Data Warehousing und Data Mining

Speicherung multidimensionaler Daten

Ulf Leser

Wissensmanagement in der Bioinformatik

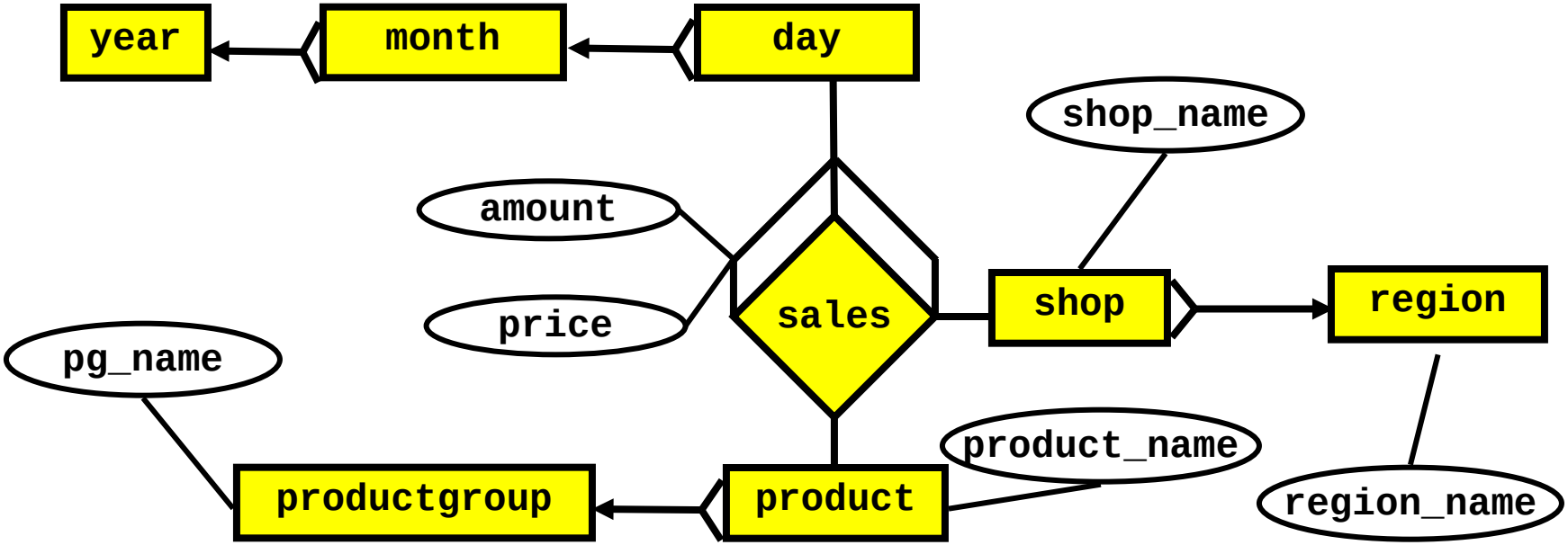
Inhalt dieser Vorlesung

- Relationales OLAP (ROLAP)
  - Snowflake-, Star- und Fullfactschema
  - Speicherplatz und Queries
  - Schemavarianten
- Beispiel: ROLAP in Oracle
- Evolution in Dimensionen
- Multidimensionales OLAP (MOLAP)

Relationales OLAP

- MDDM und relationales Modell nicht vergleichbar
  - Konzeptionelle versus logische Ebene
  - Falsch: „Relationales Modell ist zweidimensional, MDDM ist beliebig-dimensional, das ist inkompatibel“
- Relationales Modell kann beliebig-dimensionale Daten repräsentieren
- Verschiedene Arten der Abbildung möglich
  - Unterschiede in Speicherverbrauch, Redundanz, Performance von INSERT/SELECT/UPDATE, ...

Beispiel



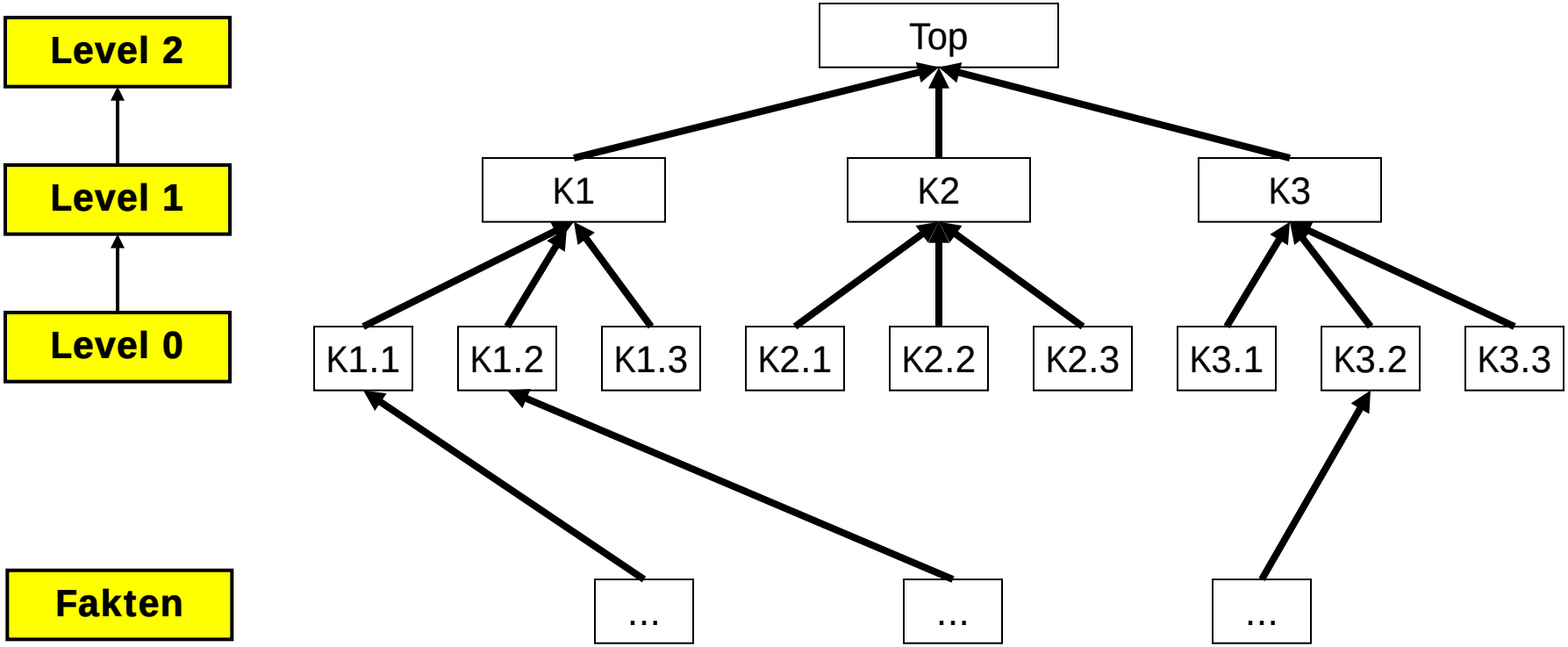
Summe aller Verkäufe
der Produktgruppe „Wasser“
nach Shop und Jahr

Annahmen für Kostenbetrachtungen

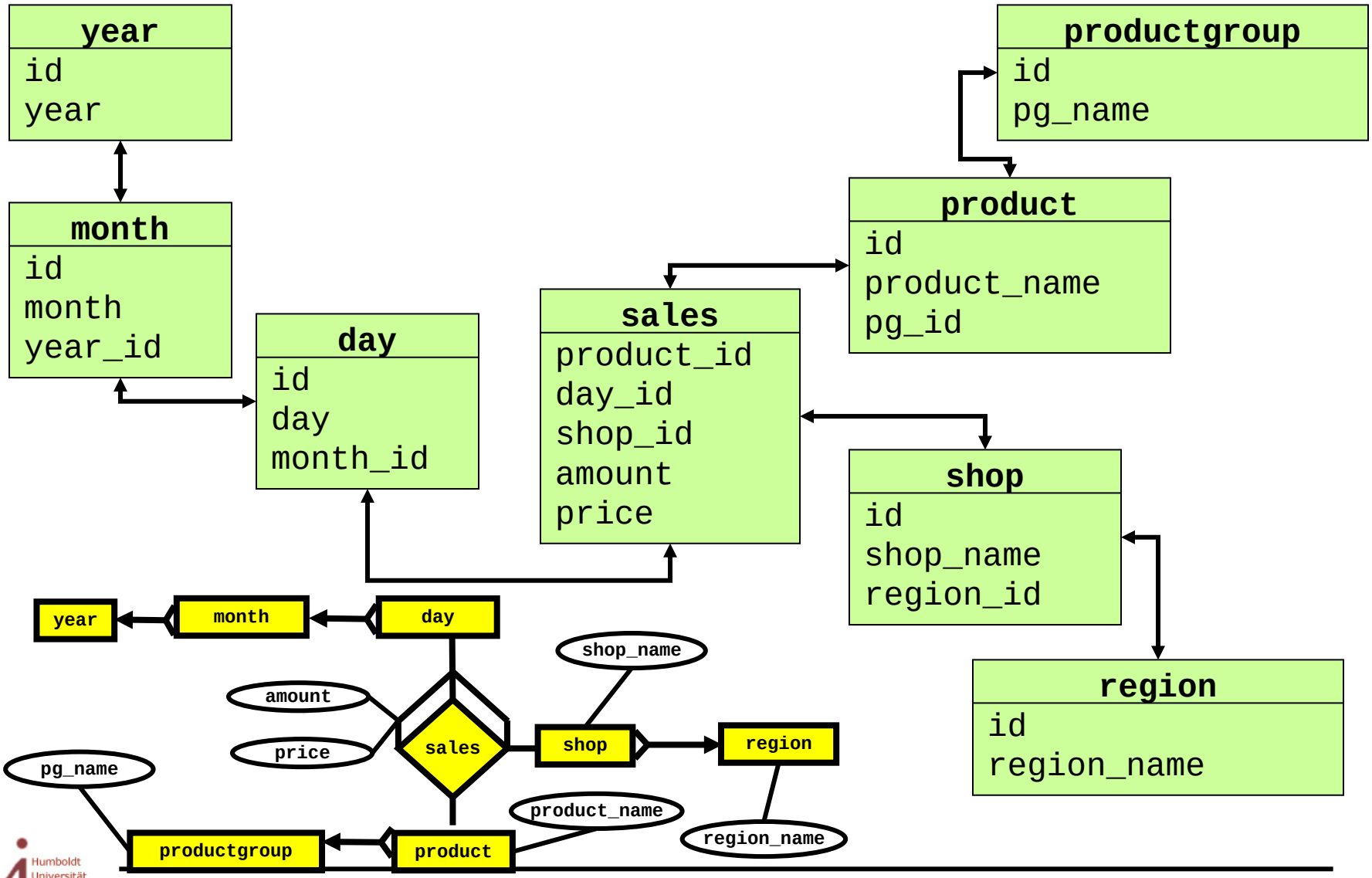
- d Dimensionen, je k Klassifikationsstufen plus Top
- Jeder Klassifikationsknoten hat 3 Kinder
  - Level k: $1=3^0$ Knoten (Top)
  - Level k-1: $3=3^1$ Knoten
  - ...
  - Level 0: Höchste Granularität, 3^k Knoten
- Zusammen: n= $\sum_{i=0...k} 3^i$ Knoten pro Dimension
- m Fakten, gleichverteilt in allen Dimensionen
- Attribut: b Bytes
- Knoten haben nur ID
- Es gibt f Measures

