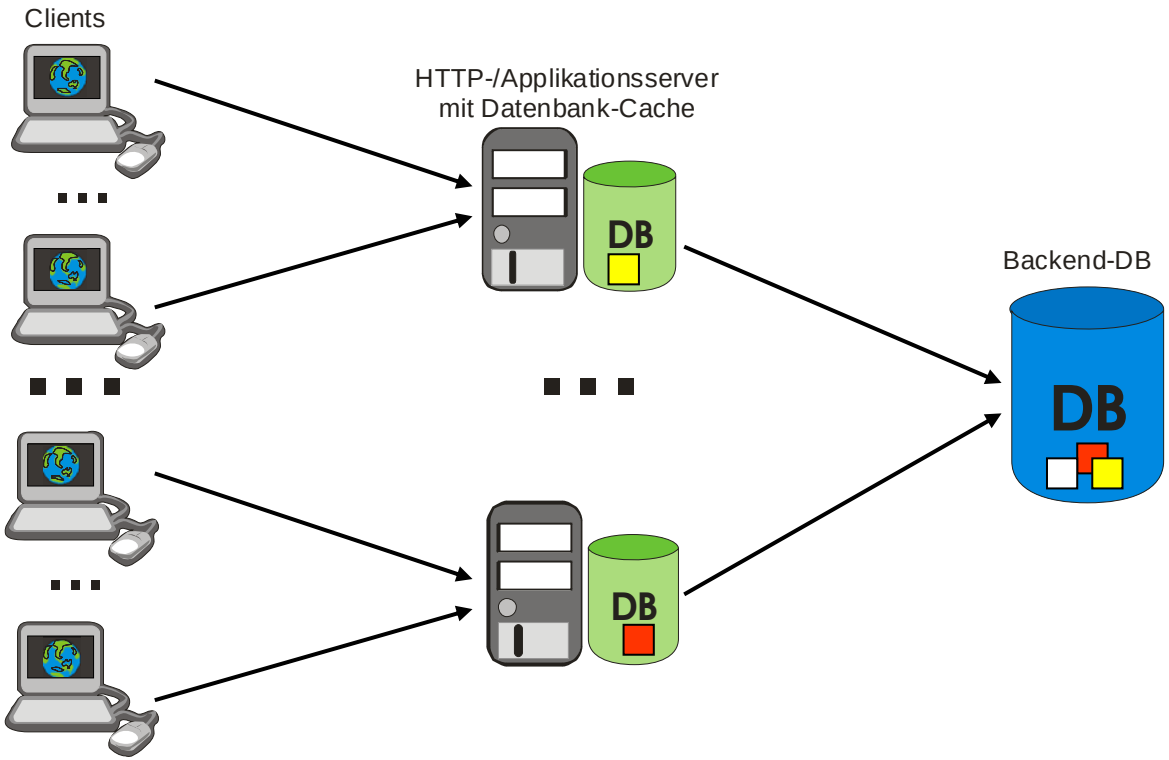
➔ ***Datenbank wird zum Flaschenhals***



Mehrschichtenarchitektur mit DB-Caching

- Jeder Applikationsserver besitzt einen Datenbank-Cache
- Datenbank-Cache enthält Ausschnitt der Daten aus der Backend-DB

➔ ***Informationen aus dem Datenbank-Cache verwenden***



Constraint-basiertes Datenbank-Caching

Ein Cache sollte

- … für den Benutzer transparent sein
- … keine/kaum Änderungen an bestehenden Web-Anwendungen erfordern
- … seinen Inhalt automatisch an die Benutzeranfragen anpassen

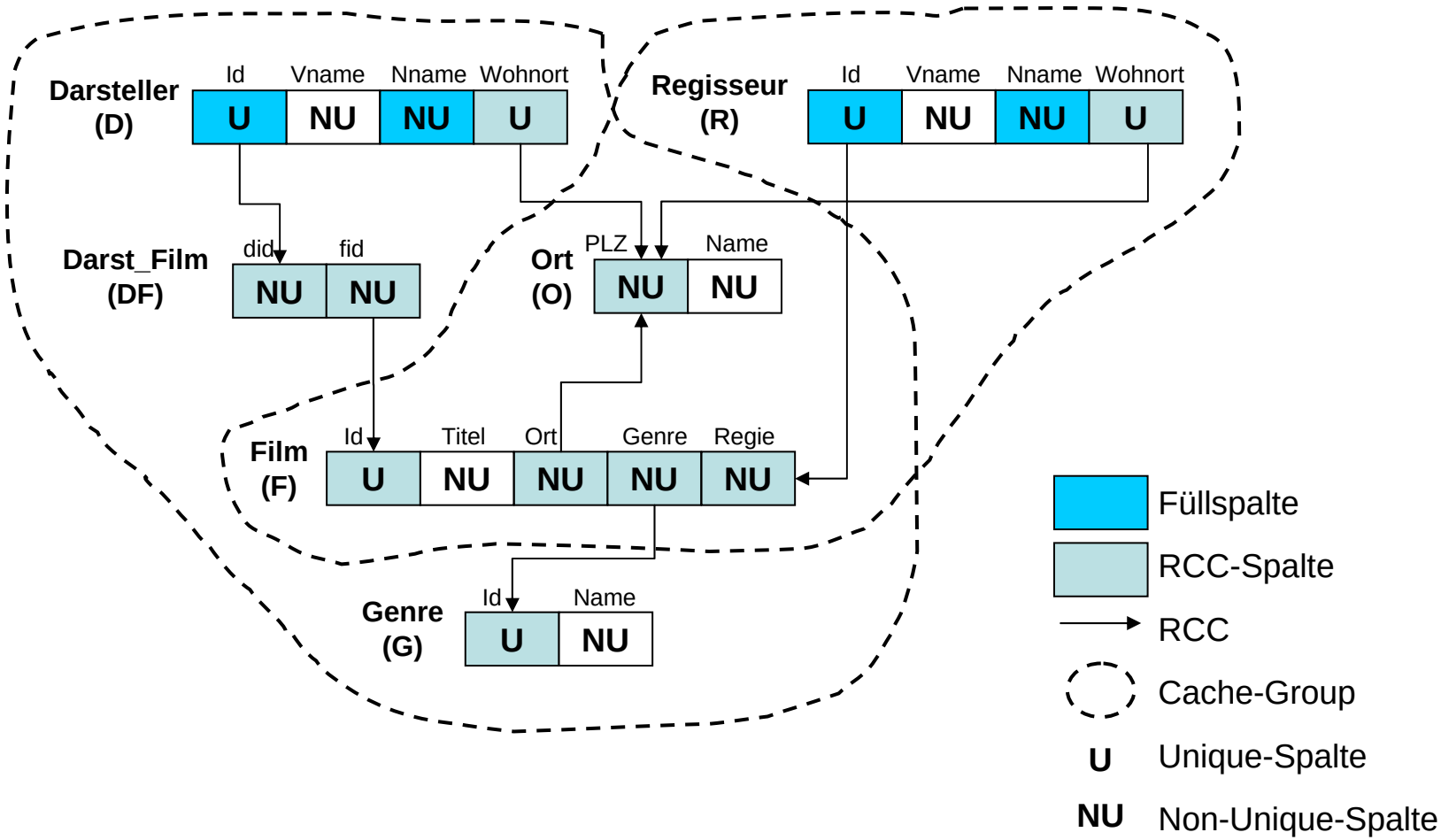
➔ ***Constraint-basiertes Datenbank-Caching erfüllt diese Forderungen***

Cache-Constraints

- Spezifizieren den Inhalt des Cache
- Passen den Cache-Inhalt dynamisch an das Anfrageverhalten an
- Bilden einen Mechanismus, der das Füllen des Cache mit Datensätzen steuert
- Müssen jederzeit eingehalten werden, damit die Anfrageauswertung korrekte Ergebnisse liefert

