ISDA 05
SQL

Prof. Dr. Volker Markl

mit Folienmaterial von Prof. Dr. Felix Naumann

Fachgebiet Datenbanksysteme und Informationsmanagement
Technische Universität Berlin

http://www.dima.tu-berlin.de/

Rückblick

- Einführung
- Basisoperatoren
  - Selektion (σ)
  - Projektion (π)
  - Vereinigung ($\cup$)
  - Differenz (\ oder -)
  - Cartesisches Produkt ($\times$)
  - Umbenennung (ρ)
- Komplexe Ausdrücke
- Abgeleitete Operatoren
  - Join ($\bowtie$)
  - Schnitt ($\cap$)
  - Division (/)
- Operatoren auf Multimengen
- Erweiterte Operatoren
  - Duplikateliminierung
  - Generalisierte Projektion (Gruppierung und Aggregation)
  - Outer Joins und Semi-Join
  - Sortierung

(Kapitel 5 des Lehrbuches)

Überblick

- **Historie von SQL**
- Einfache Anfragen
- Anfragen über mehrere Relationen
- Geschachtelte Anfragen
- Operationen auf einer Relation
- DDL und DML

Kapitel 6 des Lehrbuchs

Motivation

- Verbreiteteste Datenbankanfragesprache
- Ad-hoc und einfach
- Deklarativ
  - Nicht prozedural
  - Optimierbar System optimiert Anfragen
- Very-High-Level Language
- Anfragen an relationale Algebra angelehnt
  - Hinzu kommt DML

- Achtung: Syntax kann sich von System zu System leicht unterscheiden.
- Achtung: Funktionalität kann sich von System zu System leicht unterscheiden.

SQL – Historie

- SEQUEL (1974, IBM Research Labs San Jose)
- SEQUEL2 (1976, IBM Research Labs San Jose)
  - System R
- SQL (1982, IBM)
- ANSI-SQL (SQL-86; 1986)
  - Erste genormte Version
- ISO-SQL (SQL-89); 1989;
  - drei Sprachebenen: Level 1, Level 2, + IEF
- (ANSI / ISO) SQL2 (als SQL-92 verabschiedet)
- (ANSI / ISO) SQL3 (als SQL-99 verabschiedet)
  - Objektrelationale Erweiterungen
- (ANSI / ISO) SQL:2003
- Trotz Standardisierung: teilweise Inkompatibilitäten zwischen Systemen der einzelnen Hersteller

SQL:2003

- Part 1: SQL/Framework (92 Seiten)
  - Überblick
- Part 2: SQL/Foundation (1310 Seiten)
  - Datenmodell, DDL, DML, Abfragen
- Part 3: SQL/CLI (Call-Level Interface; 414 Seiten)
  - Zugriff auf DBMS mittels Funktionsaufrufen
- Part 4: SQL/PSM (Persistent Stored Modules); 182 Seiten)
  - Prozedurale Erweiterungen
- Part 9: SQL/MED (Management of External Data; 504 Seiten)
  - Neue Datentypen und Funktionen
- Part 10: SQL/OLB (Object Language Bindings; 382 Seiten)
  - Java
- Part 11: SQL/Schemata (Information and Definition Schemata; 284 Seiten)
- Part 13: SQL/JRT (Java Routines und Types; 212 Seiten)
  - Externe Java Routinen
- Part 14: SQL/XML (XML-related Specifications; 154 Seiten)
  - XML Datentyp und Erweiterung von SQL um XQuery

Zusammen:
3534 Seiten

Überblick

- Historie von SQL
- **Einfache Anfragen**
- Anfragen über mehrere Relationen
- Geschachtelte Anfragen
- Operationen auf einer Relation
- DDL und DML

Beispielschema

- Filme(Titel, Jahr, Länge, inFarbe, StudioName, ProduzentID)
- spielt_in(FilmTitel, FilmJahr, Name)
- Schauspieler(Name, Adresse, Geschlecht, Geburtstag)
- Manager(Name, Adresse, ManagerID, Gehalt)
- Studios(Name, Adresse, VorsitzenderID)

Der Grundbaustein

- SELECT …
  FROM …
  WHERE …

  Grundstruktur einer SQL-Anfrage

  - SELECT *
    FROM Filme
    WHERE StudioName = ‚Disney' AND Jahr= 1990;

- Lesereihenfolge (und Schreibreihenfolge):
  - FROM: Relation(en) aus denen die Daten stammen
  - WHERE: Bedingung(en) an die Daten
  - SELECT: Schema der Ergebnisrelation
    - *: Alle Attribute der Inputrelationen
- Ausführung
  - Für jedes Tupel aus „Filme" prüfe die Bedingungen und gebe gültige Tupel aus.

Projektion in SQL (SELECT)

- Spezifikation in der SELECT Klausel
  - SELECT *
    - Alle Attribute
  - SELECT Titel, Jahr, inFarbe
    - Projektion auf die drei Attribute

- Erweiterte Projektion
  - Umbenennung:
    - SELECT Titel AS Name, Jahr AS Zeit
  - Arithmetischer Ausdruck:
    - SELECT Titel, Länge * 0.016667 AS Stunden
  - Konstanten:
    - SELECT Titel, Länge * 0.016667 AS Stunden, ‚std.' AS inStunden

Selektion in SQL

- Spezifikation in der WHERE Klausel
  - Bedingungen wie in einer Programmiersprache
  - Sechs Vergleichsoperatoren
    - =, <>, <, >, <=, >=
  - Operanden
    - Konstanten und Attributnamen
    - Auch Attribute, die nicht in der SELECT Klausel genannt werden.
  - Arithmetische Ausdrücke für numerische Attribute
    - Z.B.: (Jahr -1930) * (Jahr-1930) < 100
  - Konkatenation für Strings
    - ‚Star' || ‚Wars' entspricht ‚StarWars'

Selektion in SQL

- Ergebnis ist Boole‘scher Wert
  - TRUE oder FALSE
  - Können mit AND, OR und NOT verknüpft werden.
    - Klammerungen sind erlaubt.