Gleichmäßige Klassifikationshierarchie

Zu jedem Knoten gibt es gleich viele Kinder/Fakten



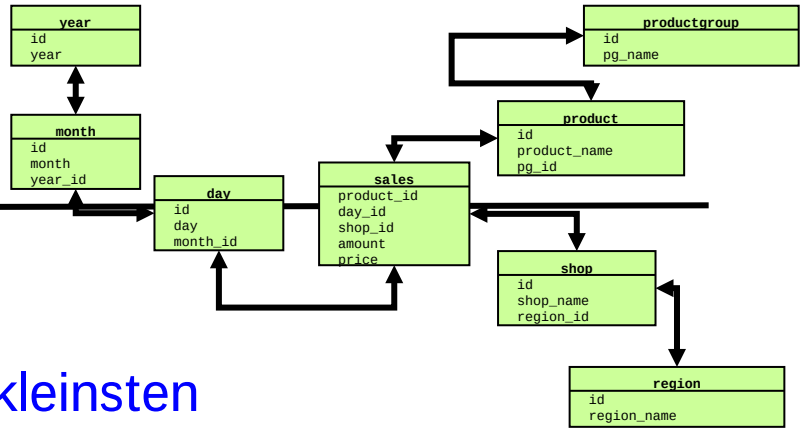
Variante 1 – Voll Normalisiert



Snowflake Schema

Quelle: www.thebridgesummit.co

Snowflake Schema



- Faktentabelle
  - Measures plus Fremdschlüssel der kleinsten Klassifikationsstufe jeder Dimension
- Dimensionstabellen
  - Eine Tabelle pro Klassifikationsstufe
  - Attribute: Knotenattribute (ignorieren wir) und ID
  - Ein Tupel pro Klassifikationsknoten
- Eigenschaften
  - Voll normalisiert
  - Speicherplatz:

$$\left(\left(d + f\right) * m + d * n\right) * b$$

Ein Tupel pro Klassifikationsknoten jeder Dimension

Fremdschlüssel je Dimension

Measures

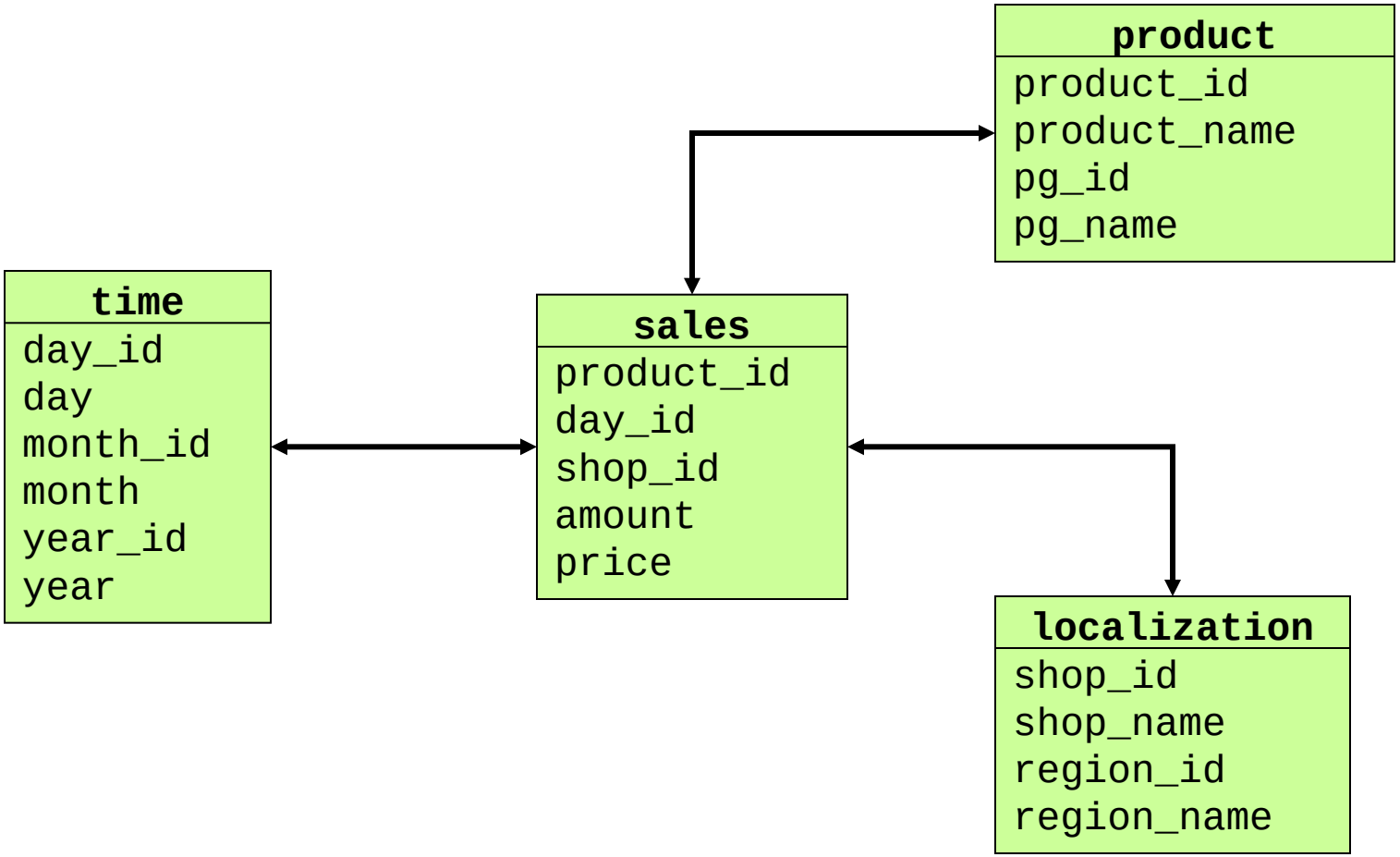
Snowflake Schema: Query

Summe aller Verkäufe der Produktgruppe „Wasser“ nach Shop und Jahr

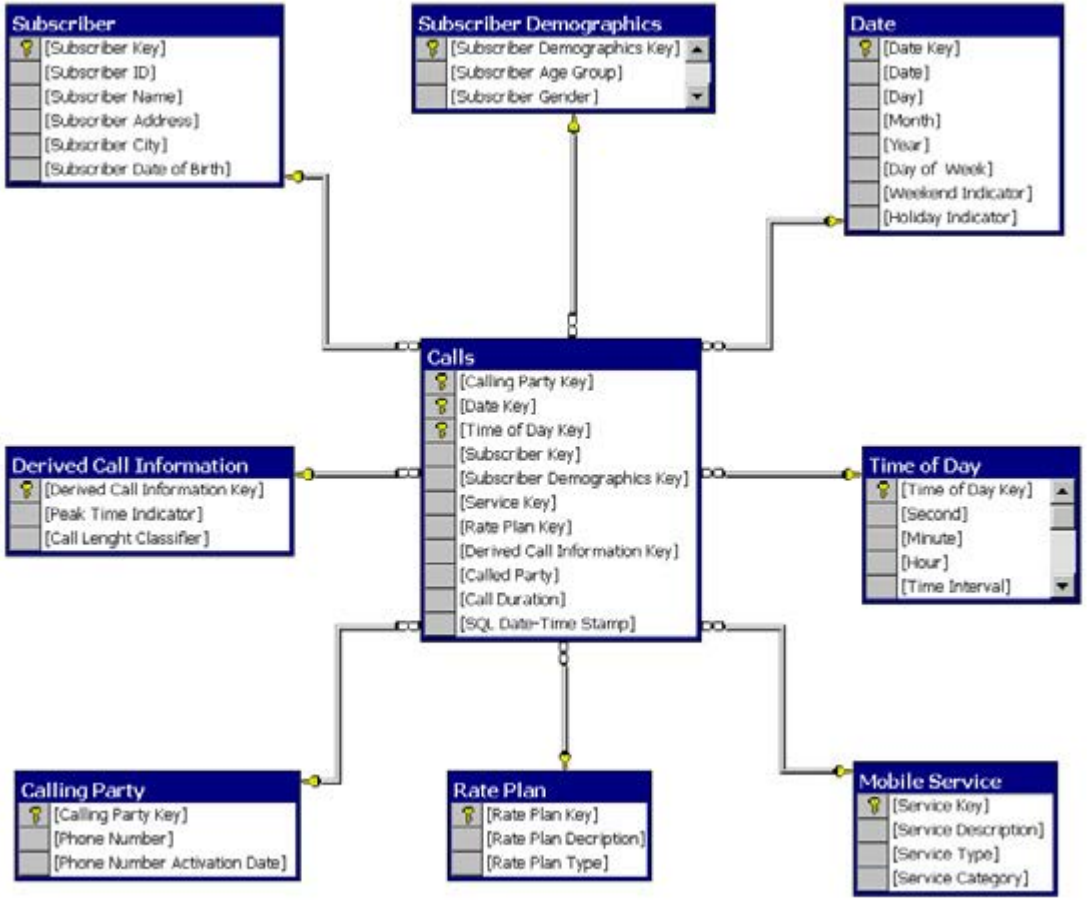
```
SELECT   X.shop_name, Y.year, sum(amount*price)
FROM     sales S, ... PG, P, D, M, Y, shop X
WHERE    PG.name=„Wasser“ AND
         PG.id = P.pg_id AND
         P.id = S.product_id AND
         D.id = S.day_id AND
         M.id = D.month_id AND
         Y.id = M.year_id AND
         X.id = S.shop_id
group by X.shop_name, Y.year
```

- 6 Joins
  - 4 Joins, wenn weder shop_name noch year notwendig
- Anzahl Joins steigt in jeder Dimension linear mit Länge der Aggregationspfade in der Anfrage

Variante 2: Denormalisiert



Star Schema



Quelle: Camilovic, „A Call Detail Records Data Mart: Data Modelling and OLAP Analysis", Comput. Sci. Inf. Syst, 2009

Star Schema

- Faktentabelle
  - Measures plus Fremdschlüssel der kleinsten Klassifikationsstufe jeder Dimension
- Dimensionstabellen
  - Eine Tabelle pro Dimension
  - Attribute: ID und Knotenattribute aller Klassifikationsstufen
  - Ein Tupel pro Klassifikationsknoten mit Level 0
- Eigenschaften
  - Denormalisierte Dimensionen
  - Speicherplatz

Ein Tupel pro Klassifikationsknoten Level 0

$$\left(\left(d + f\right) * m + d * 3^{k} * k\right) * b$$

Ein Attribut pro Klassifikationsstufe

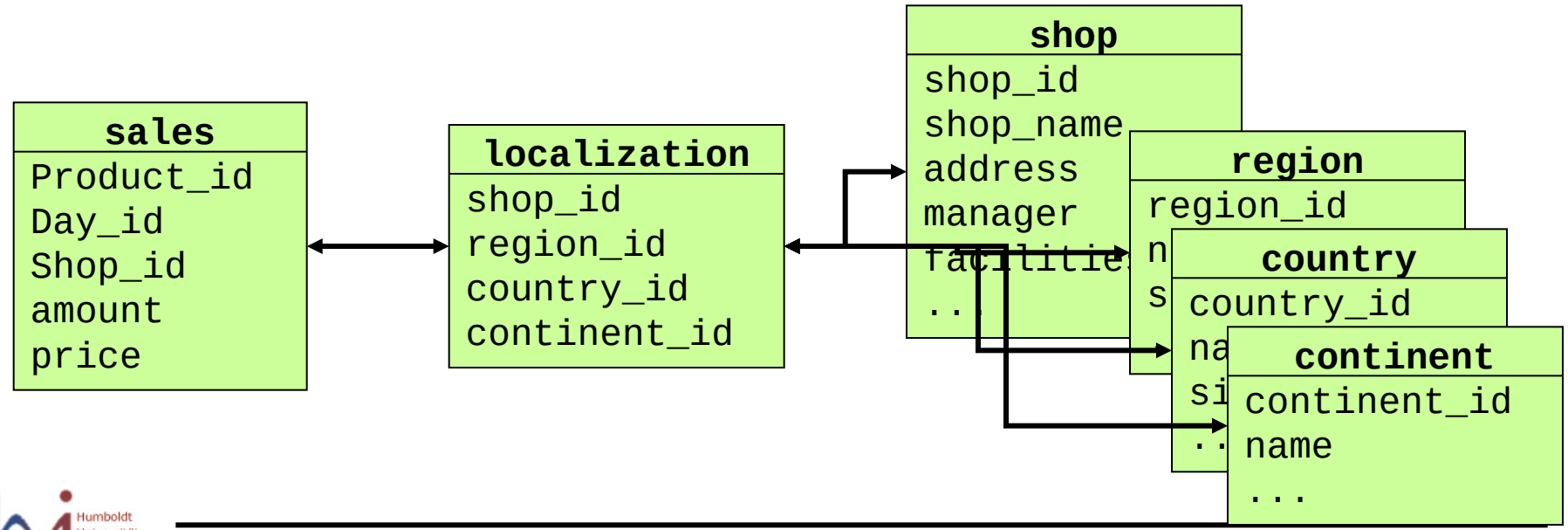
Star Schema: Query

```
SELECT    L.shop_name, T.year, sum(amount*price)
FROM      sales S, product P, time T, localization L
WHERE     P.pg_name=„Wasser“ AND
          P.product_id = S.product_id AND
          T.day_id = S.day_id AND
          L.shop_id = S.shop_id
group by  L.shop_name, T.year
```