Definition Cache Group

= Menge von Cache-Tabellen und Cache-Constraints



Definition Füllspalte, RCC

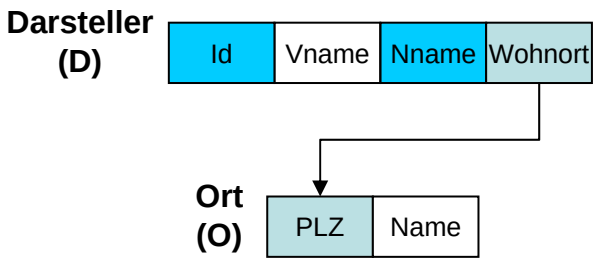
Füllspalte

- Cache-Constraint, der für das Füllen der Cache Group mit Datensätzen verantwortlich ist
- Liste spezifiziert alle Werte *w*, die in den Cache eingelagert werden dürfen
- Cache-Verwaltung sorgt dafür, dass benötigte Datensätze eingelagert werden

Referenzieller Cache-Constraint (RCC)

- Definiert Wertebeziehung zwischen zwei Spalten der Cache-Tabellen
- Gewährleistet, dass für jeden Wert in der RCC-Quellspalte die entsprechenden Join-Partner in der Zieltabelle ebenfalls vorhanden sind

➔ ***RCCs ermöglichen die Auswertung von Anfragen mit Equi-Joins***



Sondierungsverfahren

Sondierung

- Prüfung, ob ein Wert im Cache vorhanden ist (einfacher Existenztest)

```
SELECT *
FROM Regisseur R, Film F
WHERE R.id = '4711' AND R.id = F.regie
```

- Tabelle ist bei positivem Ergebnis Einstiegspunkt in Cache-Group

  ➔ ***Equi-Joins entlang der RCCs können im Cache ausgewertet werden***

- Cache-Verwaltung muß garantieren, daß alle Datensätze mit diesem Wert im Cache vorhanden sind, wenn die Sondierung positiv ausfällt

  ➔ ***Wert muss wertvollständig sein***

Verfahren 1: (Altes Sondierungsverfahren)

- Sondierung wird im Allg. auf Füllspalten und Unique-Spalten durchgeführt

  ➔ ***Alle Werte der Sondierungsspalte müssen wertvollständig sein***

  ➔ ***Sondierungsspalte ist bereichsvollständig***

  ➔ ***Bereichsvollständige Füllspalten notwendig***

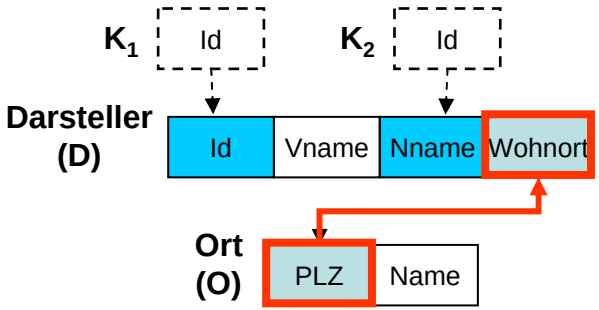
Neues Sondierungsverfahren

Verfahren 2: (Neues Sondierungsverfahren)

- Sondierung wird auf RCC-Quellspalten durchgeführt

  ➔ ***Ein Schritt zurück entlang der RCCs***
- RCCs garantieren Wertvollständigkeit für Werte in der RCC-Zielspalte
- Bereichsvollständige Spalten werden nicht benötigt

```
SELECT *
FROM ORT
WHERE PLZ = 67663
```