- Beispiele
  - SELECT Titel
    FROM Filme
    WHERE Jahr > 1970 AND NOT inFarbe;
  - SELECT Titel
    FROM Filme
    WHERE (Jahr > 1970 OR Länge < 90) AND StudioName = ‚MGM‘;

Stringvergleiche

- Datentypen
  - Array fester Länge, Buchstabenliste variabler Länge, Konstanten
- foo _ _ _ _ _ = foo = ‚foo‘
- Vergleiche mit <, >, <=, >=
  - Lexikographischer Vergleich
  - ‚fodder‘ < ‚foo‘; ‚bar‘ < ‚bargain‘
- Patternmatching
  - string LIKE pattern bzw. string NOT LIKE pattern
  - Pattern hat spezielle Zeichen
    - ‚%‘: Beliebige Sequenz von 0 oder mehr Zeichen
    - ‚_‘: Ein beliebiges Zeichen
  - SELECT Titel FROM Filme
    WHERE Titel LIKE ‚Star _ _ _ _‘;
    - Star Wars und Star Trek
  - SELECT Titel FROM Filme WHERE Titel LIKE ‚%‘‘s%‘;

Datum und Uhrzeit

- Spezielle Datentypen und Repräsentationen
  - Datumskonstante:
    - DATE ‚YYYY-MM-DD‘
    - DATE ‚1948-05-14‘
  - Zeitkonstante
    - TIME ‚HH:MM:SS.S‘
    - TIME ‚15:00:02.5‘
  - Zeitstempel
    - TIMESTAMP , 1948-05-14 15:00:02.5‘
  - Zeitvergleiche
    - TIME ‚15:00:02.5‘ < TIME ‚15:02:02.5‘ ergibt TRUE
    - DATE ‚1948-05-14‘ >= DATE ‚1949-11-12‘ ergibt FALSE

Nullwerte

- Darstellung: NULL bzw. ⊥
- Mögliche Interpretationen
  - Unbekannter Wert
    - Geburtstag eines Schauspielers
  - Wert unzulässig
    - Ehegatte eines unverheirateten Schauspielers
  - Wert unterdrückt
    - Geheime Telefonnummer
- Regeln für Umgang mit Nullwerten
  - Arithmetische Operationen mit NULL ergeben NULL
  - Vergleich mit NULL ergibt Wahrheitswert UNKNOWN
  - NULL ist keine Konstante, sondern erscheint nur als Attributwert
- Beispiel (sei der Wert von x NULL):
  - x+3 ergibt NULL.
  - NULL+3 ist kein zulässiger Ausdruck.
  - x = 3 ergibt UNKNOWN.
- Abfrage von Nullwerten
  - Geburtstag IS NULL

Wahrheitswerte

and	true	unknown	false
true	true	unknown	false
unknown	unknown	unknown	false
false	false	false	false

or	true	unknown	false
true	true	true	true
unknown	true	unknown	unknown
false	true	unknown	false

not	
true	false
unknown	unknown
false	true

- Eselsbrücke
  - TRUE = 1, FALSE = 0, UNKNOWN = ½
  - AND: Minimum der beiden Werte
  - OR: Maximum der beiden Werte
  - NOT: 1 – Wert
  - Beispiel
    - TRUE AND (FALSE OR NOT(UNKNOWN))
      = MIN(1, MAX(0, (1 - ½)))
      = MIN(1, MAX(0, ½)
      = MIN(1, ½) = ½.

Wahrheitswerte

Titel	Jahr	Länge	inFarbe	Studio	ProduzentID
Total Recall	1990	NULL	True	Fox	12345

- Überraschendes Verhalten
  - SELECT *
    FROM Filme
    WHERE Länge <= 90 OR Länge > 90;

    UNKNOWN UNKNOWN

    UNKNOWN
  - Tupel erscheint nicht im Ergebnis.

Sortierung

- ORDER BY Klausel ans Ende der Anfrage
  - ORDER BY <Attributliste> DESC/ASC
  - ASC (aufsteigend) ist default
  - SELECT *
    FROM Filme
    WHERE StudioName = ‚Disney' AND Jahr = 1990
    ORDER BY Länge, Titel;

  - SELECT *
    FROM Filme
    WHERE StudioName = ‚Disney' AND Jahr = 1990
    ORDER BY Länge ASC, Titel DESC;

Groß- und Kleinschreibung

- In SQL wird Groß- und Kleinschreibung nicht beachtet
  - From = FROM = from = FrOm
  - Auch bei Attribut- und Relationennamen
  - Aber
    - ‚FROM‘ ≠ ‚from‘ ≠ from

Text in Anführungszeichen -> Groß/Kleinschreibung wird beachtet!

Überblick

- Historie von SQL
- Einfache Anfragen
- **Anfragen über mehrere Relationen**
- Geschachtelte Anfragen
- Operationen auf einer Relation
- DDL und DML

Motivation

- Hauptkraft der Relationalen Algebra ist die Kombination von Relationen
- Erst mit mehreren Relationen sind viele interessante Anfragen möglich.

- Nennung der beteiligten Relationen in der FROM Klausel

Kreuzprodukt und Join

- Filme(Titel, Jahr, Länge, inFarbe, StudioName, ProduzentID)
- Manager(Name, Adresse, ManagerID, Gehalt)

- SELECT Name
  FROM Filme, Manager
  WHERE Titel = ‚Star Wars' AND ProduzentID = ManagerID;
- Semantik
  - Betrachte jedes Tupelpaar der Relationen Filme und Manager.
  - Wende Bedingung der WHERE Klausel auf jedes Tupelpaar an
  - Falls Bedingung erfüllt, produziere ein Ergebnistupel.
- Kreuzprodukt gefolgt von Selektion: Join

Uneindeutige Attributnamen

- Schauspieler(Name, Adresse, Geschlecht, Geburtstag)
- Manager(Name, Adresse, ManagerID, Gehalt)

- Bei gleichen Attributnamen aus mehreren beteiligten Relationen:
  - Relationenname als Präfix:
    - SELECT Schauspieler.Name, Manager.Name
      FROM Schauspieler, Manager
      WHERE Schauspieler.Adresse = Manager.Adresse;

- Präfix ist auch erlaubt wenn Attributname eindeutig ist.
  - Erleichtert das Lesen von SQL Anfragen

Tupelvariablen

- Zur eindeutigen Kennzeichnung von Tupeln beteiligter Relationen
  - „Alias“ einer Relation
  - Insbesondere: Bei der mehrfachen Verwendung einer Relation in einer Anfrage
  - Gesucht: Schauspieler, die zusammen leben
    - SELECT Star1.Name, Star2.Name
      FROM Schauspieler Star1, Schauspieler Star2
      WHERE Star1.Adresse = Star2.Adresse
  - Auch sinnvoll als abkürzenden Schreibweise
    - SELECT S.Name, M.Name
      FROM Schauspieler S, Manager M
      WHERE S.Adresse = M.Adresse;
  - Ohne explizites Angeben einer Tupelvariablen wir der Relationsname selbst als Tupelvariable verwendet.