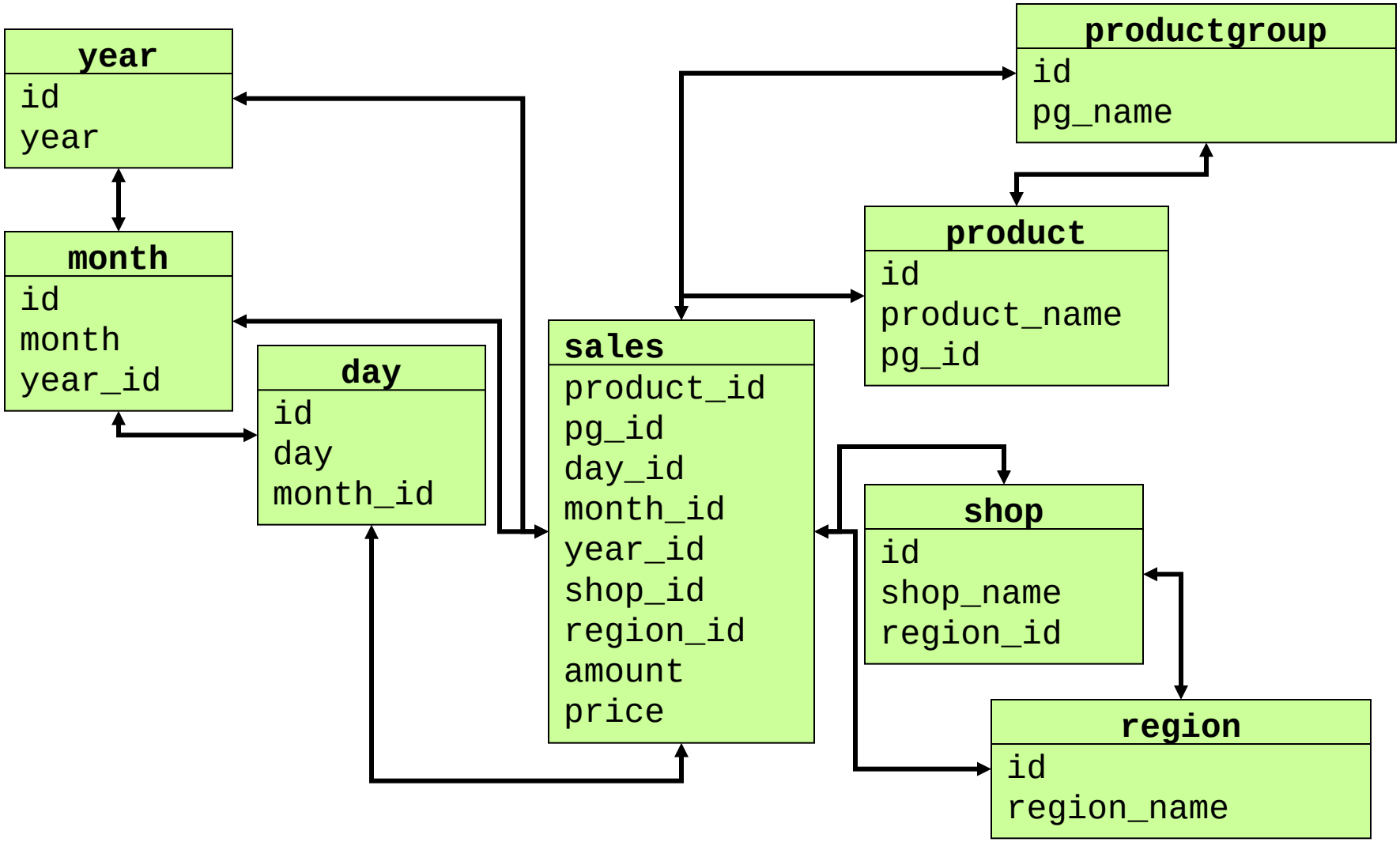
- 3 Joins
  - Keine Reduktion wenn Jahrname nicht notwendig
  - 2 Joins, wenn Shopname nicht notwendig
- Anzahl Joins
  - Unabhängig von Länge der Aggregationspfade in der Anfrage
  - Steigt linear mit der Anzahl Dimensionen in der Anfrage

Variante: Knotenattribute im Star Schema

- Eigenen Tabellen pro Klassifikationsstufe
  - Dimensionstabelle repräsentiert Hierarchie und enthält nur Keys
  - Normalisiert, speicherplatzeffizient
- Dimensionstabelle
  - Keys mehrfach vorhanden
  - Zusätzlichen Joins für Knotenattribute notwendig



Variante 3: Fullfact



Fullfact

- Faktentabelle
  - Measures plus Fremdschlüssel jeder Klassifikationsstufe
- Dimensionstabellen
  - Eine Tabelle pro Klassifikationsstufe
  - Attribute: Knotenattribute (ignorieren wir) und ID
  - Ein Tuple pro Klassifikationsknoten
- Eigenschaften
  - Normalisierte Dimensionen, denormalisierte Fakten
  - Speicherverbrauch

$$\left(\left(d * k + f\right) * m + d * n\right) * b$$

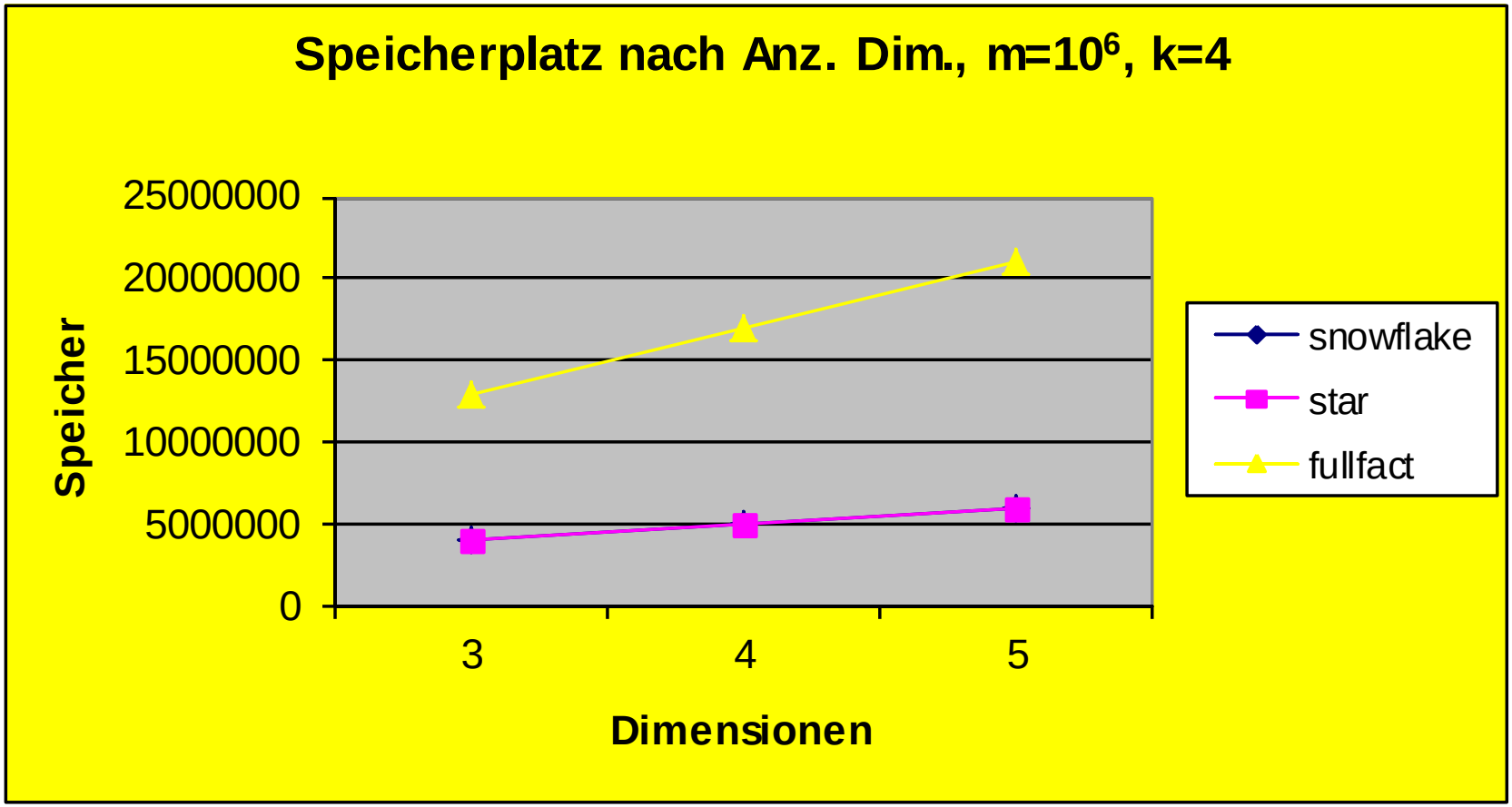
Ein FK pro Klassifikationsstufe

Fullfact Query

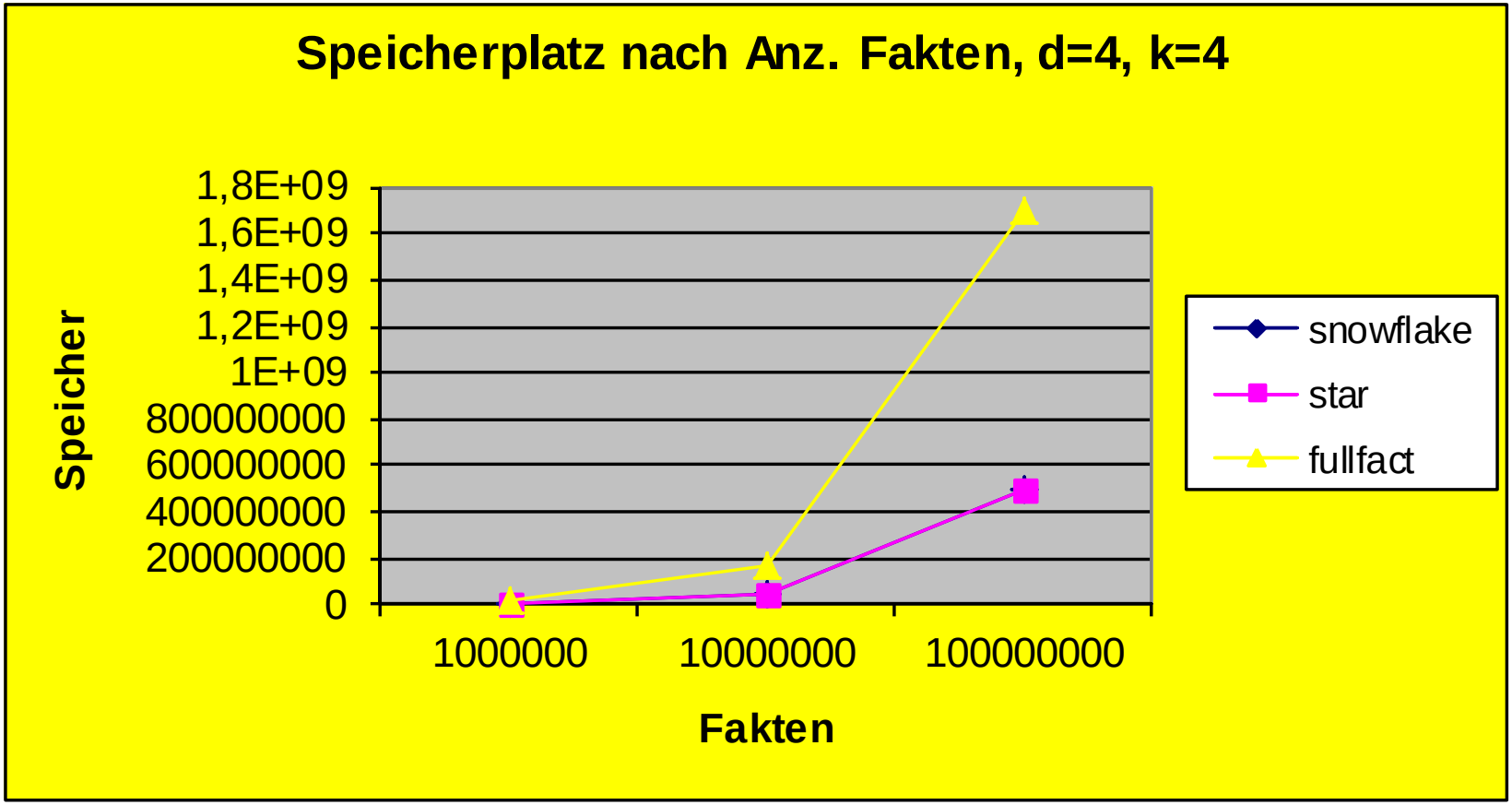
```
SELECT   X.shop_name, T.year, sum(amount*price)
FROM     sales S, productgroup PG, year Y, shop X
WHERE    PG.pg_name=„Wasser“ AND
         PG.id = S.pg_id AND
         Y.id = S.year_id AND
         X.id = S.shop_id
group by X.shop_name, Y.year
```