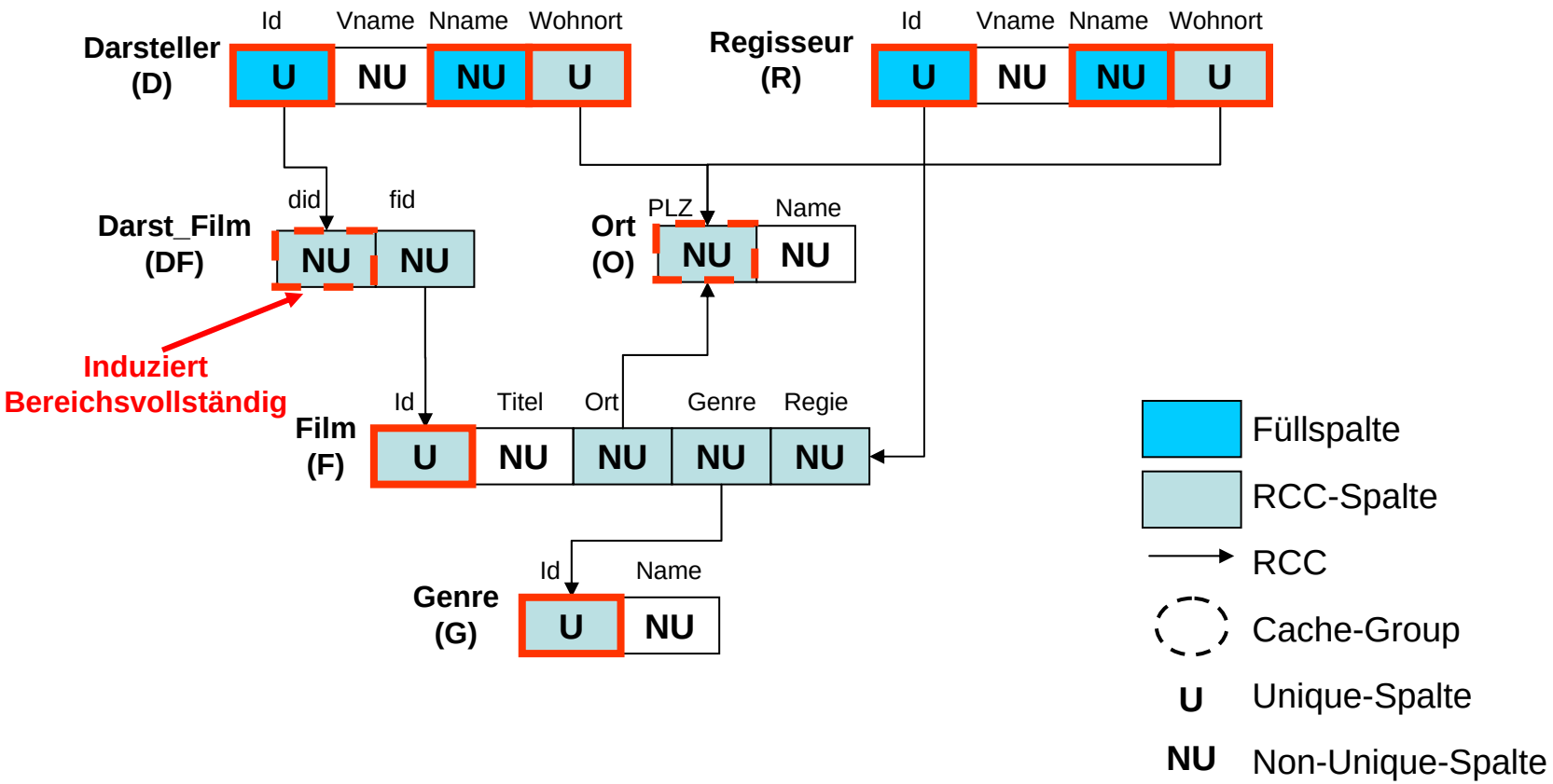
Problem

- Füllspalten benötigen besondere Behandlung

Lösung

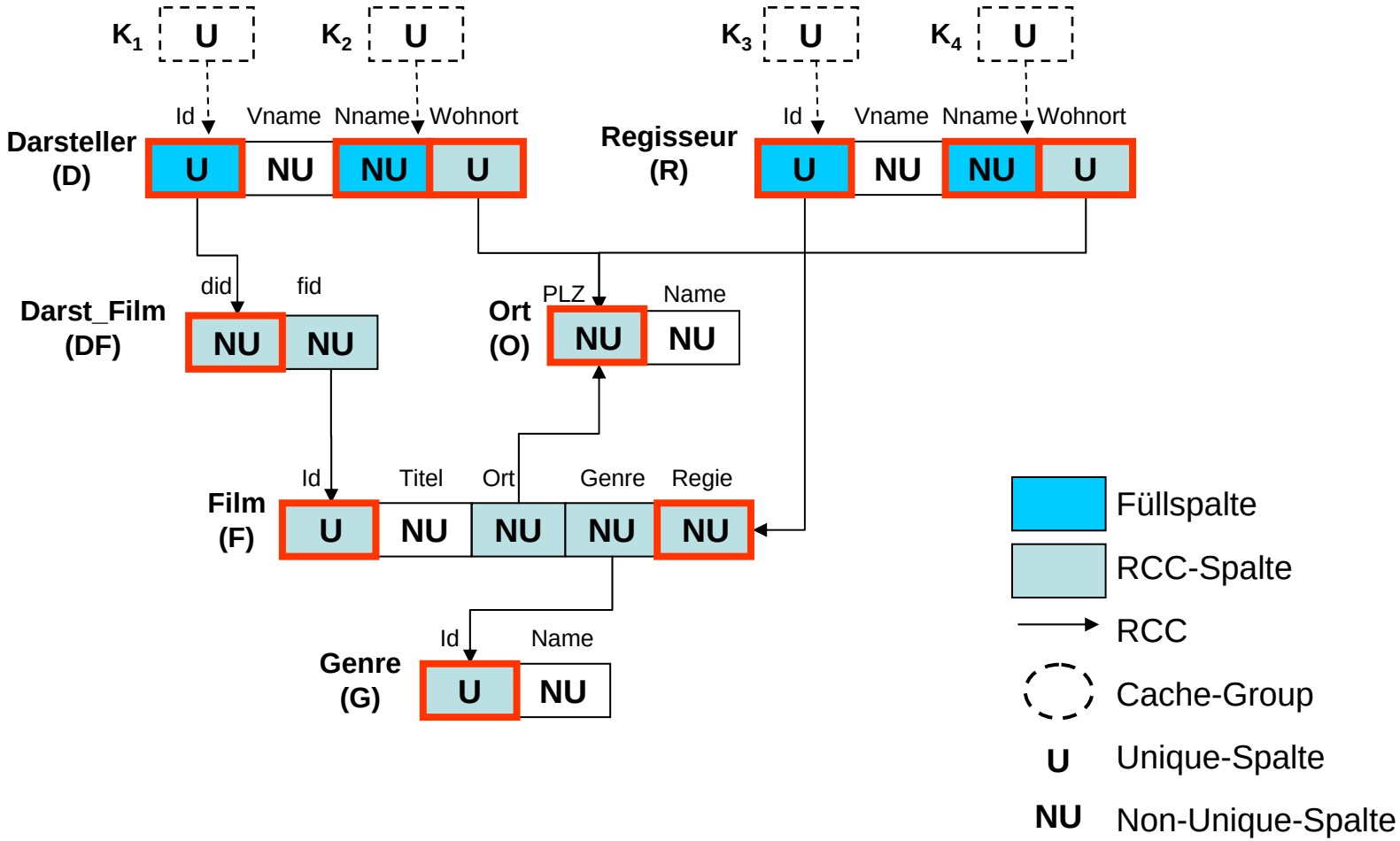
- Einführung von Kontrolltabellen
- Nur Werte in den Kontrolltabellen werden vollständig gemacht

Einstiegspunkte: Altes Verfahren



➔ ***10 potentielle Einstiegspunkte in die Cache-Group***

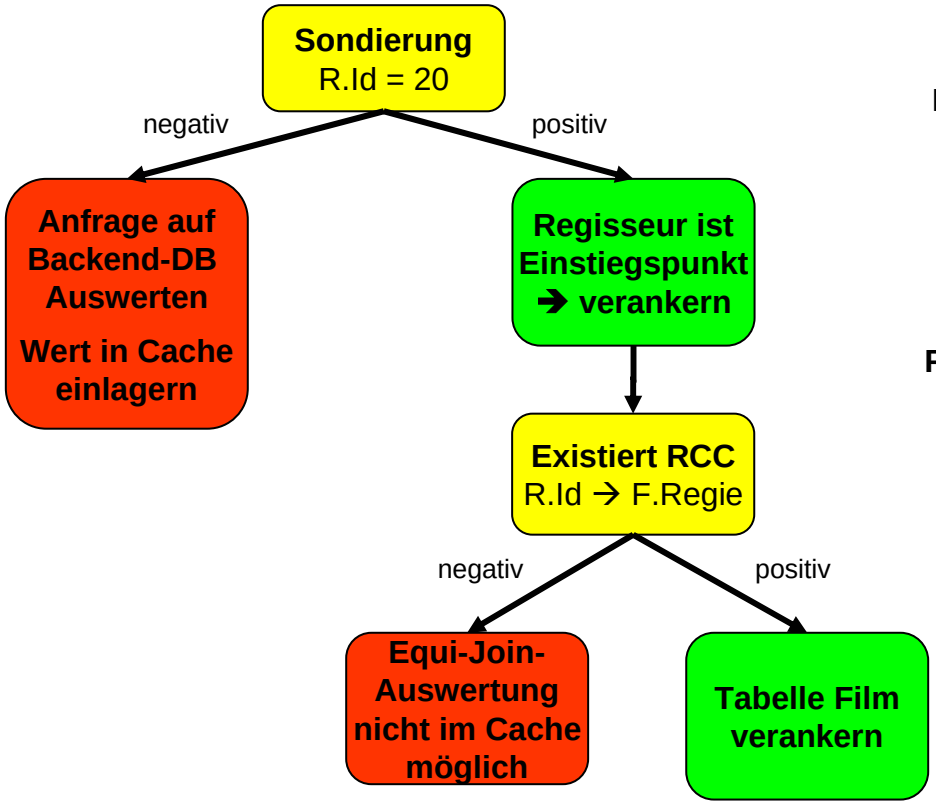
Einstiegspunkte: Neues Verfahren

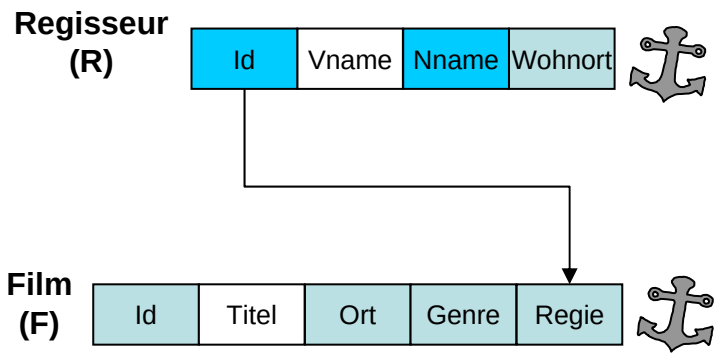


➔ ***11 potentielle Einstiegspunkte in die Cache-Group***

Anfragebearbeitung

Anfrage:
```
SELECT *
FROM Regisseur R, Film F
WHERE R.Id = 20 AND R.Id = F.Regie
```



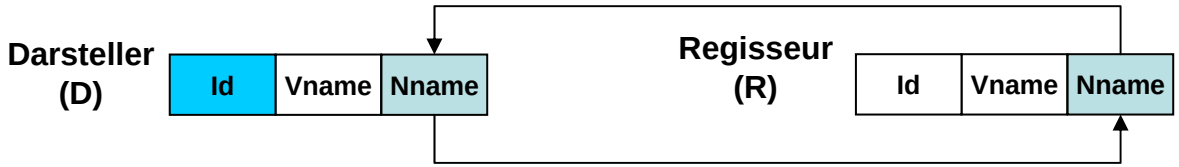


RCC-Zyklen

Homogene Zyklen:

