Einführung in Datenbanken

Übung 12: Einführung in den Datenbank-Entwurf

Prof. Dr. Stefan Brass
PD Dr. Alexander Hinneburg
Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2020/21

http://www.informatik.uni-halle.de/~brass/db20/

Zur Klausur

- Das Rektorat hat verordnet, dass es keine Präsenzprüfungen bis Ende April geben kann.
- Die Klausur vom 1. März wird voraussichtlich online stattfinden (mit Videoüberwachung etc.).

  Es sei denn, die für Mitte Februar angekündigten Regeln machen es mir unmöglich, die Klausur so durchzuführen, dass ich an die Integrität der Prüfung glauben kann. Dann würde die Klausur verschoben (nach den 30. April).
- Die Klausur vom 7. April wird
  - entweder verschoben (voraussichtlich Anfang Mai)
  - oder einzeln mündlich per Videokonferenz durchgeführt.
- In besonderen Härtefällen (mit guter Begründung!) sind einzelne mündliche Prüfungen möglich.

Inhalt

Hausaufgabe 10a (1)

- Numerieren Sie die Angestellten nach Abteilungen (`deptno`), und in jeder Abteilung alphabetisch nach dem Namen durch.

  Sortieren Sie die Ausgabe nach der so erzeugten Spalte „`id`“. Sie dürfen den Angestellten mit einem Nullwert in `deptno` ignorieren. Es gibt einen Bonuspunkt, wenn Sie Angestellte ohne Abteilung ganz hinten einsortieren (innerhalb dieser Gruppe wieder alphabetisch).

id	deptno	ename
1	10	CLARK
2	10	KING
3	10	MILLER
4	20	ADAMS
⋮	⋮	⋮
13	30	WARD
14	NULL	SMITH

Hausaufgabe 10a (2)

- emp(empno, ename, job, mgr°→emp, hiredate, sal, comm°, deptno°→dept)
- Tipp: Zählen Sie, wie viele Angestelle vor dem aktuellen Angestellten kommen müssen (oder gleich sind).
- Beispiel (noch nicht die Lösung, nur alphabetische IDs):

```
SELECT COUNT(*) id, e.ename
FROM   emp vorher, emp e
WHERE  vorher.ename <= e.ename
GROUP  BY e.empno, e.ename
ORDER  BY id
```

- Für ADAMS wird nur ADAMS gezählt,
  für ALLEN werden ADAMS und ALLEN gezählt,
  für BLAKE werden ADAMS, ALLEN, BLAKE gezählt, u.s.w.

Hausaufgabe 10a (3)

- Man muss jetzt die gewünschte Sortierung durch die WHERE-Bedingung beschreiben.

  Was bedeutet es, dass der Angestellte `vorher` in der Sortierreihenfolge (`deptno`, `ename`) vor dem Angestellten `e` kommt (dessen Position in dieser Ausgabezeile berechnet wird)? Genauer ist ein $\leq$-Vergleich nötig.

- Lösung ohne Bonuspunkt:

```
SELECT COUNT(*) id, e.deptno, e.ename
FROM   emp vorher, emp e
WHERE  vorher.deptno < e.deptno
OR     (vorher.deptno = e.deptno
        AND vorher.ename <= e.ename)
GROUP  BY e.empno, e.deptno, e.ename
ORDER  BY id
```

Die Klammern sind nicht nötig. `AND` bindet stärker als `OR`.

Hausaufgabe 10a (4)

- Lösung mit Bonus-Punkt (Angestellte mit Nullwert in deptno ganz hinten, innerhalb dieser Gruppe alphabetisch):

```
SELECT COUNT(*) id, e.deptno, e.ename
FROM   emp vorher, emp e
WHERE  vorher.deptno < e.deptno
OR     (vorher.deptno = e.deptno
        AND vorher.ename <= e.ename)
OR     (vorher.deptno IS NOT NULL
        AND e.deptno IS NULL)
OR     (vorher.deptno IS NULL
        AND e.deptno IS NULL
        AND vorher.ename <= e.ename)
GROUP  BY e.empno, e.deptno, e.ename
ORDER  BY id
```

Hausaufgabe 10a (5)

- „Sie dürfen in dieser Aufgabe nur Sprachkonstrukte verwenden, die in der Vorlesung behandelt wurden.“
- Die folgende Lösung funktioniert, ist aber nicht zulässig:

```
SELECT ROW_NUMBER() OVER
         (ORDER BY deptno NULLS LAST, ename)
         AS id,
       deptno, ename
FROM   emp
ORDER  BY id
```

- Solche Konstrukte werden in der Vorlesung „DB-Programmierung“ im Sommer behandelt.