Tupelvariablen

Name	Adresse	Geschlecht	Geburt
Carrie Fisher	123 Maple St., Hollywood	F	9/9/99
Mark Hamill	456 Oak Rd., Brentwood	M	8/8/88
Brad Pitt	123 Maple St., Hollywood	M	7/7/77

- SELECT Star1.Name, Star2.Name
  FROM Schauspieler Star1, Schauspieler Star2
  WHERE Star1.Adresse = Star2.Adresse;

Star1.Name	Star2.Name
Carrie Fisher	Carrie Fisher
Carrie Fisher	Brad Pitt
Brad Pitt	Carrie Fisher
Brad Pitt	Brad Pitt
Mark Hamill	Mark Hamill

- SELECT Star1.Name, Star2.Name
  FROM Schauspieler Star1, Schauspieler Star2
  WHERE Star1.Adresse = Star2.Adresse
  AND Star1.Name <> Star2.Name;

Star1.Name	Star2.Name
Carrie Fisher	Brad Pitt
Brad Pitt	Carrie Fisher

- SELECT Star1.Name, Star2.Name
  FROM Schauspieler Star1, Schauspieler Star2
  WHERE Star1.Adresse = Star2.Adresse
  AND Star1.Name < Star2.Name;

Star1.Name	Star2.Name
Brad Pitt	Carrie Fisher

Interpretation von Anfragen

- Drei Interpretationsvarianten für Anfragen mit mehreren Relationen
  - Nested Loops (geschachtelte Schleifen)
    - Bei mehreren Tupelvariablen: Eine geschachtelte Schleife für jede Variable
  - Parallele Zuordnung
    - Alle Kombinationen werden parallel bezüglich der Bedingungen geprüft.
  - Relationale Algebra
    - Bilde Kreuzprodukt
    - Wende Selektionsbedingungen auf jedes Resultat-Tupel an

Interpretation von Anfragen

- Gegeben drei Relationen: R(A), S(A) und T(A)
- Gesucht: $R \cap (S \cup T)$
  - SELECT R.A
    FROM R, S, T
    WHERE R.A = S.A OR R.A = T.A;

- Problemfall: T ist leer, hat also kein Tupel
- Vermeintliches Resultat: $R \cap S$
- Tatsächliches Resultat: leere Menge
  - Nested Loops
  - Parallele Zuordnung
  - Relationale Algebra

Mengenoperationen in SQL

- Vereinigung: UNION
- Schnittmenge: INTERSECT
- Differenz: EXCEPT
- Mengenoperationen nur zwischen (geklammerten) Anfrageergebnissen
  - Schauspieler(Name, Adresse, Geschlecht, Geburtstag)
  - Manager(Name, Adresse, ManagerID, Gehalt)

  - (SELECT Name, Adresse FROM Schauspieler
    WHERE Geschlecht = ‚F‘)
    INTERSECT
    (SELECT Name, Adresse FROM Manager
    WHERE Gehalt > 1.000.000);

Mengenoperationen in SQL

- Schauspieler(Name, Adresse, Geschlecht, Geburtstag)
- Manager(Name, Adresse, ManagerID, Gehalt)
- Filme(Titel, Jahr, Länge, inFarbe, StudioName, ProduzentID)
- spielt_in(FilmTitel, FilmJahr, Name)

- (SELECT Name, Adresse FROM Schauspieler)
  EXCEPT
  (SELECT Name, Adresse FROM Manager);
- (SELECT Titel, Jahr FROM Filme)
  UNION
  (SELECT FilmTitel AS Titel, FilmJahr AS Jahr FROM spielt_in)

Überblick

- Historie von SQL
- Einfache Anfragen
- Anfragen über mehrere Relationen
- **Geschachtelte Anfragen**
- Operationen auf einer Relation
- DDL und DML

Motivation

- Eine Anfrage kann Teil einer anderen Anfrage sein
  - Beliebig tiefe Schachtelung

- Anwendungsmöglichkeiten
  - Subanfrage erzeugt einen einzigen Wert, der in der WHERE-Klausel mit einem anderen Wert verglichen werden kann.
  - Subanfrage erzeugt eine Relation, die auf verschiedene Weise in WHERE-Klausel verwendet werden kann.
  - Subanfrage erzeugt eine Relation, die in der FROM Klausel verwendet werden kann.
    - Wie jede normale Relation

Skalare Subanfragen

- Allgemeine Anfrage produzieren Relationen.
  - Mit mehreren Attributen
    - Zugriff auf ein bestimmtes Attribut ist möglich
  - i.A. mit mehreren Tupeln
  - Manchmal (garantiert) nur ein Tupel
    - „Skalare Anfrage“
    - Verwendung wie eine Konstante möglich

Skalare Subanfragen

- Manager(Name, Adresse, ManagerID, Gehalt)
- Filme(Titel, Jahr, Länge, inFarbe, StudioName, ProduzentID)
- Gesucht: Produzent von Star Wars
  - SELECT Name
    FROM Filme, Manager
    WHERE Titel = ‚Star Wars‘ AND Jahr = ‚1977‘
    AND ProduzentID = ManagerID;
- Oder aber
  - SELECT Name
    FROM Manager
    WHERE ManagerID =
    (SELECT ProduzentID
    FROM Filme
    WHERE Titel = ‚Star Wars‘ AND Jahr = ‚1977‘);
- Wir erwarten genau ein Tupel als Ergebnis der Teilanfrage
  - Falls kein Tupel: Laufzeitfehler
  - Falls mehr als ein Tupel: Laufzeitfehler

Bedingungen mit Relationen

- Bestimmte SQL Operatoren auf Relationen erzeugen Boole'sche Werte
  - EXISTS R
    - TRUE, falls R nicht leer
  - x IN R
    - TRUE falls x gleich einem Wert in R ist (R hat nur ein Attribut)
    - Verallgemeinerung auf Tupel gleich
    - x NOT IN R: TRUE falls x keinem Wert in R gleicht
  - x > ALL R
    - TRUE falls x größer als jeder Wert in R ist (R hat nur ein Attribut)
    - Alternativ: <, >, <=, >=, <>, =
    - x <> ALL R: Entspricht x NOT IN R
  - x > ANY R
    - TRUE falls x größer als mindestens ein Wert in R ist (R hat nur ein Attribut)
    - Alternativ: <, >, <=, >=, <>, =
    - x = ANY R: Entspricht x IN R
  - Negation mit NOT ist immer möglich.