- Enden nach einem Durchlauf

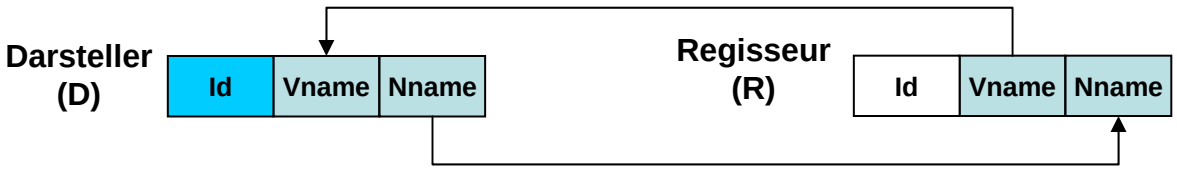
➔ ***Sind sicher, d. h. es entstehen keine rekursiven Ladevorgänge***



Heterogene Zyklen:

- Sind im Allg. nicht sicher, d. h. sie verursachen rekursive Ladevorgänge

➔ ***Keine heterogene Zyklen in Cache-Group-Definition***



Existierende Prototypen: MTCache

- Entwickelt von Microsoft
- Cache auf Basis von materialisierten Sichten
- Werte in den Sichten durch Kontrolltabellen spezifiziert
- Cache besitzt Schattendatenbank, zur Erstellung des Anfrageplans
- Erweiterung des SQL-Sprachumfangs
  (zur Beschreibung von Konsistenzanforderungen)

Leseoperationen

- Kostenbasierter Optimierer erstellt Anfrageplan
- Anfrageplan enthält Cache- und Backend-Tabellen

Änderungsoperationen

- Änderungsoperationen werden in Backend-Datenbank ausgeführt
- Cache bleibt durch regelmäßige Schnappschüsse aktuell

Existierende Prototypen: DBCache

- Constraint-basierter Datenbank-Cache von IBM
- Erweiterungen im DB2-Datenbanksystem (neuer Objekttyp: Cache Table)
- Verwendung von Funktionen zum Zugriff auf föderierte Datenbanken
- Einsatz des alten Sondierungsverfahrens
- Überprüfung der potentiellen Einstiegspunkte in einer Existenz-Anfrage

```
SELECT * FROM R, D WHERE R.Id = 20 AND D.Nname = 'Müller,
➔ EXISTS(R.Id = 20) AND EXISTS(D.Nname = 'Müller')
```

- Daemon lagert benötigte Datensätze in Cache ein (asynchron)

Leseoperationen

- Janus-Plan: Zwei Anfragepläne (Local- und Remote-Plan)
- Switch-Operator entscheidet welcher Anfrageplan ausgeführt wird

Änderungsoperationen

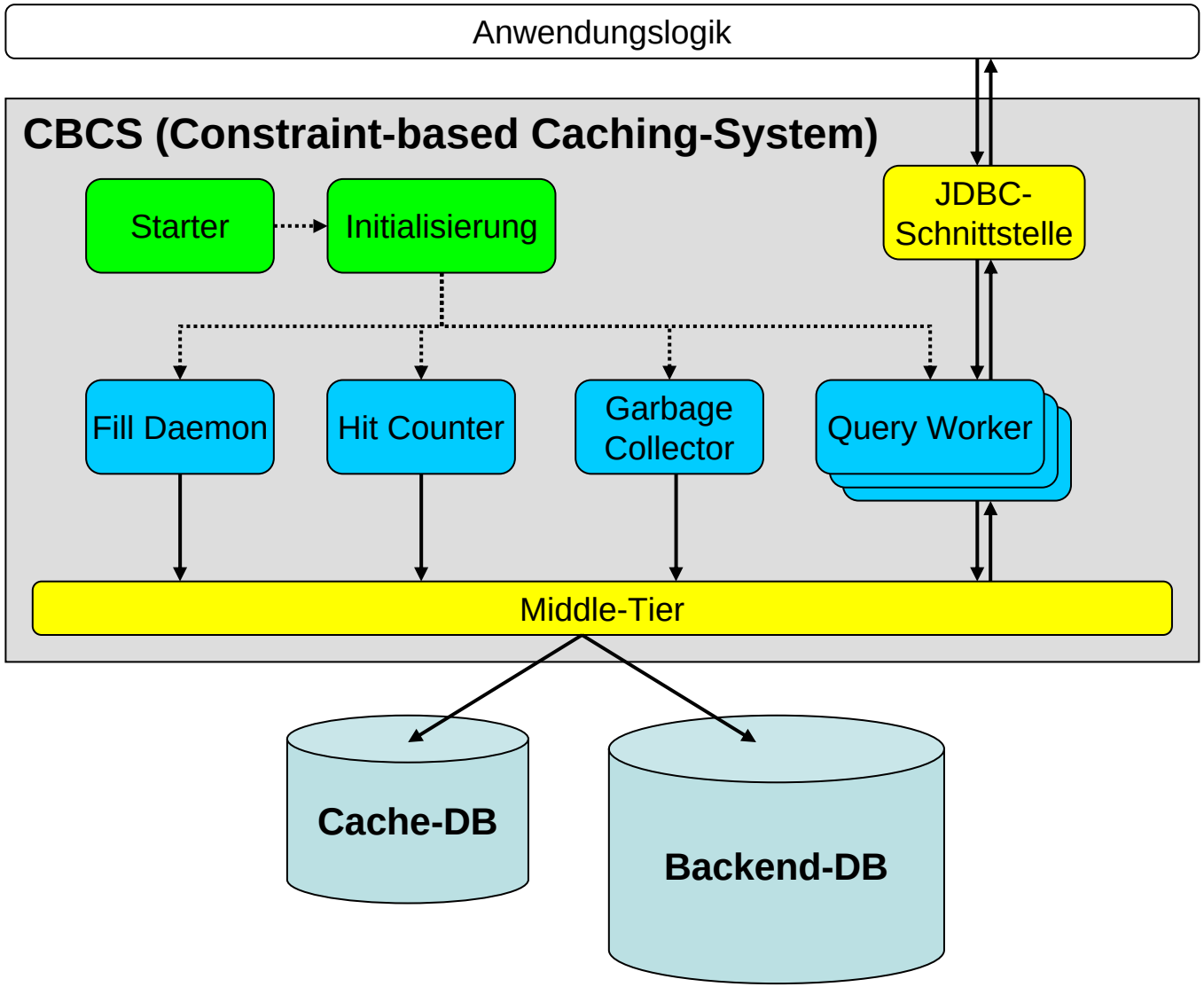
- Werden auf der Backend-Datenbank ausgeführt
- Änderungen werden durch Nachrichten an Cache propagiert (asynchron)

Vergleich der Prototypen

Stärken und Schwächen von MTCache und DBCache

MTCache	DBCache
+ Transparenter Cache	+ Transparenter Cache
+ Kontrolltabellen verhindern Einlagerung von Werten mit geringer Selektivität	- Füllspalten sind bereichsvollständig (wegen alten Sondierungsverfahren). Existenz einer Liste mit erlaubten Werten zum Einlagern fehlt.
- Kontrolltabellen von Administrator verwaltet. (Nicht bekannt, ob automatische Anpassung der Werte)	+ Einlagerung von Datensätzen geschieht nach Bedarf
	- Sondierung: Nur eine Existenzanfrage mit konjunktiver Verknüpfung aller potentiellen Einstiegspunkte
- Kann wegen Erweiterung der SQL-Syntax nur auf Microsoft SQL-Server eingesetzt werden	- Durch die Integration in die IBM DB2 kann kein anderes DBMS als Cache eingesetzt werden