- 3 Joins (1 Join, wenn Shopname/Jahrname unnötig)
- Anzahl Joins
  - Unabhängig von Länge der Aggregationspfade in der Anfrage
  - Steigt linear mit der Anzahl Klassifikationsstufen in der Anfrage

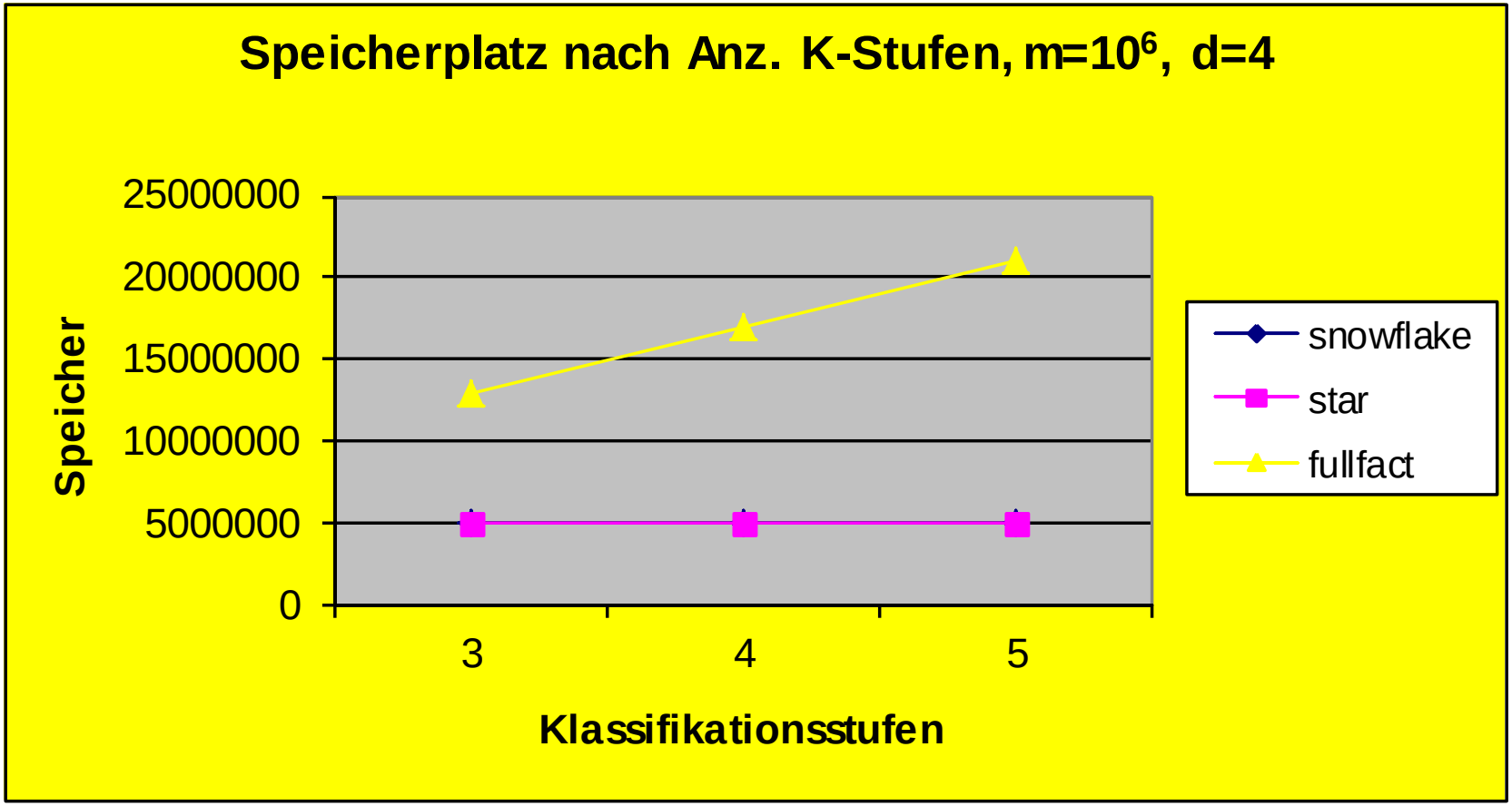
Speicherverbrauch 1



Speicherverbrauch 2



Speicherverbrauch 3



Fazit – Speicher und Query

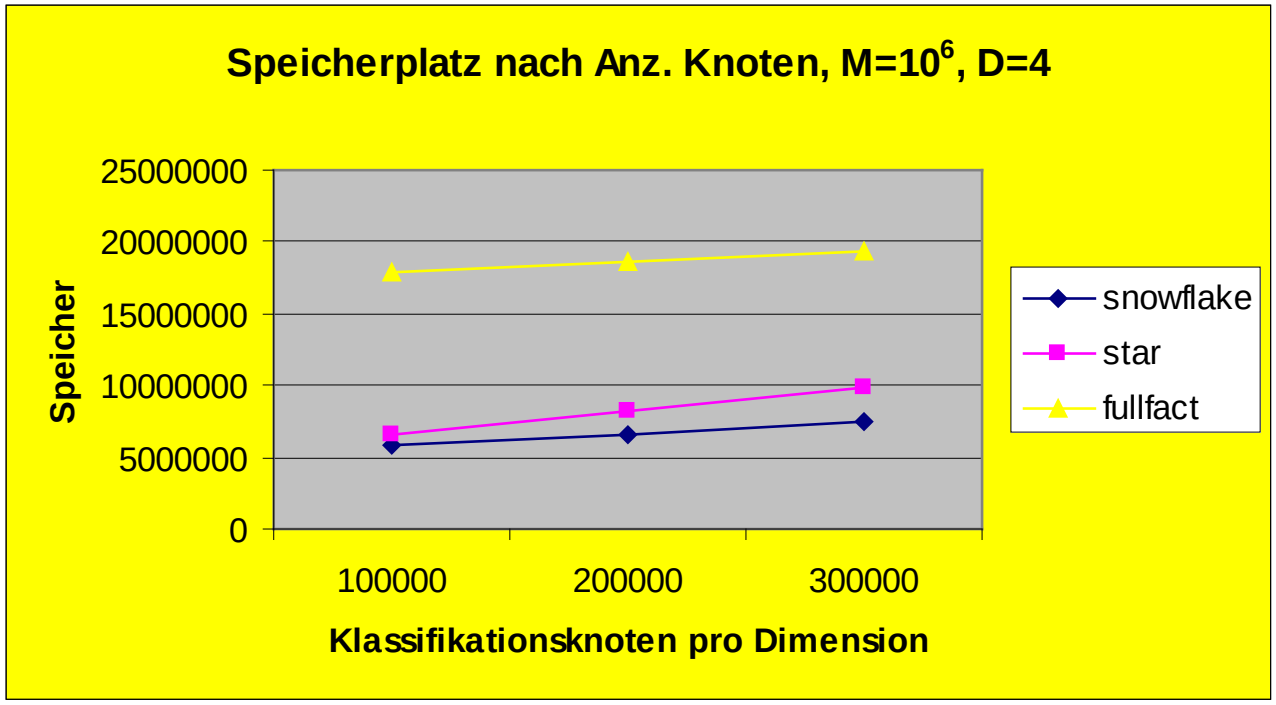
- Speicherverbrauch Snowflake / Star praktisch identisch
  - Bedarf für Dimensionen vernachlässigbar
- Fullfact mit deutlich höherem Speicherverbrauch
  - Dafür minimale Anzahl Joins
- Star braucht idR weniger Joins als Snowflake
- Laufzeitverhalten hängt von vielen Faktoren ab
  - Bereichs- oder Punktanfrage, Selektivität
  - Indexierung
  - Gruppierung und Aggregation
  - ...
  - Aber: Joins sind tendenziell immer teuer

Inhalt dieser Vorlesung

- Relationales OLAP (ROLAP)
  - Snowflake-, Star- und Fullfactschema
  - Speicherplatz und Queries
  - Schemavarianten
- Evolution in Dimensionen
- Beispiel: ROLAP in Oracle
- Multidimensionales OLAP (MOLAP)

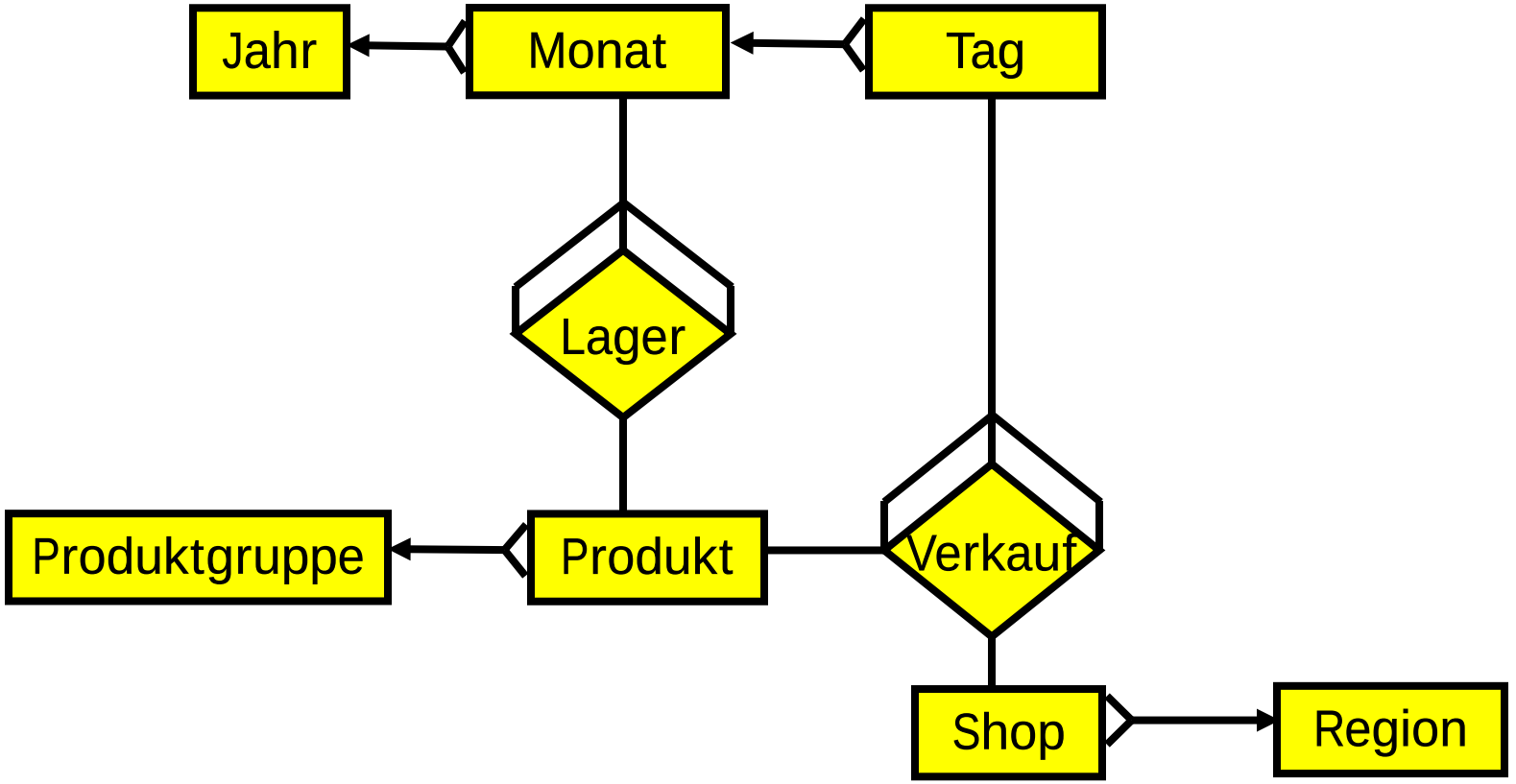
Entartete DWH

- Feingranulare Dimensionen
  - Anzahl Klassifikationsknoten nähert sich Anzahl Fakten
  - Bsp.: Experimentelle Daten (Knoten = Gene, Werte = Ergebnisse von Tests)



Galaxy Schema

Mehrere Faktentabellen



Fact Constellation Schema

- Fact Constellation Schema = Star Schema mit eigenen Summentabellen für prä-aggregierter Daten
- Alternative: Einbeziehung als “spezielle Fakten“ in Faktentabelle