Bedingungen mit Tupeln

- Verallgemeinerung von IN, ALL und ANY auf Tupel
  - t IN R
    - TRUE falls t ein Tupel in R ist (mehr als ein Attribut möglich)
    - Setzt gleiche Schemata voraus
    - Setzt gleiche Reihenfolge der Attribute voraus
  - t > ALL R
    - TRUE falls t größer als jedes Tupel in R ist
    - Vergleiche in Standardreihenfolge der Attribute
  - t <> ANY R
    - TRUE falls R mindestens ein Tupel hat, das ungleich t ist

Bedingungen mit Tupeln

- SELECT Name
  FROM Manager
  WHERE ManagerID IN
  (SELECT ProduzentID
  FROM Filme
  WHERE (Titel, Jahr) IN
  (SELECT FilmTitel, FilmJahr
  FROM spielt_in
  WHERE SchauspielerName = ‚Harrison Ford‘
  ));
- Analyse am besten von innen nach außen
- Namen von Produzenten von Filmen mit Harrison Ford
- Alternative Formulierung
  - SELECT Name
    FROM Manager, Filme, spielt_in
    WHERE ManagerID = ProduzentID
    AND Titel = FilmTitel
    AND Jahr = FilmJahr
    AND SchauspielerName = ‚Harrison Ford‘;

Korrelierte Subanfragen

- Bisher: Subanfragen einmalig ausführen und das Ergebnis weiterverwenden
  - Korrelierte Subanfragen werden mehrfach ausgeführt, einmal pro Bindung der korrelierten Variable der äußeren Anfrage
  - SELECT Titel, Jahr
    FROM Filme Alt
    WHERE Jahr < ANY
      (SELECT Jahr
        FROM Filme
        WHERE Titel = Alt.Titel);
    - Ausführung der Subanfrage für jedes Tupel in Filme
    - Alle mehrfachen Filme mit Ausnahme der jeweils jüngsten Ausgabe sind im Ergebnis.

Scope: Attributnamen gehören i.d.R. zur Tupelvariablen der aktuellen Anfrage. Sonst: Suche von innen nach außen.

SQL entfernt idR keine Duplikate!

Subanfragen in FROM-Klausel

- Bisher: Nur Subanfragen in WHERE-Klausel
  - Anstelle einfacher Relation steht eine geklammerte Subanfrage
  - Es muss ein Alias vergeben werden.
  - SELECT M.Name
    FROM Manager M, (SELECT ProduzentID AS ID
    FROM Filme, spielt_in
    WHERE Titel = FilmTitel
    AND Jahr = FilmJahr
    AND Schauspieler = ‚Harrison Ford') Produzent
    WHERE M.ManagerID = Produzent.ID;

Joins

- Man kann Joins auch auf andere Weise ausdrücken.
  - Besonders praktisch für Teilanfragen
  - Filme CROSS JOIN spielt_in
    - Kreuzprodukt
    - Doppelte Attributnamen werden mit Präfix der Relation aufgelöst
  - Filme JOIN spielt_in ON Titel = FilmTitel AND Jahr = FilmJahr
    - Theta-Join
    - SELECT Titel, Jahr, Länge, inFarbe, StudioName, ProduzentID, SchauspielerName
      FROM Filme JOIN spielt_in ON Titel = FilmTitel AND Jahr = FilmJahr;
    - Eliminiert redundante Attribute FilmTitel und FilmJahr
  - Schauspieler NATURAL JOIN Manager
    - Natural Join; Eliminiert redundante Attribute

Outer Joins

- Schauspieler(Name, Adresse, Geschlecht, Geburtstag)
- Manager(Name, Adresse, ManagerID, Gehalt)
- Schauspieler NATURAL FULL OUTER JOIN Manager;
- Schauspieler NATURAL LEFT OUTER JOIN Manager;
- Schauspieler NATURAL RIGHT OUTER JOIN Manager;

- Filme FULL OUTER JOIN spielt_in
  ON Titel = FilmTitel AND Jahr = FilmJahr;
  - Widerspruch in sich? Es sollen schliesslich alle Tupel erhalten bleiben…
- Filme LEFT OUTER JOIN spielt_in
  ON Titel = FilmTitel AND Jahr = FilmJahr;
- Filme RIGHT OUTER JOIN spielt_in
  ON Titel = FilmTitel AND Jahr = FilmJahr;

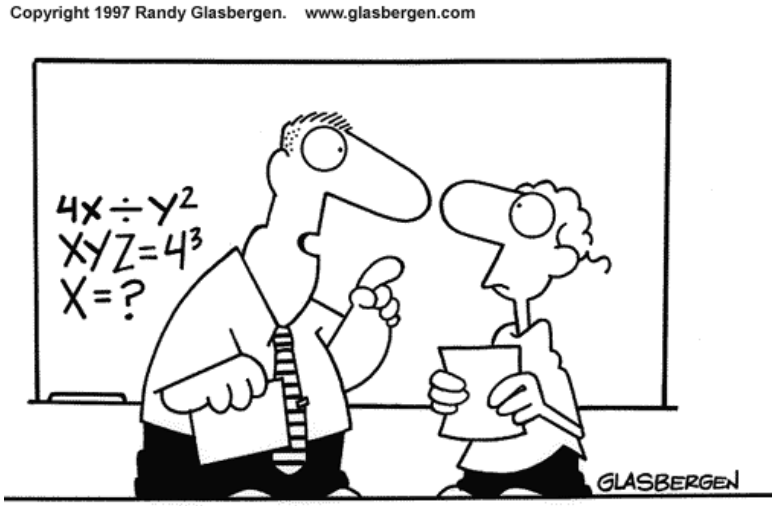
Create your Own Exam: SQL

- Bitte erstellen Sie eine Multiple Choice Aufgabe zum Thema SQL
  - Formulieren Sie eine Frage und 3 Antworten (A, B, C)
  - Davon sollte mindestens eine Antwort richtig und mindestens eine Antwort falsch sein