CBCS: Überblick



CBCS: Initialisierungskomponente

- Bereitet alle Komponenten für den Start vor
- Behandelt Fehler während der Initialisierung

Zwei Initialisierungsarten

- ***Normale Initialisierung***
  - Auslesen der Konfigurationsdatei
  - Erzeugung der internen Datenstruktur (Tabellen, Spalten, RCCs)
  - Erstellen von Prepared Statements
  - Initialisieren der übrigen Komponenten (Query Worker, Fill Daemon, …)
- ***Erweiterte Initialisierung (Ur-Initialisierung)***
  - Anlegen der spezifizierten Cache-Tabellen im Cache
  - Anlegen der Kontroll- und Füllwert-Tabellen im Cache
  - Anlegen von Indizes auf allen potentiellen Einstiegspunkten

CBCS: Middle-Tier-Komponente

- Verwaltet die Verbindungen zur Cache- und Backend-Datenbank
- Bietet eine homogene Sicht auf die Tabellen
- Stellt den übrigen Komponenten Schnittstellen zum DB-Zugriff bereit

Vorteil:

- Cache- und Backend-Datenbank können leicht ausgetauscht werden
- Im CBCS muss „*nur*“ die Middle-Tier-Komponente angepasst werden

CBCS: Query Worker

- Kern des CBCS
- Analysiert und Verarbeitet Benutzeranfragen

Bearbeitung von Leseoperationen

- Anfrage gegen unterstützte SQL-Syntax validieren
- Prädikate der Where-Klausel extrahieren
- Für jeden potentiellen Einstiegspunkt Sondierung durchführen (neues Sondierungsverfahren)
- Anfrage modifizieren
  d. h. Backend-Tabellen durch verankerte Cache-Tabellen ersetzen
- Modifizierte Anfrage ausführen und Ergebnis zurückliefern

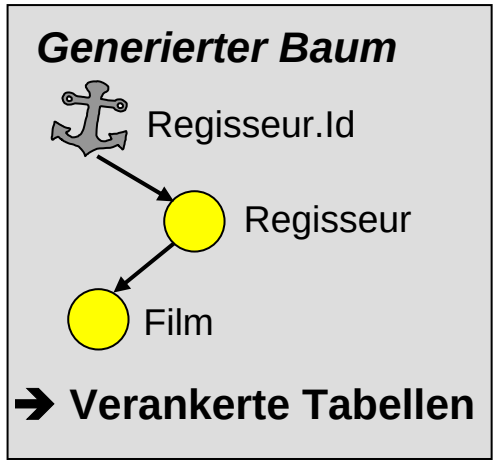
Bearbeitung von Änderungsoperationen

- Wird noch nicht unterstützt

Anfragebearbeitung im Query Worker

```
SELECT r.Name, Film.Titel
FROM Regisseur r, Film
WHERE r.Id = Regie
AND r.Id = '4711'
```

Einstiegspunkte suchen



Anfrage modifizieren

```
SELECT r.Name, CA_FILM.Titel
FROM CA_REGISSEUR r, CA_FILM
WHERE r.Id = Regie
AND r.Id = '4711'
```

Anfrage ausführen; Ergebnis zurückgeben

Middle-Tier-Komponente

CBCS: Fill Daemon

- Läuft in einem eigenständigen Thread
- Lagert Datensätze in den Cache ein
- Wird vom Query Worker mittels Nachrichten über fehlende Werte (in einer Füllspalte) informiert
- Einlagerung geschieht asynchron zur Arbeit der Query Worker (eigenständiger Thread)
- Vorgefertigte Insert-Anweisungen (PreparedStatements)

Untersuchte Einlagerungsstrategien

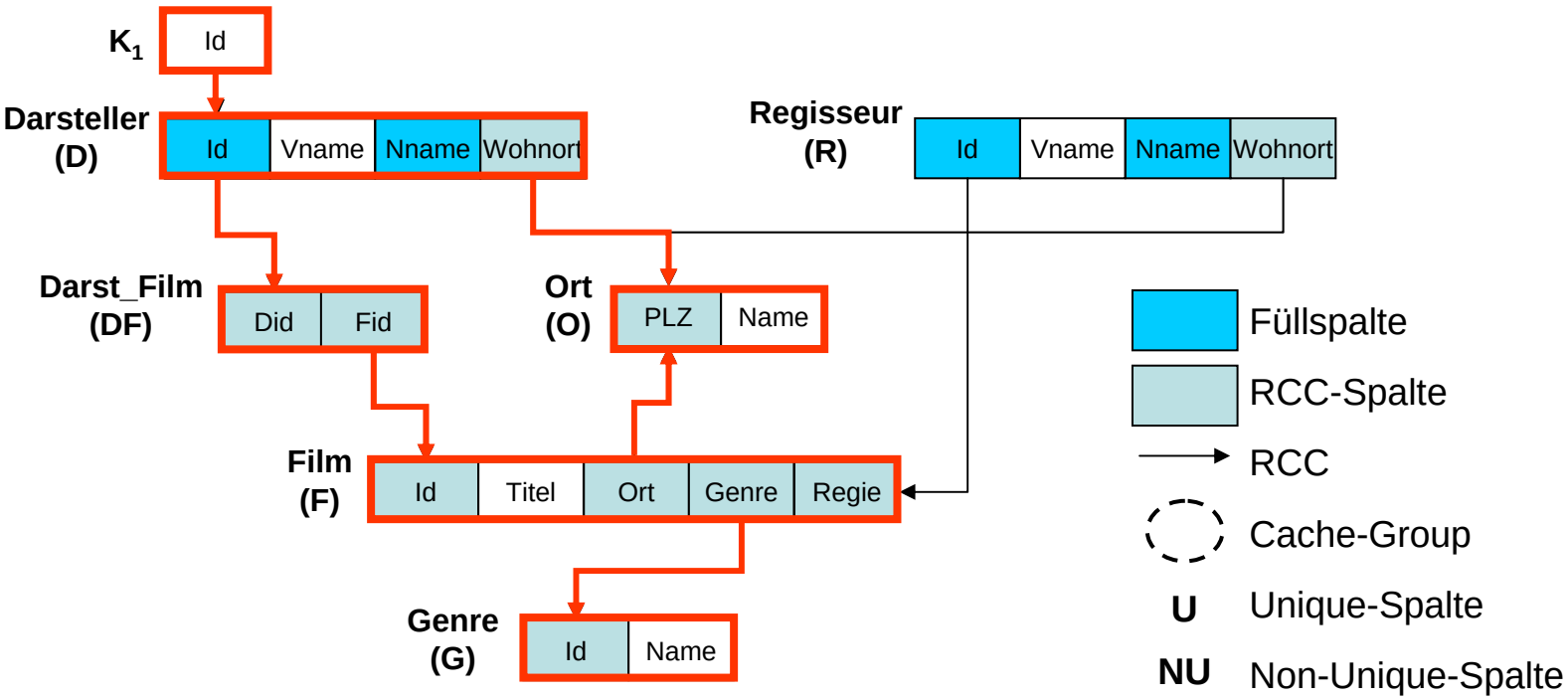
- Top-Down
- Bottom-Up

Einlagerungsstrategien: Top-Down

- ***Wichtig: Alle Cache-Constraints müssen nach jedem Einlagerungsvorgang wieder erfüllt sein !!!***
- Einlagerungsvorgang startet auf der Kontrolltabelle
- Rekursiver Abstieg entlang der ausgehenden RCCs