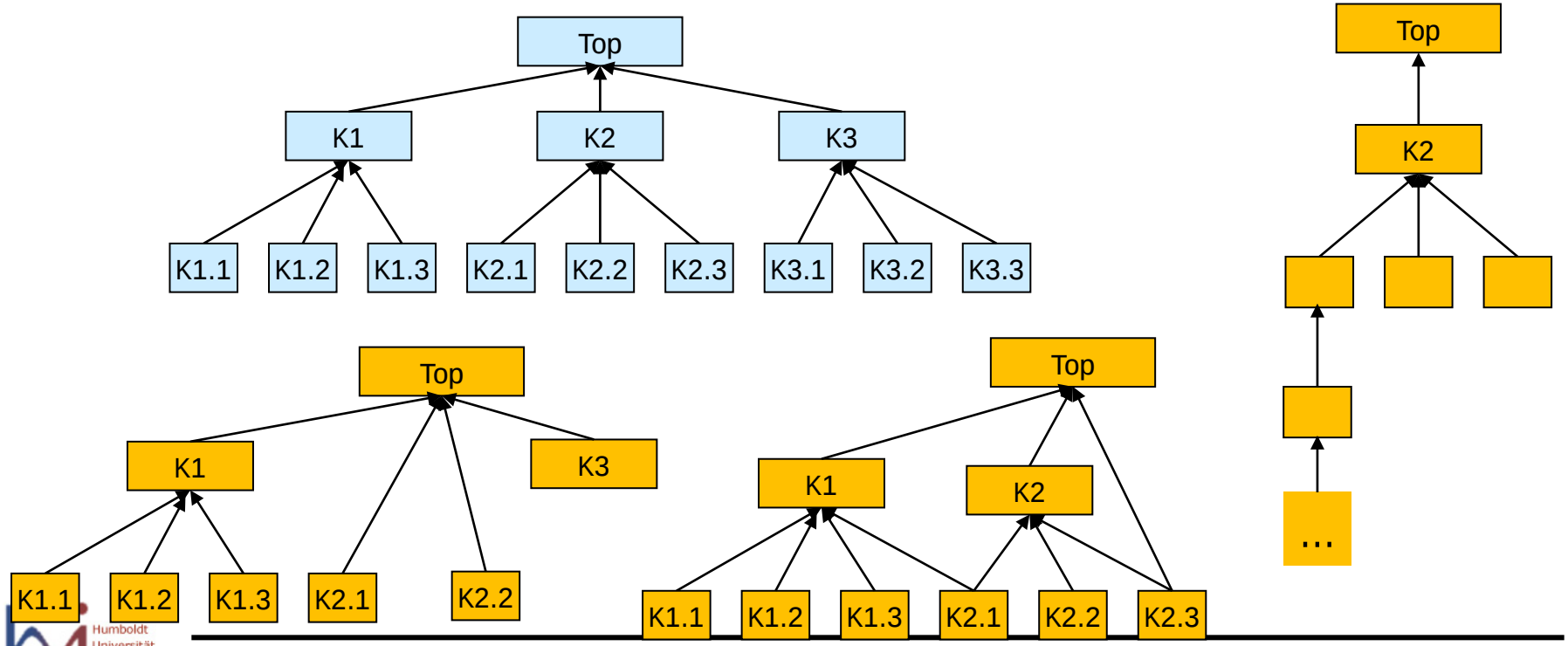
Summe über alle Tage für P=2, S=130

Summe über alle Tage und Produkte für S=130

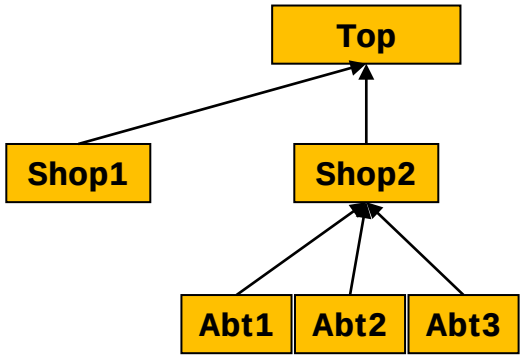
Verkauf	Day	Product	shop
3.99	1	2	130
14.80	1	2	130
12.98	1	4	130
16.02	1	4	130
18.79	NULL	2	130
29.00	NULL	4	130
47.79	NULL	NULL	130

Nicht-Standard Hierarchien

- Bisherige Annahmen: Jede Hierarchie
  - ist balanciert: Blätter haben alle den gleichen Abstand zu Root
  - hat eine feste maximale Tiefe
  - ist eine Hierarchie: Jeder Knoten hat nur einen Vorgänger



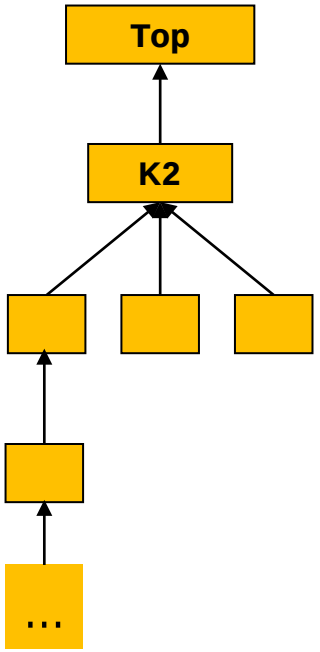
Unbalancierte Hierarchien



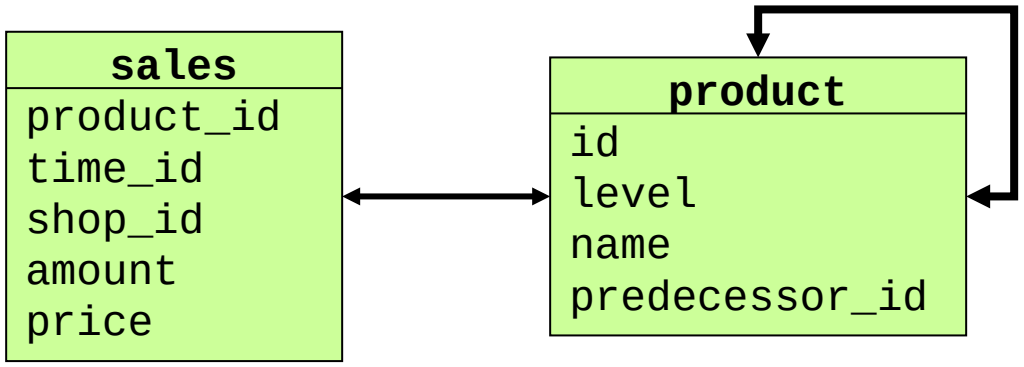
- Beispiel: Teilung nur von großen Shops in Abteilungen
- Im Snowflake Schema
  - Fakten mit „Fremdschlüsseln“ aus verschiedenen Tabellen
    - Fakten adressieren unterschiedliche Ebene 0 Stufen
  - DB kann das über PK-FK Constraints nicht prüfen (Trigger)
- Eine Lösung: Hierarchie mit Dummy-Werten auffüllen
  - Jeder Shop bekommt eine NULL-Abteilung
  - Müssen bei Ausgabe unterdrückt werden
  - Bei (generierten) hierarchischen Aggregationen entfernen
- Im Starschema: Auffüllen mit NULLs

Unbeschränkt tiefe Hierarchien

- Unbeschränkt lange Klassenhierarchien
  - Stücklisten (Teil von ...)
  - Stark variierende, länderspezifische Organisationsstrukturen
  - Auch „parent-child hierarchies“ genannt
- In Star noch Snowflake nicht möglich
  - Snowflake: Jede Stufe eine Tabelle – aber Zahl der Stufen steht nicht fest
  - Star: Jede Stufe mind. ein Attribut – aber Zahl der Stufen steht nicht fest
- Immer: Es ist unklar, wie viele Joins man braucht, um ein komplettes Roll-Up zu berechnen



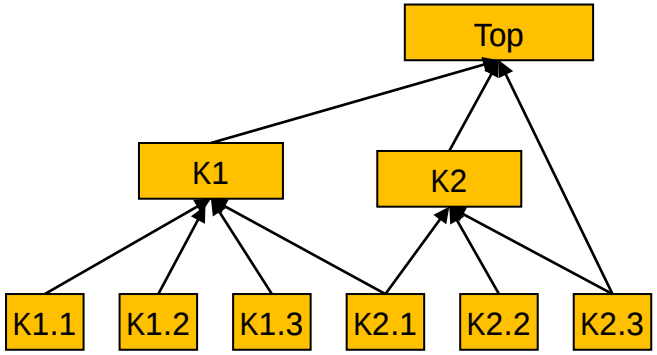
Modellierung über Selbst-Referenzialität

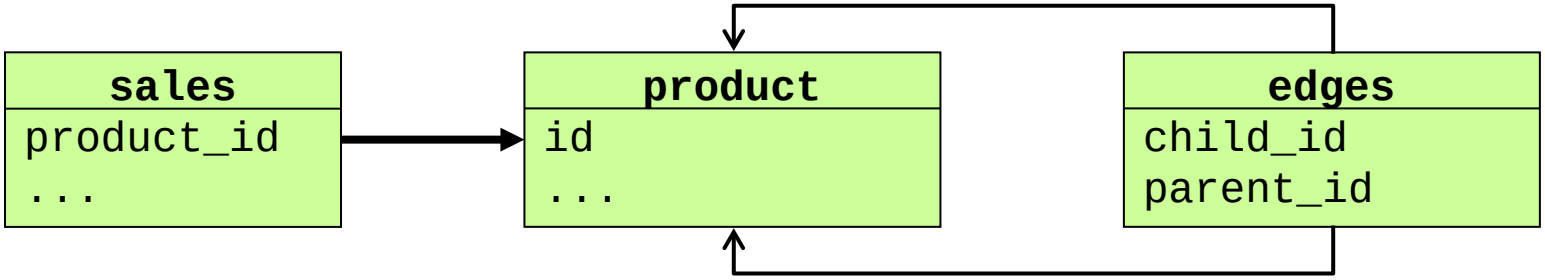


- Ermöglicht beliebige Hierarchien
  - Umfasst auch unbalancierte Hierarchien
- Probleme
  - Rekursives SQL oder Traversierung über Host-Language nötig
  - Keine individuellen Attribute pro Stufe
  - Keine Unterstützung durch SQL-OLAP
  - Eingeschränkte Prüfbarkeit der referenziellen Integrität
    - **`predecessor_id`** kann auf IDs in allen Stufen zeigen

Nicht-hierarchische Hierarchien

- Knoten können mehrere Vorgänger (auf verschiedenen Stufen) haben
- Selbst-Referenzialität reicht nicht mehr
- Graphbasierte Modellierung
  - Gleiche Nachteile wie Selbst-Referenzialität
  - Zuordnung von Werten bei Gruppierung nicht klar





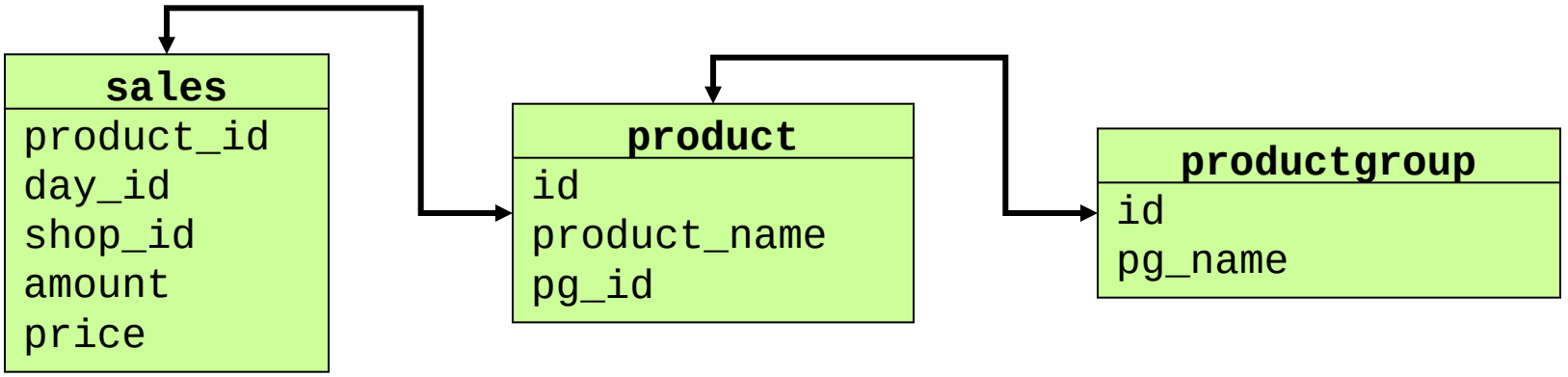
Inhalt dieser Vorlesung

- Relationales OLAP (ROLAP)
- Beispiel: ROLAP in Oracle
- Evolution in Dimensionen
- Multidimensionales OLAP (MOLAP)