- Geben Sie die Aufgabe an Ihren rechten Nachbarn. Diskutieren Sie gemeinsam und markieren Sie die richtigen Lösungen

- Geben Sie am Ende der Vorlesung Ihre Aufgabe bei mir ab

5 min



Überblick

- Historie von SQL
- Einfache Anfragen
- Anfragen über mehrere Relationen
- Geschachtelte Anfragen
- **Operationen auf einer Relation**
- DDL und DML

Duplikateliminierung

- Relationale DBMS verwenden idR Multimengensemantik, nicht Mengensemantik.
  - Duplikate entstehen durch
    - Einfügen von Duplikaten in Basisrelation
    - Veränderung von Tupeln in Basisrelation
    - Projektion in Anfragen
    - Durch Subanfragen (UNION)
    - Vermehrung von Duplikaten durch Kreuzprodukt
  - Duplikateliminierung durch SELECT DISTINCT Attributnamen
    - Kosten sind hoch: Sortierung

Duplikateliminierung

- SELECT ManagerID, Name
  FROM Manager
  WHERE ManagerID IN
    (SELECT ProduzentID
      FROM Filme
      WHERE (Titel, Jahr) IN
        (SELECT FilmTitel, FilmJahr
          FROM spielt_in
          WHERE
  SchauspielerName =
        ‚Harrison Ford‘
        ));

- SELECT ManagerID, Name
  FROM Manager, Filme,
  spielt_in
  WHERE ManagerID =
  ProduzentID
  AND Titel = FilmTitel
  AND Jahr = FilmJahr
  AND SchauspielerName =
  ‚Harrison Ford‘;
- SELECT DISTINCT ManagerID,
  Name
  FROM Manager, Filme,
  spielt_in
  WHERE ManagerID =
  ProduzentID
  AND Titel = FilmTitel
  AND Jahr = FilmJahr
  AND SchauspielerName =
  ‚Harrison Ford‘;

Duplikateliminierung bei Mengenoperationen

- Mengenoperationen in SQL entfernen Duplikate
  - UNION, INTERSECT, EXCEPT
    - wandeln Multimengen in Mengen um und verwenden Mengensemantik
  - Duplikateliminierung verhindern durch ALL
    - (SELECT Titel, Jahr, FROM Filme)
      UNION ALL
      (SELECT FilmTitel AS Titel, FilmJahr AS Jahr FROM spielt_in);
    - Film mit drei Schauspielern erscheint also 4 Mal im Ergebnis
  - R INTERSECT ALL S
  - R EXCEPT ALL S

Aggregation

- Standardaggregationsoperatoren
  - SUM, AVG, MIN, MAX, COUNT
  - Angewendet auf einzelne Attribute in der FROM-Klausel
- Typische weitere Aggregationsoperatoren
  - VAR, STDDEV
- COUNT(*) zählt Anzahl der Tupel
  - in der Relation, die durch die FROM und WHERE Klauseln definiert wird.
- Kombination mit DISTINCT
  - COUNT(DISTINCT Jahr)
  - SUM(DISTINCT Gehalt)

Aggregation

- SELECT AVG(Gehalt)
  FROM Manager;
- SELECT COUNT(*)
  FROM spielt_in;
- SELECT COUNT(Schauspieler)
  FROM spielt_in;
- SELECT COUNT(DISTINCT Schauspieler)
  FROM spielt_in;

Gruppierung

- Gruppierung mittels GROUP BY nach der WHERE-Klausel
- SELECT StudioName, SUM(Länge)
  FROM Filme
  GROUP BY StudioName
- In SELECT-Klausel zwei Sorten von Attributen
  - Gruppierungsattribute
  - Aggregierte Attribute
  - Nicht-aggregierte Werte der SELECT-Klausel müssen in der GROUP BY-Klausel erscheinen.
  - Beide Sorten müssen nicht erscheinen
- SELECT StudioName
  FROM Filme
  GROUP BY StudioName
- SELECT SUM(Länge)
  FROM Filme
  GROUP BY StudioName

Gruppierung

- Gruppierung bei mehreren Relationen wird am Schluss durchgeführt.
  - SELECT Name, SUM(Länge)
    FROM Manager, Filme
    WHERE ManagerID = ProduzentID
    GROUP BY Name
  - Reihenfolge der Ausführung
    - FROM-Klausel
    - WHERE-Klausel
    - GROUP BY-Klausel
    - SELECT-Klausel