  „Window Functions“ wurden in SQL-2003 eingeführt.

Hausaufgabe 10b (1)

- In der Angestellten-Tabelle `emp` enthält die Spalte `mgr` die Nummer (`empno`) des direkten Vorgesetzten.
- Es sollen jetzt auch die indirekten Vorgesetzten-Beziehungen berechnet werden unter der Voraussetzung, dass es maximal vier Hierarchie-Ebenen gibt.

  D.h. e1 hat den direkten Vorgesetzten e2, der wiederum hat den direkten Vorgesetzten e3, und dieser hat schließlich den Vorgesetzten e4.

chef	ang	diff	path
FORD	SMITH	1	FORD|SMITH
⋮	⋮	⋮	⋮
JONES	SMITH	2	JONES|FORD|SMITH
⋮	⋮	⋮	⋮
KING	SMITH	3	KING|JONES|FORD|SMITH
KING	ADAMS	3	KING|JONES|SCOTT|ADAMS

Hausaufgabe 10b (2)

```
SELECT e2.ename chef, e1.ename ang, 1 diff,
       e2.ename || '|' || e1.ename path
FROM   emp e1, emp e2
WHERE  e1.mgr = e2.empno
UNION  ALL -- Zwei Hierarchie-Ebenen Abstand:
SELECT e3.ename chef, e1.ename ang, 2 diff,
       e3.ename || '|' || e2.ename || '|' ||
           e1.ename path
FROM   emp e1, emp e2, emp e3
WHERE  e1.mgr = e2.empno AND e2.mgr = e3.empno
UNION  ALL -- Drei Hierarchie-Ebenen Abstand:
SELECT e4.ename chef, e1.ename ang, 3 diff,
       e4.ename || '|' || e3.ename || '|' ||
           e2.ename || '|' || e1.ename path
FROM   emp e1, emp e2, emp e3, emp e4
WHERE  e1.mgr = e2.empno AND e2.mgr = e3.empno
       AND e3.mgr = e4.empno
```

Hausaufgabe 10b (3)

- Lösung mit Teilanfragen (Anfang):

```
WITH
eins AS
     (SELECT e2.ename chef, e1.ename ang, 1 diff,
             e2.ename || '|' || e1.ename
                 AS path
      FROM   emp e1, emp e2
      WHERE  e1.mgr = e2.empno),
zwei AS
     (SELECT ueber.chef, unter.ang, 2 diff,
             ueber.chef || '|' || unter.path
                 AS path
      FROM   eins unter, eins ueber
      WHERE  unter.chef = ueber.ang),
```

Hausaufgabe 10b (4)

- Lösung mit Teilanfragen (Fortsetzung):

```
drei   AS
       (SELECT ueber.chef, unter.ang, 3 diff,
               ueber.chef || '|' || unter.path
                   AS path
        FROM   zwei unter, eins ueber
        WHERE  unter.chef = ueber.ang)

SELECT * FROM eins
UNION  ALL
SELECT * FROM zwei
UNION  ALL
SELECT * FROM drei
```

Hausaufgabe 10b (5)

- Lösung mit Rekursion (erst in „DB-Programmierung“):

```
WITH    RECURSIVE
eins AS (SELECT e2.ename chef, e1.ename ang,
                1 AS diff,
                e2.ename|| '|' ||e1.ename AS path
         FROM   emp e1, emp e2
         WHERE  e1.mgr = e2.empno),
alle AS (SELECT * FROM eins
         UNION  ALL
         SELECT ueber.chef, unter.ang,
                unter.diff + 1 AS diff,
                ueber.chef || '|' || unter.path
                    AS path
         FROM   alle unter, eins ueber
         WHERE  unter.chef = ueber.ang)
SELECT  * FROM alle
```

Hausaufgabe 10c (1)

- Die folgende Anfrage ist in relationaler Algebra zu schreiben. Sie bezieht sich auf die Beispiel-Datenbank der Vorlesung:
  - STUDENTEN(SID, VORNAME, NACHNAME, EMAIL^o)
  - AUFGABEN(ATYP, ANR, THEMA, MAXPT)
  - BEWERTUNGEN(SID$\rightarrow$STUDENTEN, (ATYP, ANR)$\rightarrow$AUFGABEN, PUNKTE)
- Welche Studierenden haben für irgendeine Aufgabe die volle Punktzahl bekommen?