MDDM Konstrukte in Oracle

- Oracle Sprachelemente
  - Dimensionen
  - Klassifikationsstufen
  - Klassifikationspfade
- Definition mit eigenen DDL Befehlen
- Setzen auf existierendes relationales Schema auf
  - Star oder Snowflake

Beispiel: Snowflake Schema



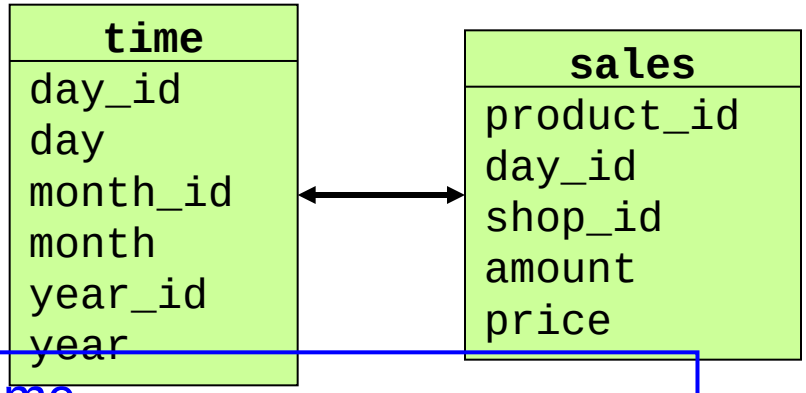
Dimension

Klassifikations-stufen

Klassifikations-pfade

```
CREATE DIMENSION s_pg
  LEVEL product IS (product.id)
  LEVEL productgroup IS (productgroup.id)
HIERARCHY pg_rollup (
  product CHILD OF productgroup
  JOIN KEY (pg_id)
    REFERENCES productgroup
);
```

Beispiel: Star Schema



1. Logischer Stufenname
2. Relationales Attribut, dass die Stufe identifiziert

Relationale Repräsentation der Stufenschritte

```
CREATE DIMENSION s_time
  LEVEL day IS (time.day_id)
  LEVEL month IS (time.month_id)
  LEVEL year IS (time.year_id)
HIERARCHY pg_rollup (
  day CHILD OF
  month CHILD OF
  year)
ATTRIBUTE day_id DETERMINES (day)
ATTRIBUTE month_id DETERMINES (month)
ATTRIBUTE year_id DETERMINES (year)
```

Multiple Pfade

Dimension

Klassifikationsstufen

Klassifikationspfad 1

Klassifikationspfad 2

```
CREATE DIMENSION times_dim
  LEVEL day IS TIMES.TIME_ID
  LEVEL month IS TIMES.CALENDAR_MONTH_DESC
  LEVEL quarter IS TIMES.CALENDAR_QUARTER_DESC
  LEVEL year IS TIMES.CALENDAR_YEAR
  LEVEL fis_week IS TIMES.WEEK_ENDING_DAY
  LEVEL fis_month IS TIMES.FISCAL_MONTH_DESC
  LEVEL fis_quarter IS TIMES.FISCAL_QUARTER_DESC
  LEVEL fis_year IS TIMES.FISCAL_YEAR
HIERARCHY cal_rollup (
  day CHILD OF
  month CHILD OF
  quarter CHILD OF
  year )
HIERARCHY fis_rollup (
  day CHILD OF
  fis_week CHILD OF
  fis_month CHILD OF
  fis_quarter CHILD OF
  fis_year )
<attribute determination clauses>...
```

Eigenschaften

- Vollständige und überlappungsfreie Hierarchien
  - *„The child_key_columns must be non-null and the parent key must be unique and non-null. You need not define constraints to enforce these conditions, but queries may return incorrect results if these conditions are not true.“*
  - *„.. the joins between the dimension tables can guarantee that each child-side row joins with one and only one parent-side row. In the case of denormalized dimensions, determine whether the child-side columns uniquely determine the parent-side (or attribute) columns. If you use constraints to represent these relationships, they can be enabled with the NOVALIDATE and RELY clauses if the relationships represented by the constraints are guaranteed by other means.„*
- Verwendung und Anzeige über PL/SQL Packages
  - **`DEMO_DIM, DBMS_OLAP,…`**
- Sinn
  - Für Query Optimierung (z.B. „Materialisierte Sichten“)
  - Für Client-Software / Visualisierung / Navigation

Inhalt dieser Vorlesung

- Relationales OLAP (ROLAP)
  - Snowflake-, Star- und Fullfactschema
  - Speicherplatz und Queries
  - Schemavarianten
- Beispiel: ROLAP in Oracle
- Evolution in Dimensionen
  - Änderung in den Klassifikationsknoten
  - Änderungen in der Klassifikationshierarchie
- Multidimensionales OLAP (MOLAP)

Änderungen in den Dimensionen

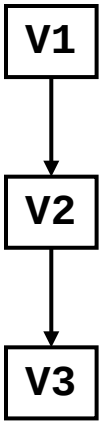
- DWH speichern Daten über lange Zeiträume
- Fakten „ändern“ sich nicht – werden nur hinzugefügt
- In langen Zeiträumen passieren Änderungen in den Dimensionen
  - In den Werten (Namen / Eigenschaften von Produkten, Status von Kunden etc.)
  - In den Dimensionsstrukturen (neue Organisationsebenen, Zusammenfassung von Produktgruppen, Auslistung, ...)
- „Alte“ Fakten müssen im „alten“ Bezugsrahmen bleiben
- Sehr reales und eher selten diskutiertes Problem

„Slowly Changing“ Dimensions

- Updates bei den Klassifikationsknoten einer Hierarchie
  - In den Werten: Keine Korrekturen (Buch hätte schon immer „gebunden“ sein sollen), sondern Änderungen (Buch wurde bisher nur gebunden verkauft, jetzt nur noch broschiert)
  - In den Strukturen: Deletes und Inserts – Knoten existieren nur in Zeiträumen (mit unterschiedlichen Werten)
- Alte Knoten werden von alten Fakten adressiert, neue Knoten von neuen Fakten
- Einfaches SQL-Update führt zu Verfälschungen
  - Verkäufe eines gebundenen Buchs vor Stichtag werden plötzlich zu Verkäufen eines broschierten Buchs
- Lösung: Klassifikationsknoten versionieren

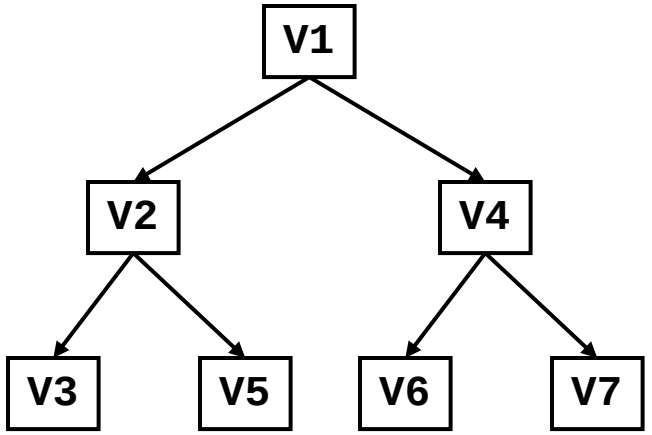
Versionierungsmodelle

Linear



- Zu jeden Zeitpunkt existiert von jedem Tupel nur eine Version
- Standardmodell

Hierarchisch



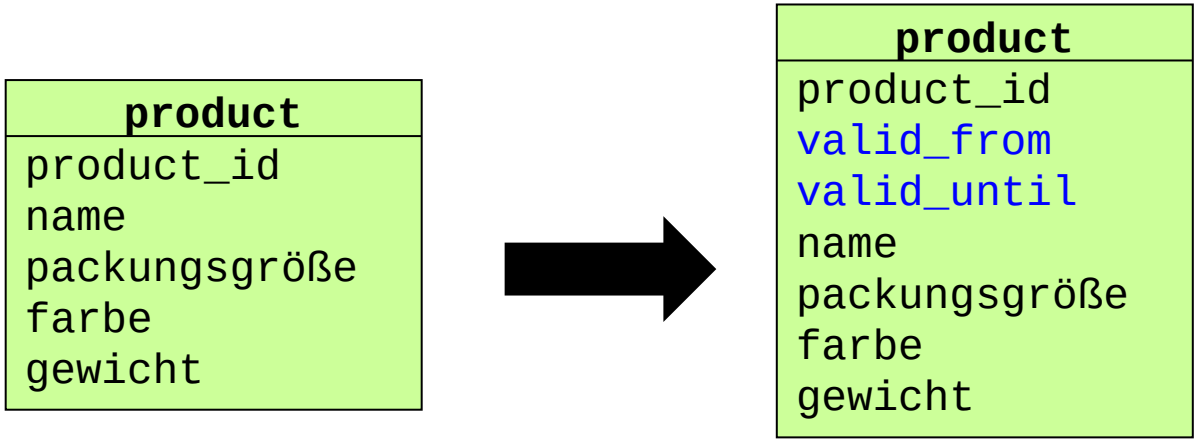
- Zeitliche Reihenfolge halbgeordnet
  - z.B. Software-Versionsverwaltungen
- Deutlich komplexer

Lineare Versionen in RDBMS

- RDBMS muss Versionen von Tupeln verwalten
  - Änderung in einem Attributwert – eine neue Version
  - Gleichzeitige Änderungen in mehreren Attributwerten – eine neue Version
  - Schemaänderungen werden hier nicht betrachtet
  - Tupel werden referenziert – auch beachten (FK / PK)
- Anforderungen
  - Schneller Zugriff auf aktuelle Version
  - Möglicher Zugriff auf Version zu beliebigem Zeitpunkt d_0
  - Abfrage eines konsistenten Gesamtzustands zu beliebigem Zeitpunkt (über alle Dimensionen und Fakten)
- Zwei prominente Varianten
  - Single-Table
  - Schattentabellen

Variante 1: Single-Table

- Erweiterung jeder Tabelle T um die folgenden Attribute
  - valid_from
  - valid_until
  - Primärschlüssel: id → (id, valid_from)



Interpretation

- Aktuelle Version: valid_until=-1
  - Nur eine Version darf valid_until=-1 haben
  - Durch Trigger sicherstellen
- [valid_from, valid_until] ist das Zeitintervall, in dem ein Tupel gültig ist (war)
  - Intervalle dürfen sich nicht überschneiden
  - Durch Trigger sicherstellen
- Lücken zwischen Intervallen sind möglich
  - Objekte haben existiert, wurden gelöscht, wieder eingefügt ...
  - Vorsicht vor Lücken durch Zeitmessungsgenauigkeit
  - Lösung: stunden-/taggenaue Versionen

DELETE

- DELETE Objekt k (identifiziert durch ID) aus T
  - Gibt es k in T mit valid_until=-1?
  - Nein: Nichts tun
    - Objekt gibt es gerade nicht oder gab es noch nie
  - Ja: Lebensende der aktuellen Version setzen
    - UPDATE T SET valid_until=SYSDATE
      WHERE id=k AND valid_until=-1;
  - FKs in abhängigen Tabellen bleiben gültig
  - Aber: Kein Einfügen mehr mit FK auf gelöschtes Tupel gestatten
    - Trigger

INSERT

- INSERT Objekt k (identifiziert durch ID) in T
  - Gibt es k in T mit valid_until=-1?
  - Nein: Objekt gibt es (jetzt) nicht, einfügen
    - INSERT INTO T
      VALUES (id=k, valid_from=SYSDATE, valid_until=-1, ...)
  - Ja: Fehlermeldung

UPDATE

- UPDATE Objekt k (identifiziert durch ID) in T
  - Gibt es k in T mit valid_until=-1?
  - Nein: Nichts tun
    - Objekt gibt es gerade nicht
  - Ja: Aktuelle Version abschließen, neue Version einfügen
    - UPDATE T SET valid_until=SYSDATE
      WHERE id=k AND valid_until=-1;
    - INSERT INTO T
      VALUES (id=k, valid_from=SYSDATE, valid_until=-1, ...)
  - FKs in abhängigen Tabellen bleiben gültig
  - Aber: Einfügen nur noch mit FK auf aktuelles Tupel gestatten
    - Trigger

Verschiedene SELECTs

- SELECT aktuelle Version von k

```
SELECT * FROM T
WHERE   valid_until=-1 AND id=k;
```

- SELECT aktueller Zustand (alle gültigen Objekte) von T

```
SELECT * FROM T
WHERE   valid_until=-1;
```