ABER: Gesamte Einlagerung muss innerhalb einer TA geschehen

➔ ***Lange Schreibsperren auf den Cache-Tabellen***



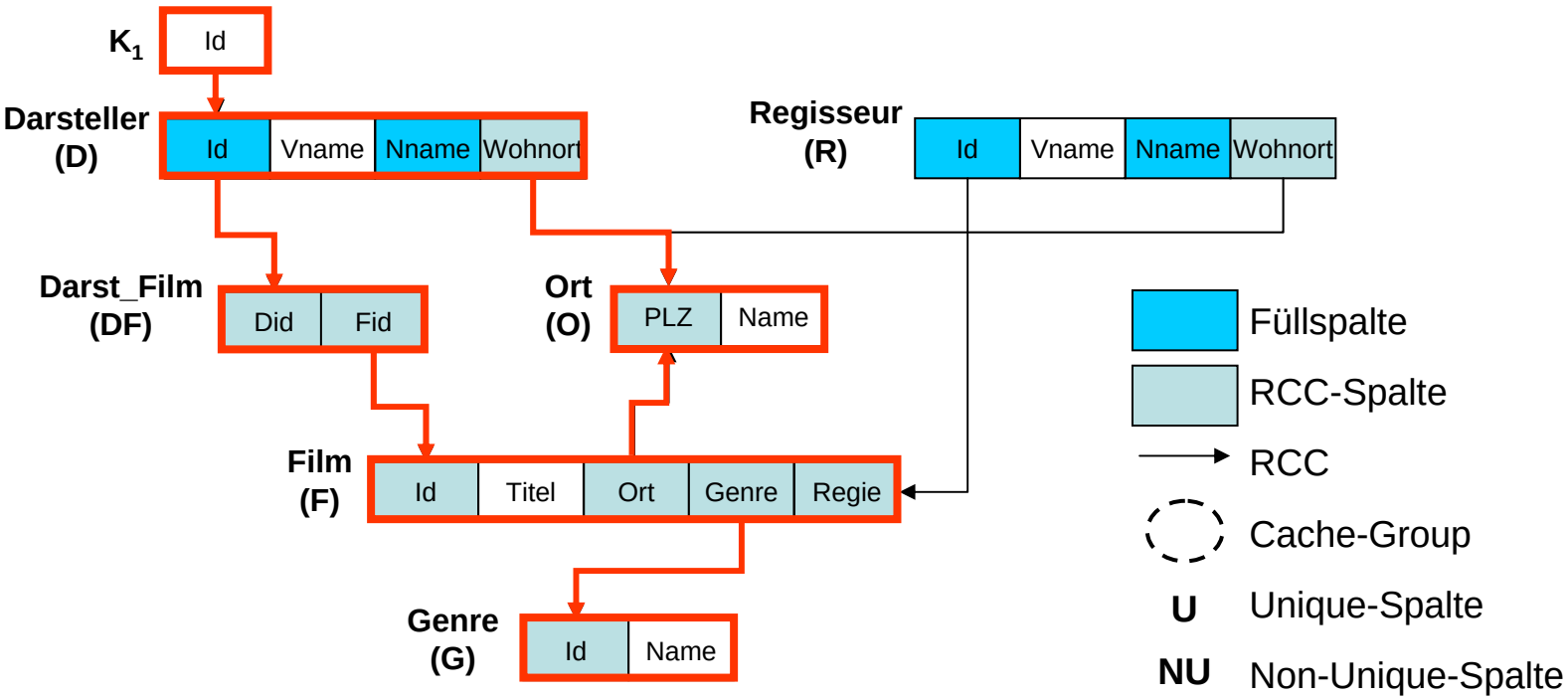
Einlagerungsstrategien: Bottom-Up

- Einlagerungsvorgang verläuft in umgekehrter Reihenfolge
- Topologische Sortierung notwendig
- Rekursiver Aufstieg entlang der RCCs
- Vorgang endet mit Füllen der Kontrolltabelle

➔ ***Füllvorgang jeder Tabelle in eigener TA möglich***

➔ ***Kurze Schreibsperren***

➔ ***Keine unnötige Blockierung der Query Worker***



Erzeugung der Füllanweisungen

(während der Initialisierungsphase)

1. Cache-Tabellen in atomare Zonen einteilen
   a. Initialisierung: Jede Cache-Tabelle in eigener atomarer Zone
   b. Anschließend alle RCC-Pfade verfolgen
   c. RCC-Zyklus gefunden: Alle am Zyklus beteiligten Tabellen in eine Zone zusammenfassen
2. Atomare Zonen topologisch sortieren
3. Insert-Anweisungen für Kontrolltabelle generieren
4. Rekursiv alle ausgehenden RCCs verfolgen;
   Insert-Anweisung für jede erreichte Tabelle generieren
5. Tabelle besitzt keine ausgehenden RCCs
   → Auf höherer Rekursionsebene fortfahren

Bei homogenem Zyklus:

- Alle ausgehenden, nicht am Zyklus beteiligten, RCCs verfolgen

Bei heterogenem Zyklus:

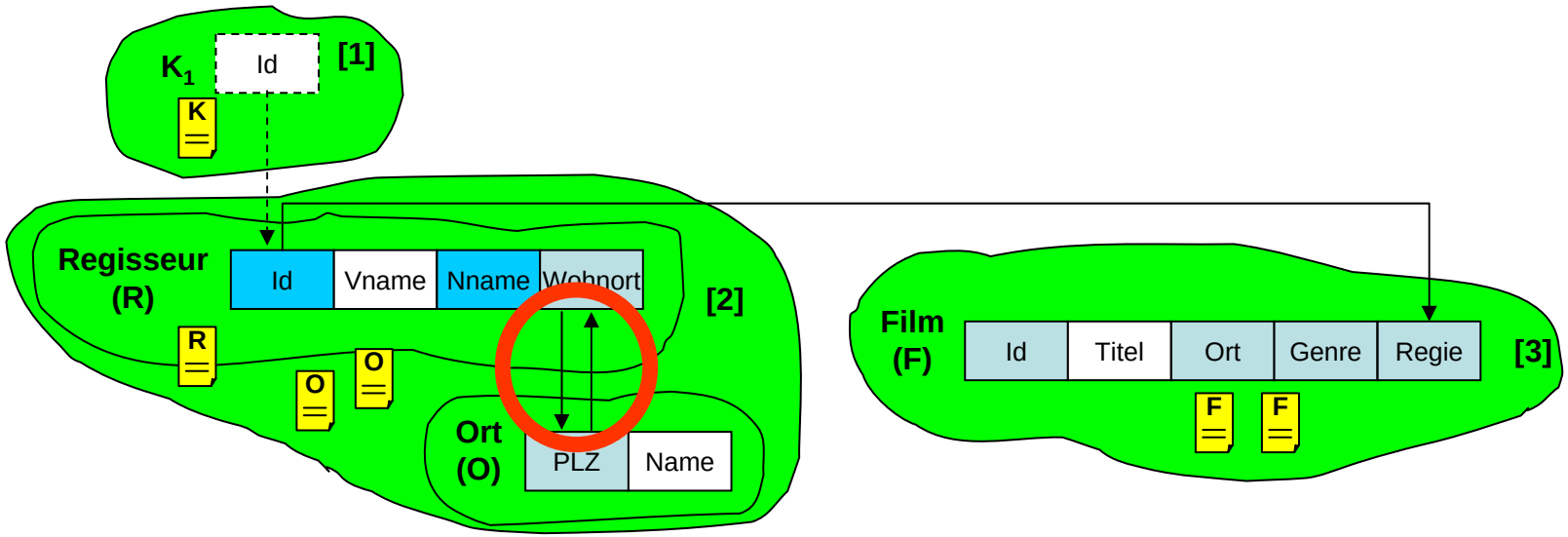
- Generierung mit Fehlermeldung abbrechen