  Geben Sie Vorname und Nachname des Studierenden sowie Aufgabentyp und Aufgabennummer aus.

VORNAME	NACHNAME	ATYP	ANR
'Lisa'	'Weiss'	'H'	1

RelaX druckt zu den Spaltennamen noch einen Relationen-Präfix.

Hausaufgabe 10c (2)

- Lösung mit natürlichem Verbund („natural join“):

```
pi VORNAME, NACHNAME, ATYP, ANR (
    sigma PUNKTE = MAXPT (
        STUDENTEN join BEWERTUNGEN join AUFGABEN))
```

Man kann das in RelaX testen:

[http://dbis-uibk.github.io/relax/calc/gist/8dc2652578ee12ae756a234c4cf21b3f]

- Der natürliche Verbund funktioniert, weil
  - Die Spalte `SID` die einzige gemeinsame Spalte zwischen `STUDENTEN` und `BEWERTUNGEN` ist, und
  - die Spalten `ATYP` und `ANR` die einzigen gemeinsamen Spalten zwischen dem Ergebnis dieses Verbundes und `AUFGABEN` sind.

RelaX klammert den Verbund implizit von links.

Hausaufgabe 10c (3)

- Die Anordnung der Tabellen in einem mehrfachen natürlichem Verbund spielt im Prinzip keine Rolle (ggf. Punktabzug wegen Stil/schlechter Performance):

```
pi VORNAME, NACHNAME, ATYP, ANR (
    sigma PUNKTE = MAXPT (
        STUDENTEN join AUFGABEN join BEWERTUNGEN))
```

- Zunächst werden `STUDENTEN` und `AUFGABEN` mit dem natürlichen Verbund verknüpft. Da sie keine gemeinsamen Spalten haben, ist dies einfach das kartesische Produkt.

  Man kann auch „`x`“ statt dem linken „`join`“ schreiben.

- Anschließend verknüpft der rechte Join nur Tupel mit gleichen Werten in `SID`, `ATYP` und `ANR`.

  Wenn ein Student eine Aufgabe nicht bearbeitet hat, findet das entsprechende Tupel aus dem linken „Join“ keinen Join-Parner in `BEWERTUNGEN`.

Hausaufgabe 10c (4)

- Lösung mit expliziten Verbund-Bedingungen und explizitem Präfix für die Attribute:

```
pi S.VORNAME, S.NACHNAME, A.ATYP, A.ANR (
    sigma B.PUNKTE = A.MAXPT (
        rho S STUDENTEN
            join S.SID = B.SID
        rho B BEWERTUNGEN
            join B.ATYP=A.ATYP and B.ANR=A.ANR
        rho A AUFGABEN))
```

- Dies funktioniert in RelaX und ist nach dem Skript korrekt.

  Die Kompatibilität mit dem Skript erfordert insbesondere auch, dass nach der Umbenennung der Spalten mit dem ρ-Operator die Spalten immer mit dem Präfix angesprochen werden.

Hausaufgabe 10c (5)

- Die folgende Version mit expliziter Verbund-Bedingung funktioniert in RelaX, wäre nach dem Skript aber inkorrekt:

```
pi VORNAME,NACHNAME,AUFGABEN.ATYP,AUFGABEN.ANR (
    sigma PUNKTE = MAXPT (
        STUDENTEN
            join STUDENTEN.SID = BEWERTUNGEN.SID
        BEWERTUNGEN
            join BEWERTUNGEN.ATYP = AUFGABEN.ATYP
                and BEWERTUNGEN.ANR=AUFGABEN.ANR
        AUFGABEN))
```

- Im Skript gibt es keinen automatischen Relationenpräfix.

  Den Join mit expliziter Bedingung darf man nur verwenden, wenn die Eingaberelationen disjunkte Mengen von Spaltennamen haben.

Inhalt

Präsenzaufgabe: UNION oder Outer Join (1)

Schreiben Sie folgende Anfrage in SQL (3 Punkte):

- Geben Sie alle Bundesstaaten aus zusammen mit der Anzahl Präsidenten, die in dem jeweiligen Staat geboren wurden.
- Dabei sollen auch Staaten mit 0 Präsidenten gelistet werden.