- SELECT alle Tupel gültig zum Zeitpunkt d

```
SELECT * FROM T
WHERE   valid_from <= d AND (valid_until=-1 OR valid_until=>d);
```

Bewertung

- INSERT / DELETE / UPDATE erfordern Trigger
  - Auch auf abhängigen Tabellen
- valid_until,valid_from sollte in praktisch alle Indexe aufgenommen werden
  - In dieser Reihenfolge: Schnelle Zugriff auf aktuelle und vergangene Versionen von Objekten
- Index-Unterstützung (alle aktuellen) schwierig
  - „valid_until=-1“ hat schlechte Selektivität -> Full Table Scan
  - Nur bei sehr vielen Änderungen steigt Selektivität und Indexierung lohnt sich
- Verlangsamung der Arbeit mit aktuellen Daten durch wachsende Tabelle

Variante 2: Schattentabellen

- Für jede zu versionierende Tabelle T wird zusätzlich eine Schattentabelle T^S angelegt
- Tabelle T speichert nur aktuell gültige Tupel, Tabelle T^s nur alte, nicht mehr gültige Versionen
- Struktur der Tabelle T^s für eine Tabelle T
  - Alle Attribute aus T
  - Zusätzlich valid_from und valid_until
  - Primärschlüssel: id $\rightarrow$ (id, valid_from)
- Außerdem: T um Attribut valid_from erweitern

INSERT

- INSERT Objekt k in T
  - k in T vorhanden ?
    - Nein: Einfügen in aktuelle Tabelle
      - INSERT INTO T
        VALUES (id=k, valid_from=SYSDATE, ...)
    - Ja: Fehlermeldung

Variante 2: DELETE

- DELETE Objekt k aus T
  - k in T vorhanden ?
  - Nein: Nichts tun
  - Ja: Tupel k aus T nach T^S verschieben
    - INSERT INTO T^S
      VALUES (id=k.ID, valid_from=k.valid_from,
      valid_until=SYSDATE, ...)
    - DELETE FROM T WHERE id=k
    - FKs in abhängigen Tabellen müssten auf Schattentabelle umgebogen werden
      - Also kann FK kein klassischer FK sein (verschiedene Zieltabellen)
      - Trigger

UPDATE

- UPDATE Objekt k in T
  - k in T vorhanden?
  - Nein: Nichts tun
  - Ja: Verschiedenes
    - Altes Tupel k verschieben
    - INSERT INTO T^S
      VALUES (id=k.ID, valid_from=k.valid_from,
      valid_until=SYSDATE, ...)
    - Werte und Zeitstempel in T updaten
    - UPDATE T SET valid_from=SYSDATE, ... WHERE id=k
    - FKs in abhängigen Tabellen müssten auf Schattentabelle umgebogen werden
      - Trigger

SELECT

- SELECT aktuellste Version von k

```
SELECT * FROM T
WHERE   id=k;
```

- SELECT aktueller Zustand von T

```
SELECT * FROM T;
```

- SELECT alle Tupel gültig zum Zeitpunkt d

```
SELECT * FROM T
WHERE valid_from<=d
UNION
SELECT * FROM T^S
WHERE valid_from<=d AND valid_until >=d;
```

Join mit Faktentabelle

- Faktentabelle erhält „t_ID“ und „t_valid_from“ als Referenz
  - Kein überwachter Sekundärschlüssel
  - Das referenzierte Tupel kann in T oder in T^S stehen
- Finde alle Fakten mit zugehörigen Werten aus T zu beliebigem Zeitpunkt

```
SELECT * FROM F, T
WHERE F.t_id = t.id AND F.t_valid_from = T.valid_from
UNION
SELECT * FROM F, TS
WHERE F.t_id = TS.id AND F.t_valid_from = TS.valid_from
```

Bewertung

- INSERT / DELETE / UPDATE erfordern Trigger
  - Auch FK-Constraints in abhängigen Tabellen müssen durch Trigger ersetzt werden
- Sehr schneller Zugriff auf aktuelle Version
  - Kein Unterschied zu nicht-versioniert
- Komplexer Zugriff auf Zustand zu Zeitpunkt d
  - Zugriff auf zwei Tabellen
- Kompliziertere DELETE / UPDATE
  - Zwei Tabellen betroffen

Vergleich

- Häufige Änderungen, eher wenig Lesezugriffe - Variante 1
- Seltenere Änderungen, vor allem Zugriff auf aktuellste Version – Variante 2
- Variante 2 eher für DWH geeignet
  - Alle Zugriffe ohne Zeitbezug laufen ohne Änderungen
  - Zugriff auf andere Zeitpunkte durch Views vereinfachen
- Vorsicht vor Effekten auf physischer RDBMS-Ebene
  - Indexdegradierung, Tabellenfragmentierung etc.
- Fremdschlüsselverwaltung und Konsistenz über Tabellen und Versionen hinweg braucht zusätzliche Maßnahmen
  - Logischer Join in Variante 2, der alle Versionen ab Zeitpunkt X bis heute sucht: 4 physische Joins notwendig

Beispiel: Versionierung in Oracle

- Oracle Flashback (lineare Vers., auch „time travel“)
  - Rücksetzen der Datenbank auf Status zu Zeitpunkt X
  - Flashback-Queries (AS OF): Zugriff auf Daten zu Zeitpunkt X
  - Änderungen werden gepuffert; UNDO-Buffergröße bestimmt maximalen zeitlichen Abstand
  - DDL Operationen zerstören UNDO Informationen
  - Umsetzung tief im System, schnell, transparente Nutzung
- Workspace Manager (hierarchische Versionierung)
  - Auf Tabellenebene (funktioniert mit einer Art Schattentabelle)
  - Benutzung für WHAT-IF, oder versuchsweise große Änderungen (ohne TX) mit anschließendem MERGE
  - Div. Einschränkungen bei Constraints, Trigger, etc.
  - Umsetzung in PL/SQL Packages, eher langsam

Inhalt dieser Vorlesung

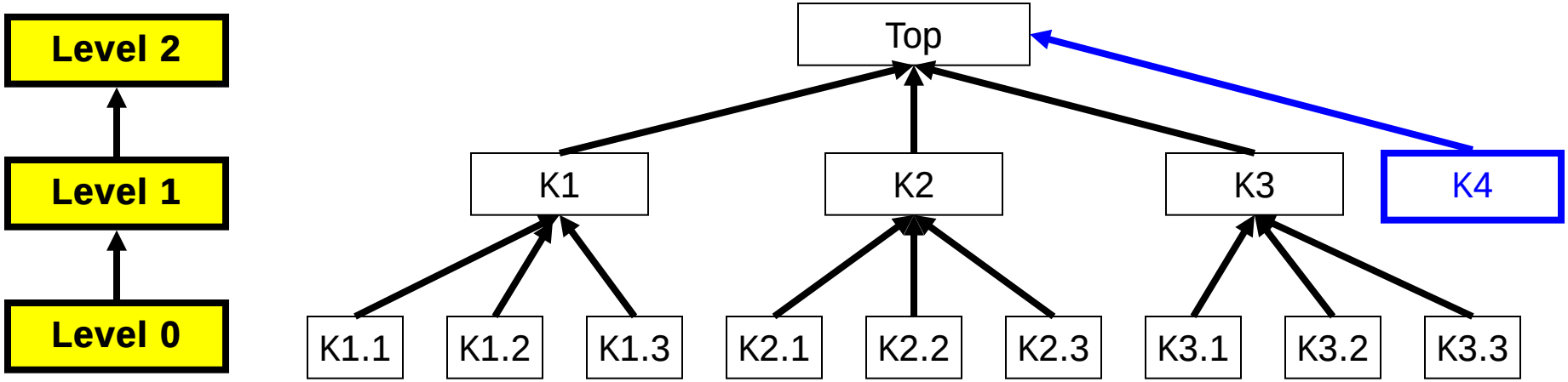
- Relationales OLAP (ROLAP)
- Beispiel: ROLAP in Oracle
- Evolution in Dimensionen
  - Änderung in den Klassifikationsknoten
  - Änderungen in der Klassifikationshierarchie
- Multidimensionales OLAP (MOLAP)

Änderungen in Klassifikationshierarchie

- INS von Klassifikationsknoten ändern die Hierarchie
  - Echte DEL gibt es praktisch nicht - Versionierung
- Das macht Anpassungen in anderen Tabellen nötig
- Aufwand für Operationen abhängig von Modellierung
- Im folgenden: Exemplarisch zwei Szenarien

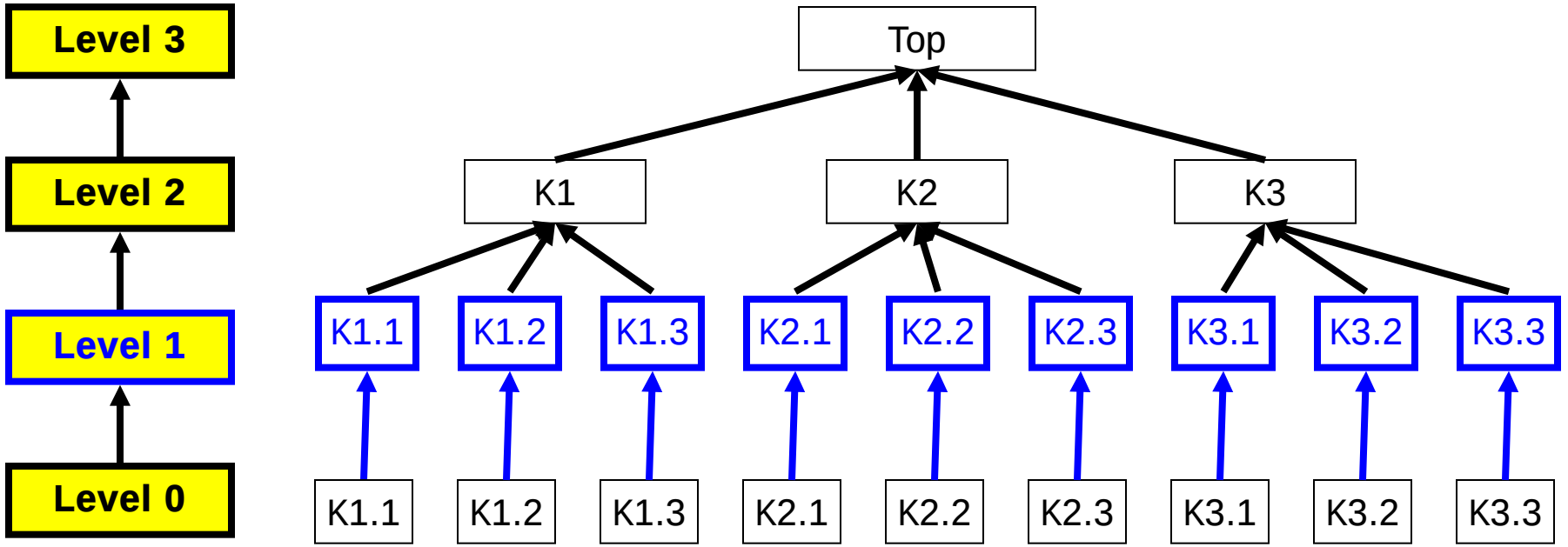
Änderung: InsN

Einfügen eines neuen Klassifikationsknoten ohne Nachkommen in einen Pfad auf Level i≠0



Änderung: InsS

Einfügen einer neuen Klassifikationsstufe in einen Pfad zwischen Level i und i-1 (i>1) mit $3^{(k+1)-i}$ Klassifikationsknoten mit Umhängen der Knoten auf Level i-1



Änderungskosten

Padding: NULL in Knoten mit Level j<i

	Snowflake	**Star**	**Fullfact**
InsN	1 Insert	1 Insert	1 Insert
InsS	1 neue Tabelle, $3^{(k+1)-i}$ Updates (Umhängung)	1 neues Attribut, 3^k Updates (Wert des Attrib.)	1 neue Tabelle, 1 neues Attribut, m Updates (in Faktentabelle)

Inhalt dieser Vorlesung

- Relationales OLAP (ROLAP)
- Beispiel: ROLAP in Oracle
- Evolution in Dimensionen
- Multidimensionales OLAP (MOLAP)