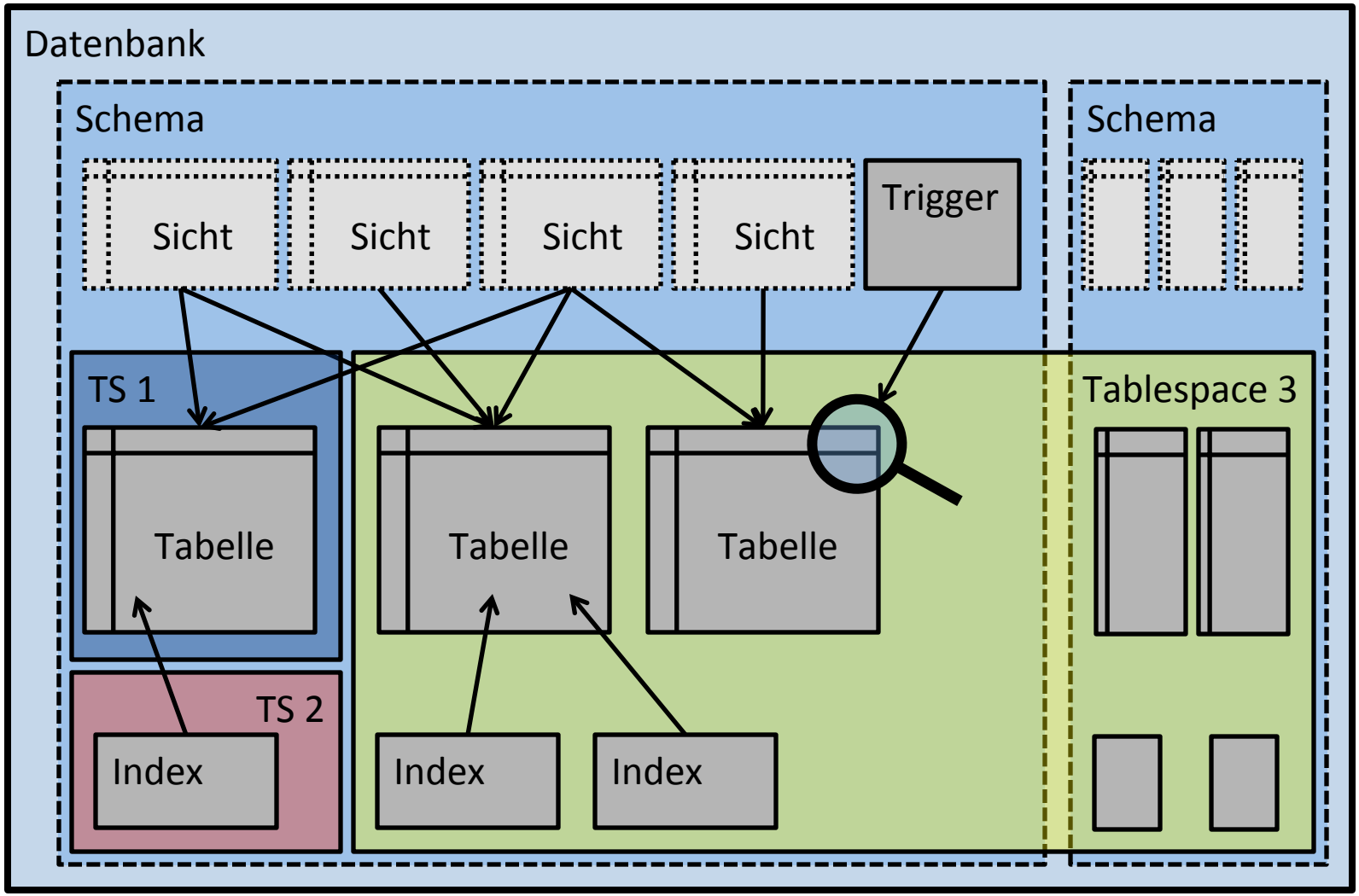
Gruppierung

- Einschränkung der Ergebnismenge nach der Gruppierung durch HAVING
  - Einschränkung der Ergebnismenge
    - SELECT Name, SUM(Länge)
      FROM Manager, Filme
      WHERE ManagerID = ProduzentID
      AND Gehalt > 1000000
      GROUP BY Name
    - SELECT Name, SUM(Länge)
      FROM Manager, Filme
      WHERE ManagerID = ProduzentID
      GROUP BY Name
      HAVING SUM(Länge) > 1000
    - SELECT Name, SUM(Länge)
      FROM Manager, Filme
      WHERE ManagerID = ProduzentID
      GROUP BY Name
      HAVING SUM(Länge) > 1000
  - Aggregationen in HAVING Klausel beziehen sich nur auf aktuelle Gruppe.
  - Nur Gruppierungsattribute dürfen unaggregiert in HAVING Klausel erscheinen (wie bei SELECT-Klausel).

Überblick

- Historie von SQL
- Einfache Anfragen
- Anfragen über mehrere Relationen
- Geschachtelte Anfragen
- Operationen auf einer Relation
- **DDL und DML, Datenbankprogrammierung**

Datenbankobjekte - Übersicht



Datenbankobjekte - Datenbank

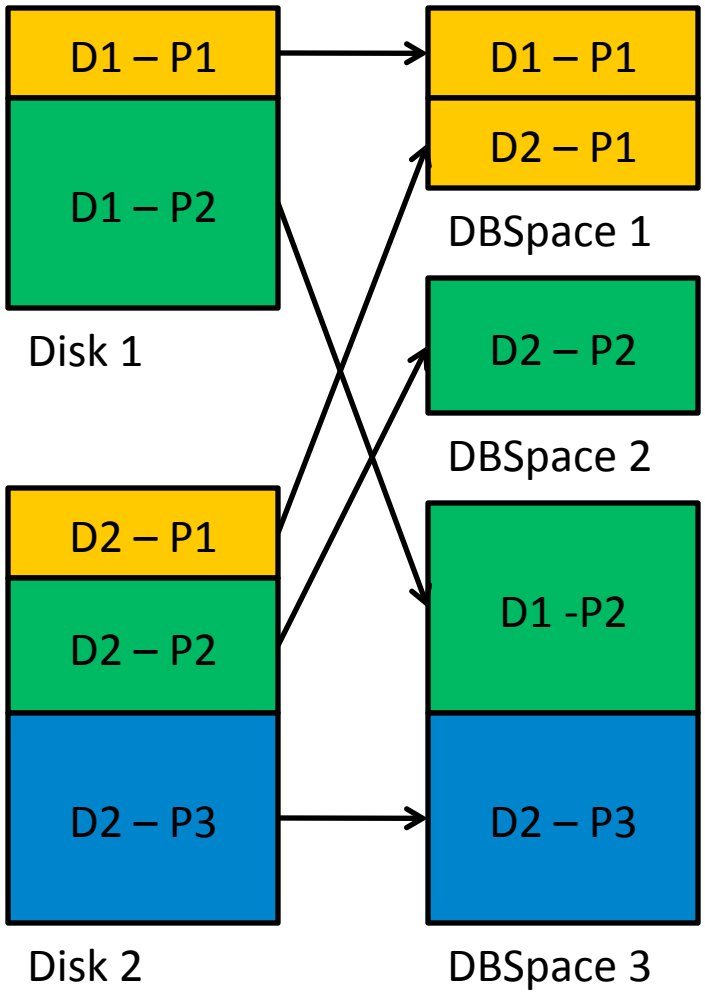
- Speichert Daten für eine oder mehrere Anwendungen
- Organisiert den effizienten Zugriff auf Daten
- Enthält alle Datenbank-Objekte
  - Datenspeicher (Tabellen)
  - Hilfsstrukturen für effizienten Datenzugriff (Indexe, MQTs)
  - Logische Organisation (Sichten)
  - Sicherung der Datenintegrität (Tabellen, Trigger)
  - Funktionserweiterung (Procedures, UDFs)
  - Rechteverwaltung (Schemata, Zugriffsmanagement)