Atomare Zonen

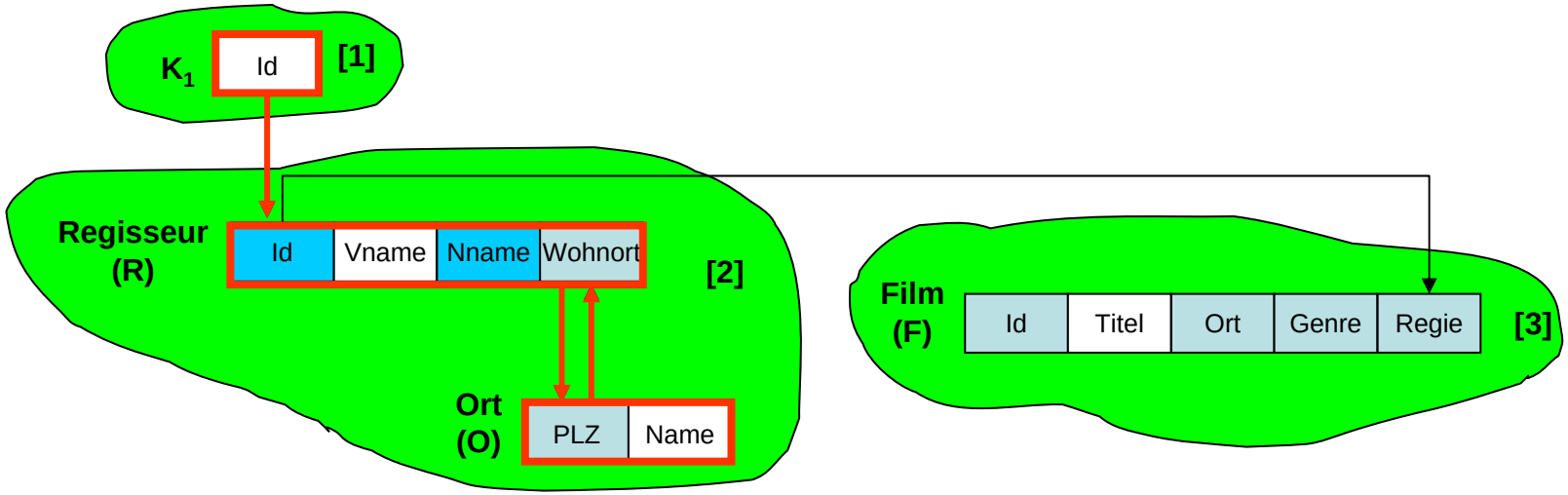
- Bottom-Up-Verfahren: Probleme bei zyklischer Referenzierung
  - ➔ ***RCCs zwischen Cache-Tabellen werden verletzt***
  - ➔ ***Alle Cache-Tabellen im Zyklus müssen in einer TA gefüllt werden***

Lösung

- Einführung von atomaren Zonen
- Atomare Zone enthält eine oder mehrere Tabellen
- Tabellen einer Zone werden in einer TA gefüllt



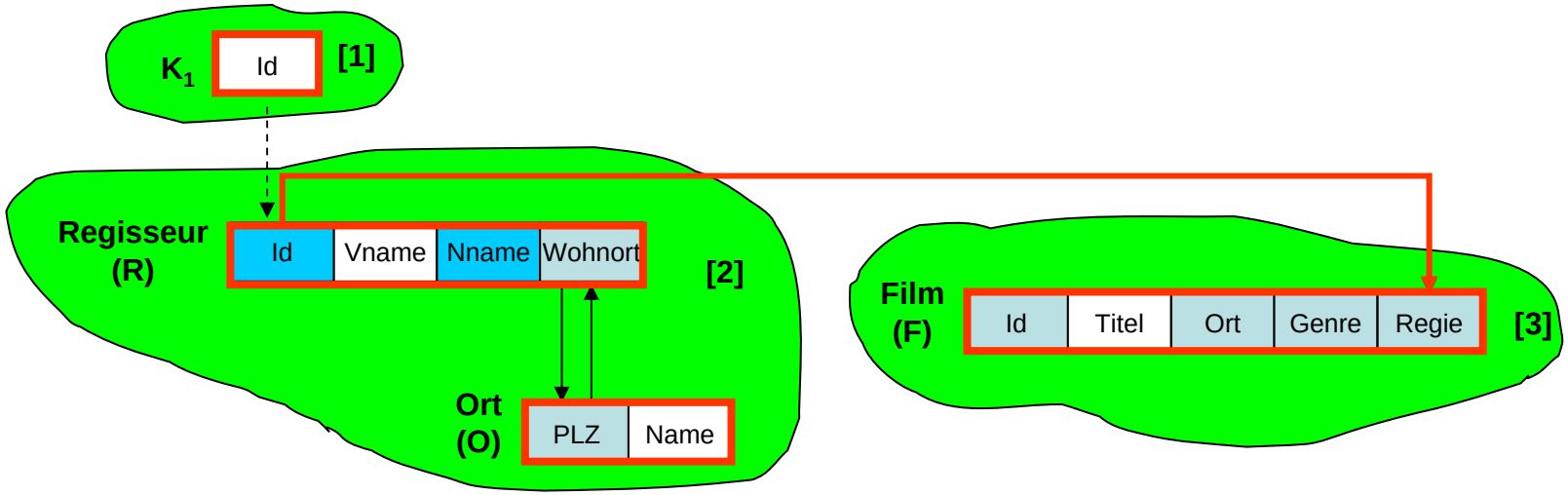
Generierung der Füllanweisungen



Generierte Anweisungen:

```
1. INSERT INTO K1 (ID, TIME, LASTACCESS, HITCOUNTER) VALUES (?, ?, ?, 0)
    2. INSERT INTO REGISSEUR
            SELECT * FROM REGISSEUR_B
            WHERE ID = ?
            AND (ID) NOT IN (SELECT ID FROM REGISSEUR)
       3. INSERT INTO ORT
               SELECT * FROM ORT_B
               WHERE PLZ IN (SELECT WOHNORT FROM REGISSEUR_B WHERE ID = ?)
               AND (PLZ, NAME) NOT IN (SELECT PLZ, NAME FROM ORT)
          4. INSERT INTO REGISSEUR
                  SELECT * FROM REGISSEUR_B
                  WHERE WOHNORT IN (SELECT PLZ FROM ORT_B
                       WHERE PLZ IN (SELECT WOHNORT FROM REGISSEUR_B WHERE ID = ?)
                  AND (ID) NOT IN (SELECT ID FROM REGISSEUR)
                                  ...
```

Generierung der Füllanweisungen (Fort.)

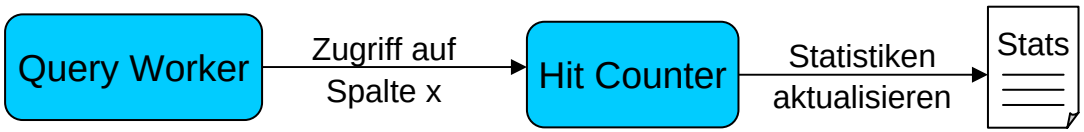


Generierte Anweisungen:

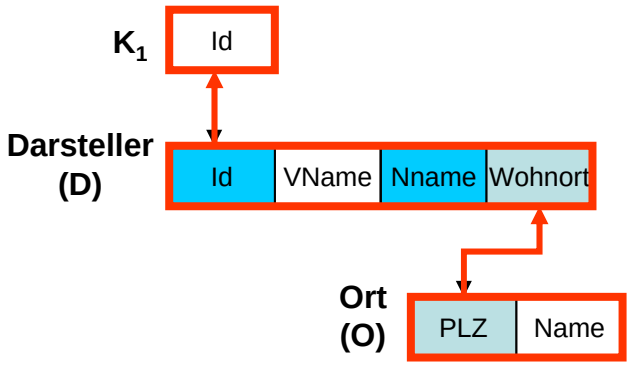
```
1. INSERT INTO K1 …
    2. INSERT INTO REGISSEUR …
        3. INSERT INTO ORT …
            4. INSERT INTO REGISSEUR …
                5. INSERT INTO FILM …
        6. INSERT INTO FILM …
```