Multidimensionales OLAP

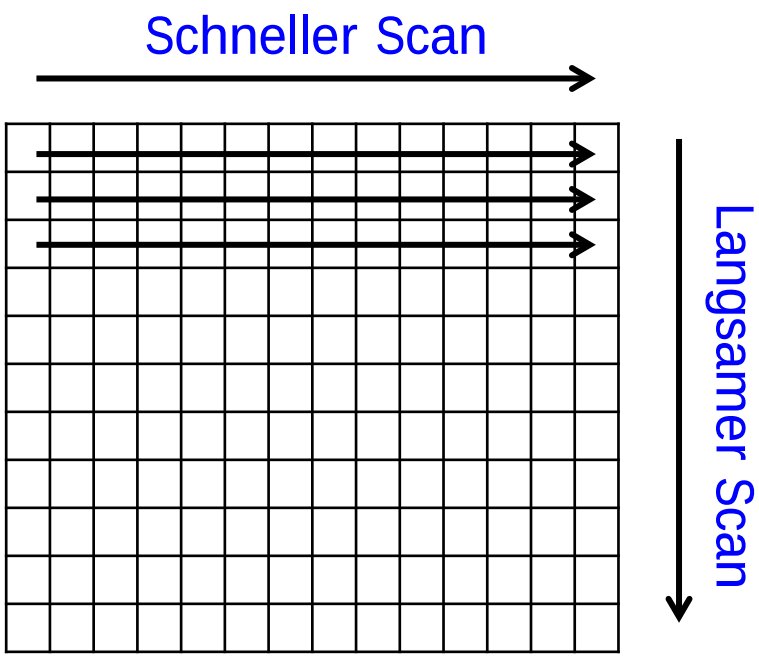
- Grundidee: Speicherung multidimensionaler Daten in multidimensionalen Arrays im Hauptspeicher
  - Ideen z.T. auch auf Sekundärspeicher übertragbar
  - Siehe VL multidimensionale Indexstrukturen
  - MOLAP in der Praxis aber immer im Main-Memory umgesetzt
    - Also müssen die Datenmengen reduziert werden
- Sehr schnell für „passende“ Zugriffsmuster
  - Zugriff auf eine Zelle, Scan entlang primärer Dimension
- „Real-time BI“ geht nur mit MOLAP im Hauptspeicher
- Im folgenden: 1 Würfel, 1 Measure, 1 Aggregatsfunktion
  - Sonst braucht man mehrere Array

Array Adressierung

- Sei $d_i = |D_i|$, $d = |D|$
- b: Speicherplatz pro Attribut (Key oder Wert)
- Arrays: Linearisierte Darstellung
  - <1,1,1>, <1,1,2>, ... , <1,1,d_1>, <1,2,1>, ... <1,d_2, d_1>, ...
- Offset der Zelle <a_1,...a_d>

$$b * \sum_{i=1}^{d} \left((a_i - 1) * \prod_{j=1}^{i-1} d_j \right)$$

- Scans sind nicht gleichschnell in verschiedenen Dimensionen
  - Wegen Caching im Hauptspeicher



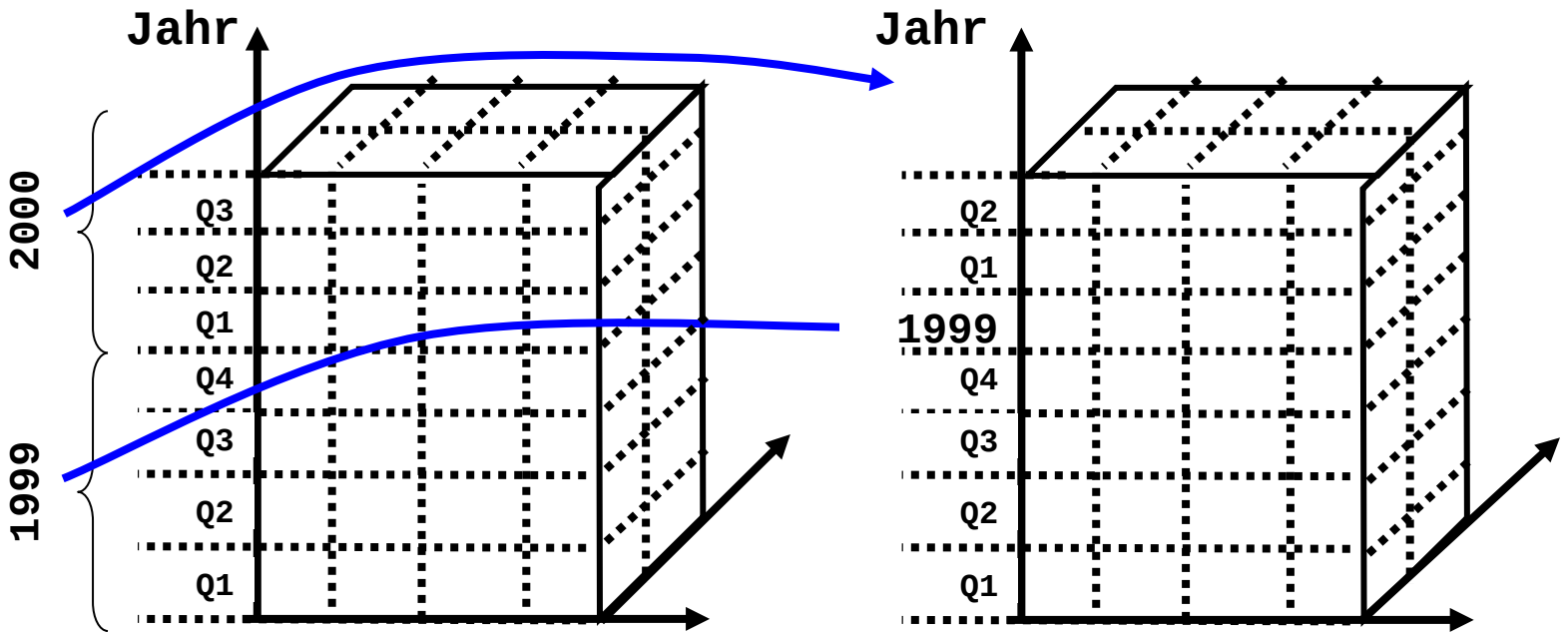
Probleme

- Umgang mit hierarchischen Dimensionen?
  - Standard: Array speichert höchste Granularität, Zugriff auf höhere Granularitäten durch on-the-fly Berechnung der Werte
  - Oder Einbettung (next slide)
- Speicherung von Fakten?
  - Müssen außerhalb des Arrays verwaltet werden
  - MOLAP beginnt oberhalb der Faktenebene
  - Aggregationsfunktion muss beim Anlegen festgelegt werden
- Wenn die Daten nicht in Hauptspeicher passen?
  - Höhere Granularität wählen oder mehr Speicher kaufen
- Langsamer Zugriff bei unpassenden Zugriffsmustern
  - Siehe multidim. Indexstrukturen (DBS-II)
- Siehe auch VL zu Hauptspeicherdatenbanken

Klassifikationshierarchien

Knoten höherer Stufen können als artifizielle Knoten in die untersten Stufe eingebettet werden

(das sind aber sehr viele Knoten bei vielen Dimensionen / Stufen)



Speicherverbrauch

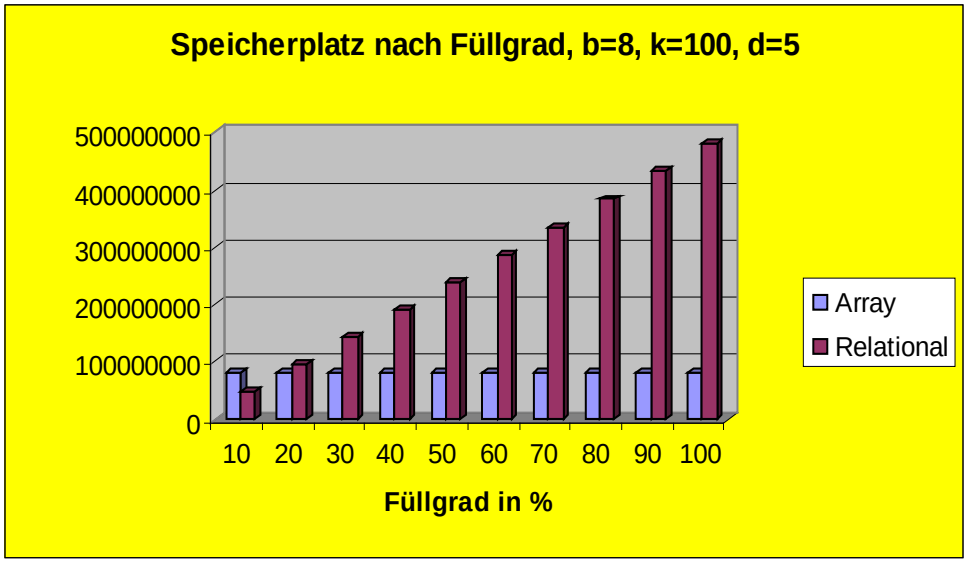
	Array	Relational (Star Schema)
Speicherung Koordinaten	Implizit (Linearisierung)	Explizit (redundant)
Leere Zellen	Belegen Platz	Belegen keinen Platz
Neue Klassifik.-knoten	Komplette Reorganisation Stark wachsender Speicher	Neue Zeile in Dim-tabelle Kaum Speicherwachstum
Speicher-verbrauch	$b * \prod_{i=1}^{d} d_i$	$b * m * (d+1)$

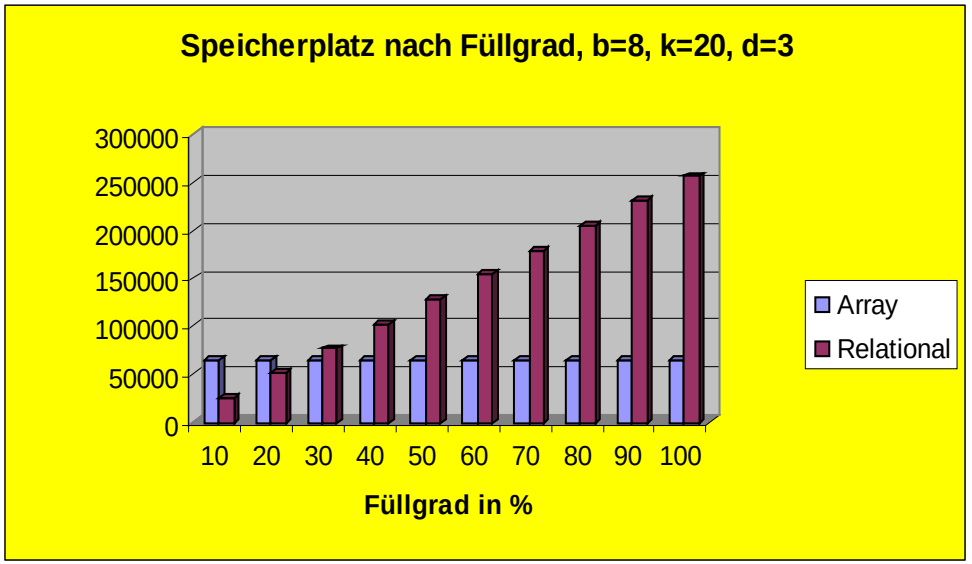
Unfair!

	Array	Relational (Star Schema)
Speicherung Koordinaten	Implizit (Linearisierung)	Explizit (redundant)
Leere Zellen	Belegen Platz	Belegen keinen Platz
Neue Klassifik.-knoten	Komplette Reorganisation Stark wachsender Speicher	Neue Zeile in Dim-tabelle Kaum Speicherwachstum
Speicher-verbrauch	$b * \prod_{i=1}^{d} d_i$	$b * m * (d+1)$
	Ohne Fakten	Mit Fakten

Vergleich Speicherplatz

- Parameter: Füllgrad, k, d
- Schon bei kleinen Füllgraden ist Array platzeffizienter
- Tatsächlicher Füllgrad oft klein, wenn viele Dims / Knoten
  - Bei weitem nicht alle Kombinationen finden sich in den Daten





Fazit MOLAP

- Viel schneller als ROLAP
  - Da nur Hauptspeicher und prä-aggregiert
- Ermöglicht schnelle, responsive User-Interfaces
- Datenmengenbegrenzung durch Hauptspeicher
- Typischer Einsatz
  - (Regelmäßige) Snapshots des (ROLAP) Cubes auf Client in MOLAP Datenbank laden
    - Dabei Vorberechnung aller Aggregate
    - Read-Only
  - Drill-Down auf untere Granularitäten durch Zugriff auf ROLAP Cubes (langsam, IO)

Selbsttest

- Erklären Sie Snowflake und Star-Schema. Was sind Vorteile / Nachteile?
- Arten von Versionierung in RDBMS? Wie kann man die realisieren?
- Wir könnte man Evolution in Klassifikationsstufen in einem Snowflake-Schema abbilden?
- Wie funktioniert der Zugriff auf eine Zelle in MOLAP? Was steht in der Zelle?
- Wie kann man unbeschränkt tiefe Hierarchien in einem RDBMS abbilden? Wie in MOLAP? Was hat das für Auswirkungen?