Datenbankobjekte - Schema

- Logische Trennung innerhalb einer Datenbank
  - Gruppierung von DB-Objekten
  - Zugriffsrechte
- Jeder Benutzer hat standardmäßig ein eigenes Schema
  - DB-Objekte werden im eigenen Schema erzeugt
- Werden hier nicht weiter betrachtet
  - DB Administration

Datenbankobjekte - DBSpace

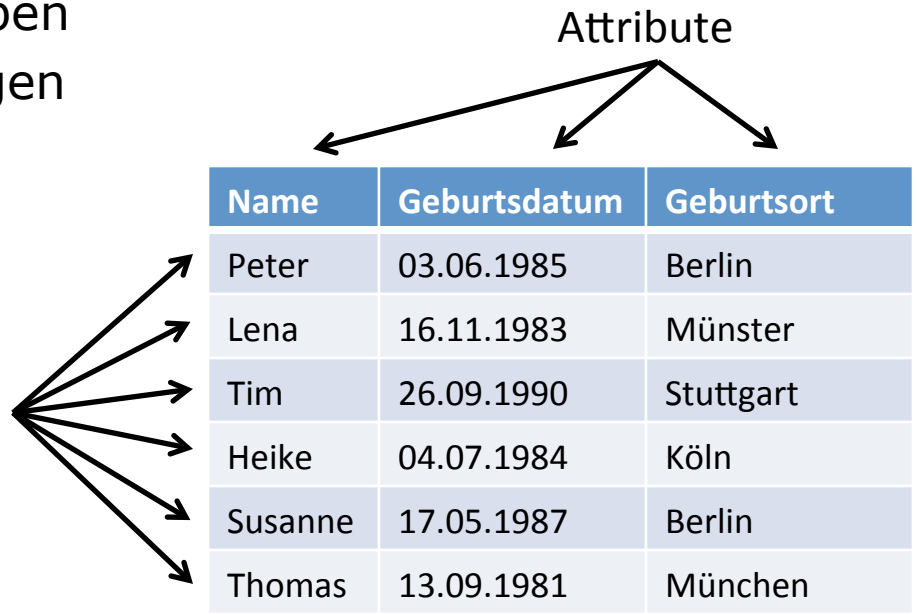
- Definition von Speicherorten für Daten
  - Zusammenfassen von Partitionen/ Festplatten
    - Mehr Platz
    - Mehr Transfer-Bandbreite
  - Begrenzung des verfügbaren Speicherplatzes
  - Auch "Tablespaces" genannt
- DBSpaces sind Schema übergreifend
- Viele Konfigurationsparameter
- Werden später betrachtet



Datenbankobjekte - Tabelle

- Speicherort für Daten
  - Daten werden in einer Tabelle gespeichert
  - Tabelle liegt in einem DBSpace
  - Daten sind über die Tabelle abfragbar

- Eine Tabelle ist eine Relation
- Festes Schema mit Schlüsselattributen
- Attribute haben einen Datentypen
- Definition von Nebenbedingungen
  - Referenzbedingungen
  - Standardwerte
  - Einzigartigkeit
  - ...



Name	Geburtsdatum	Geburtsort
Peter	03.06.1985	Berlin
Lena	16.11.1983	Münster
Tim	26.09.1990	Stuttgart
Heike	04.07.1984	Köln
Susanne	17.05.1987	Berlin
Thomas	13.09.1981	München

Datenbankobjekte - Tabelle

- Primärschlüssel (Primary Key)
  - Schlüssel, der für diese Tabelle gewählt ist
- Fremdschlüssel (Foreign Key)
  - Attribut(e) der Tabelle, die auf Attribute einer anderen Tabelle verweisen
  - Üblicherweise wird über Fremdschlüssel-Attribute gejoint
- Beispiel:

Name	Geburtsdatum	Geburtsort
Peter	03.06.1985	Berlin
Lena	16.11.1983	Münster
Tim	26.09.1990	Stuttgart
Heike	04.07.1984	Köln
Susanne	17.05.1987	Berlin
Thomas	13.09.1981	München

Ortname	Land
Berlin	BER
Münster	NRW
Stuttgart	BW
Köln	NRW
München	BAY

Primärschlüssel: (Name, Geburtsdatum)
Fremdschlüssel: Geburtsort -> Ort.Ortname

Primärschlüssel: (Ortsname)

Arbeiten mit DB-Objekten – Tabelle anlegen

Syntax:

```
CREATE TABLE tablename IN tablespacename

  ( ┌ , ──────────────────────────────────────┐
    │ columnname column-type column-options │ ,
    └ , ──────────────────────────────────────────────────────────────────────────────────────────────────┐
      FOREIGN KEY ( ┌ , ──────────┐ ) REFERENCES tablename ( ┌ , ──────────┐ ) referential-constraint │ ,
                    │ columnname │                          │ columnname │                          │
                    └ , ──────────┘                          └ , ──────────┘                          │
      PRIMARY KEY ( ┌ , ──────────┐ ))
                    │ columnname │
                    └ , ──────────┘
```

Beispiel:

```
CREATE TABLE person IN ts1
    (name VARCHAR(30) NOT NULL,
     geburtstag DATE NOT NULL,
     geburtsort VARCHAR(30),
     FOREIGN KEY (geburtsort) REFERENCES ort (name) ON DELETE CASCADE,
     PRIMARY KEY (name, geburtstag) )
```

Gültige Spaltentypen, Optionen und Referenzbedingungen werden auf den nächsten Slides aufgelistet