CBCS: Hit Counter

- Läuft in einem eigenständigen Thread
- Sammelt Statistikdaten, die der Garbage Collector zur Verdrängung von Datensätzen benötigt
- Query Worker informiert Hit Counter über jeden gefundenen Einstiegspunkt
- Hit Counter aktualisiert die Statistiken in den Kontrolltabellen, die für die Einlagerung des Einstiegspunkt verantwortlich sind
- Vorgefertigte Update-Anweisungen (für jeden potentiellen Einstiegspunkt)



Generierung der HitCounter-Anweisungen

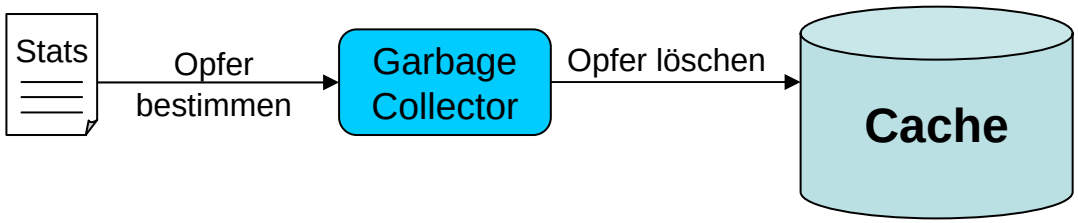


Where-Klausel:

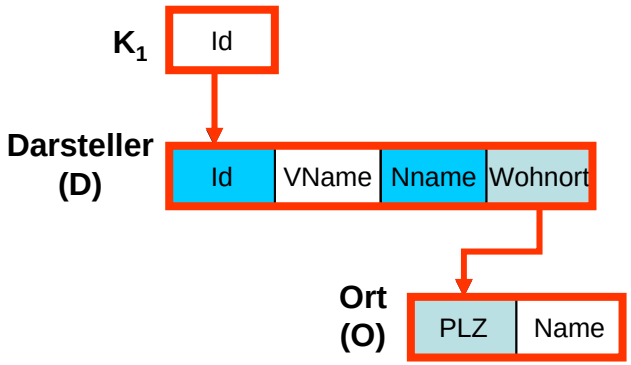
1. `WHERE PLZ = ?`
2. `WHERE WOHNORT IN (SELECT PLZ FROM ORT WHERE PLZ = ?)`
3.
```
UPDATE K1 SET lastAccess = ?, hitcounter = hitcounter + 1
  WHERE (ID IN
     (SELECT ID FROM DARSTELLER
        WHERE WOHNORT IN (SELECT PLZ FROM ORT WHERE PLZ = ?)))
```

CBCS: Garbage Collector

- Läuft in einem eigenständigem Thread
- Überprüft periodisch, ob Cache spezifizierte Füllmarke überschreitet
- Löscht bei Überschreitung Datensätze aus dem Cache
- „Opfer“ wird anhand der Statistikdaten bestimmt (LRU-Strategie)
- Löscht Opfer aus Kontrolltabelle
- Anschließend rekursives Löschen entlang der ausgehenden RCCs (nur nicht referenzierte Datensätze werden gelöscht)
- Top-Down-Ansatz (Löschvorgang beginnt auf der Kontrolltabelle)
  - ➔ ***keine langen Schreibsperren***
  - ➔ ***Cache besitzt nach jeder Löschoperation einen gültigen Zustand***



Generierung der Löschanweisungen

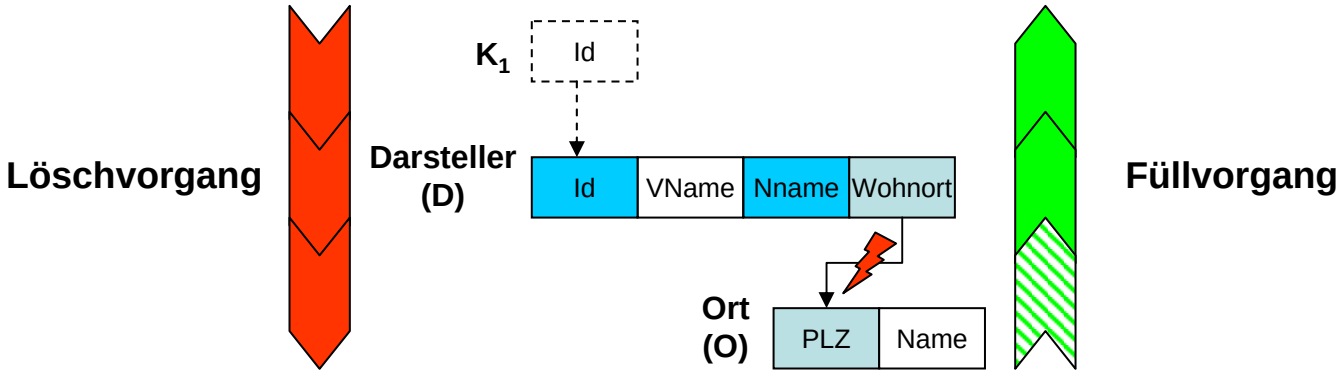


Generierte Anweisungen:

1. `DELETE FROM K1 WHERE ID = ?`
2. ```
   DELETE FROM DARSTELLER
     WHERE (ID NOT IN (SELECT ID FROM K1))
   ```
3. ```
   DELETE FROM ORT
     WHERE (PLZ NOT IN (SELECT WOHNORT FROM DARSTELLER))
   ```

Aber: Zyklische Referenzierung verhindert das Löschen von Datensätzen

Gleichzeitiges Löschen und Einlagern



➔ ***Ungültiger Cache-Zustand, da RCC D.Wohnort → O.PLZ verletzt ist***

Lösung: Einführung von Exklusivsperren

➔ ***Cache darf zu einem Zeitpunkt nur von Fill-Daemon- oder Garbage-Collector-Instanzen bearbeitet werden***

CBCS: JDBC-Schnittstelle

- Schnittstelle zwischen Benutzer und CBCS
- Einfache Einbindung in (bestehende) Web-Anwendungen
- Nimmt Anfragen des Benutzers entgegen
- Leitet alle Anfragen, für den Benutzer transparent, an die Query Worker weiter
- Stellt die Ergebnisse dem Benutzer in gewohnter Weise zur Verfügung

➔ **Der Benutzer bemerkt die Existenz des CBCS nicht**

Zusammenfassung

Motivation

- Datenbank ist Flaschenhals in typischer Mehrschichten-Architektur

Constraint-basiertes Datenbank-Caching

- Cache Group, Füllspalten, RCC, Sondierungsverfahren
- Einstiegspunkte und Anfrageauswertung

Existierende Prototypen

- DBCache und MTCache
- Stärken-Schwäche-Analyse

CBCS-Prototyp

- Übersicht über die Komponenten
- Query Worker, Fill Daemon, Hit Counter, Garbage Collector
- Anfragebearbeitung
- Einlagerungsstrategie (Bottom-Up)
- Löschstrategie (Top-Down)
- Wechselwirkung von Füll- und Löschvorgängen

Wie geht es weiter?

Weiterentwicklung in vielen Bereichen möglich:

Funktionale Erweiterungen

- Unterstützung von Änderungsoperationen
- Erweiterung der unterstützten SQL-Syntax

Verbesserung der Performance

- Ausführliche Testreihen

➔ Optimierungen am Programmcode

➔ Entwicklung effizienterer Komponenten

Erweiterung der Benutzerfreundlichkeit

- Entwicklung einer graphischen Benutzeroberfläche zur Steuerung des CBCS