Datenbankobjekte - Index

- Hilfsstruktur für schnellen Datenzugriff

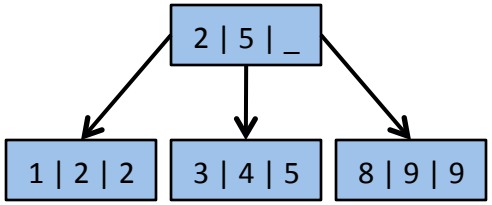
- Wird auf einem (oder mehreren) Attributen definiert
- Speichert Daten der Attribute sortiert
  - Speicherstrukturen: B-Baum-Varianten / Bitmaps / Hash-Tables
  - Effiziente Suche
  - TID (TupleID) gibt den Speicherort eines Datensatzes an (Pointer)

- Vorteile: Schnellerer Datenzugriff (Lesen)
- Nachteile: Langsameres Ändern der Daten (Schreiben)
  - Index muss aktualisiert werden

- Beispiel: Unterstützung für Anfrage: "*Personen, die in Münster geboren sind*"

Name	Geburtsdatum	Geburtsort
Peter	03.06.1985	Berlin
Lena	16.11.1983	Münster
Tim	26.09.1990	Stuttgart
Heike	04.07.1984	Köln
Susanne	17.05.1987	Berlin
Thomas	13.09.1981	München

TID	Geburtsort
•	Berlin
•	Berlin
•	Köln
•	München
•	Münster
•	Stuttgart



Index auf Attr. Geburtsort

Index anlegen/löschen

Syntax:

```
CREATE INDEX indexname ON tablename ( ,──────────┐
                                      columnname )
       IN DBSpaceName
```

Beispiel:

```
CREATE INDEX gebOrtIDX ON person (geburtsort) IN tbSpc1
```

- Legt einen Standard B-Tree Index an
  - Für andere Index Typen:
  - … ON <table> (columns) USING BITMAP …
  - … ON <table> (columns) USING RTREE …
- Einrichten eines Index kann einige Zeit dauern
- In einigen Fällen legt das Datenbanksystem selber Indexe an.
  - Bsp: Informix legt immer B-Tree Indexe für Primary Keys and Foreign Keys an.
- Löschen mit DROP INDEX

Datenbankobjekte - Sicht

- Logische Sicht auf Daten
  - Speichern keine Daten
  - Definition bezieht sich auf Tabellen oder andere Sichten
  - Wird abgefragt wie eine Tabelle
  - Sichten werden bei Anfragen on-the-fly berechnet

- Nutzen
  - Integration von Daten
  - Modellieren von Zugriffsrechten
  - Häufige Anfragen als Sichten
    - spart Optimierungskosten
    - Vereinfacht Anfragen

- Beispiel: Sicht *"Anzahl der Personen gruppiert nach Land"*

Name	Geburtsdatum	Geburtsort
Peter	03.06.1985	Berlin
Lena	16.11.1983	Münster
Tim	26.09.1990	Stuttgart
Heike	04.07.1984	Köln
Susanne	17.05.1987	Berlin
Thomas	13.09.1981	München

Ortsname	Land
Berlin	BER
Münster	NRW
Stuttgart	BW
Köln	NRW
München	BAY

Land	Anzahl Personen
BAY	1
BER	2
BW	1
NRW	2

Sichten in SQL

- `CREATE VIEW` *Name* `AS` *Anfrage*

- Beispiel:

```
CREATE VIEW ParamountFilme AS
     SELECT Titel, Jahr
     FROM Film
     WHERE StudioName = 'Paramount Pictures';
```

  - Beliebig komplexe Anfragen möglich!

- Semantik
  - Bei jeder Anfrage an die Sicht wird die SQL Anfrage der Sicht ausgeführt.
  - Die ursprüngliche Anfrage verwendet das Ergebnis als Relation.

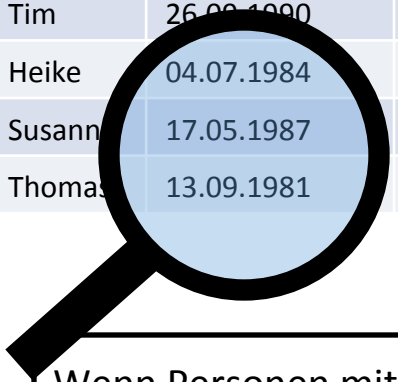
- Daten der Sicht ändern sich mit der Änderung der zugrundeliegenden Relationen.

- Entfernen der Sicht: `DROP VIEW ParamountFilme`
  - Basisdaten bleiben unverändert.

Datenbankobjekte - Trigger

- Definiert bedingte Aktionen
- Aktionen werden ausgelöst durch Ändern von Tabellendaten
  - Einfügen
  - Löschen
  - Ändern
- Zeitpunkt der Aktion
  - bevor,
  - anstatt oder
  - nach der Änderung
- Anzahl der Aktionen
  - pro Statement
  - pro betroffenem Tupel
- Komplexe Aktionen möglich
  - Umfang: SQL Procedure
  - Bedingungen (IF, WHEN, CASE)
  - Schleifen (WHILE, ITERATE)
  - SQL: SELECT, INSERT, UPDATE, DELETE
  - …

Name	Geburtsdatum	Geburtsort
Peter	03.06.1985	Berlin
Lena	16.11.1983	Münster
Tim	26.09.1990	Stuttgart
Heike	04.07.1984	Köln
Susann	17.05.1987	Berlin
Thomas	13.09.1981	München



Wenn Personen mit (geburtsdatum < “01-01-1930”) in Tabelle *Person* eingefügt werden sollen,
speichere sie stattdessen in seperater Tabelle *AltePerson*

Zusammenfassung und Ausblick

- Historie von SQL
- Einfache Anfragen
  - Der SFW Block
  - Nullwerte
  - Mengen vs. Multimengen
- Anfragen über mehrere Relationen
  - UNION, INTERSECT, EXCEPT
  - Joins und Outerjoins
- Geschachtelte Anfragen
  - In FROM und WHERE
  - EXISTS, IN, ALL, ANY
- Operationen auf einer Relation
  - GROUP BY, HAVING
  - ORDER BY
- DDL und DML, Datenbankprogrammierung
  - Create Table, Trigger, Views
  - ESQL

In der nächsten Veranstaltung:

- Data Warehousing
  (Kapitel 10.6 und 10.7 des Lehrbuches)