  Nennen Sie die Spalte mit der Anzahl `num_pres` und sortieren Sie die Ausgabe absteigend nach dieser Anzahl, bei gleicher Anzahl nach dem Staat-Namen.

state_name	num_pres
Virginia	8
⋮	⋮
Wyoming	0

Bitte keine Lösungen mit Unteranfragen unter SELECT!

- president(pres_name, birth_year, years_serv, death_age, party, state_born→state)
- state(state_name, admin_entered, year_entered)

Präsenzaufgabe: UNION oder Outer Join (2)

- Lösung mit Outer Join (nützliches Muster!):

```
SELECT s.state_name,
       COUNT(p.pres_name) AS num_pres
FROM   state s left join president p
       on s.state_name = p.state_born
GROUP  BY s.state_name
ORDER  BY num_pres DESC, s.state_name
```

- Beim `COUNT` kann man irgendein Attribut der Tupelvariable `p` nehmen, das als `NOT NULL` deklariert ist:
  - Wenn `p` ein Tupel der Tabelle `president` ist, ist `p.pres_name` nicht `NULL` und wird gezählt.
  - Wenn der aktuelle Staat dagegen keinen Präsidenten hat, ist `p` ein Tupel aus lauter Nullwerten, insbesondere ist `p.pres_name` `NULL`, und wird nicht gezählt.

Präsenzaufgabe: UNION oder Outer Join (3)

- Lösung mit Unteranfrage unter SELECT (war nicht erlaubt):

```
SELECT s.state_name,
       (SELECT COUNT(*) AS num_pres
        FROM   president p
        WHERE  p.state_born = s.state_name)
FROM   state s
ORDER  BY num_pres DESC, s.state_name
```

- Die Unteranfrage wird (zumindest konzeptionell) ein Mal für jede Belegung der Tupelvariable `s` ausgeführt.
- Den Spaltennamen `num_pres` kann man natürlich auch außerhalb der Unteranfrage definieren:

```
SELECT s.state_name,
       (SELECT COUNT(*) FROM ...) num_pres
```

Präsenzaufgabe: UNION oder Outer Join (4)

- Lösung mit UNION:

```
SELECT s.state_name, COUNT(*) AS num_pres
FROM   state s, president p
WHERE  s.state_name = p.state_born
GROUP  BY s.state_name
UNION  ALL
SELECT s.state_name, 0 AS num_pres
FROM   state s
WHERE  s.state_name NOT IN (SELECT state_born
                            FROM   president)
ORDER  BY num_pres DESC, state_name
```

Die erste Teilanfrage behandelt Staaten mit mindestens einem Präsidenten, die zweite Teilanfrage Staaten ohne Präsidenten.

Inhalt

UNION und ORDER BY (1)

- Die folgende kleine Variante der obigen Anfrage funktioniert nicht:

```
SELECT s.state_name, COUNT(*) AS num_pres
FROM   state s, president p
WHERE  s.state_name = p.state_born
GROUP  BY s.state_name
UNION  ALL
SELECT s.state_name, 0 AS num_pres
FROM   state s
WHERE  s.state_name NOT IN (SELECT state_born
                            FROM   president)
ORDER  BY num_pres DESC, s.state_name
```

```
ERROR: missing FROM-clause entry for table "s"
LINE 9: ORDER BY num_pres DESC, s.state_name
```

UNION und ORDER BY (2)

- Die Tupelvariablen sind nur innerhalb jedes `SELECT`-Blocks bekannt.
- Obwohl `s.state_name` hier in beiden Teilen den Wert der ersten Ausgabe-Spalte definiert, ist der Name der Ausgabe-Spalte nur „`state_name`“, nicht „`s.state_name`“.
- Im Beispiel gibt es effektiv ja auch zwei verschiedene Tupel-Variablen, die zufällig beide „`s`“ heißen: Auf welche sollte sich die Angabe beziehen?
- Nach einem `UNION` oder `UNION ALL` kann man also nur auf die dann noch übrigen Spalten zugreifen.

  Das gleiche gilt natürlich auch für die anderen Mengenoperationen und auch für `DISTINCT`.

UNION und ORDER BY (3)

- Das ist insofern verwirrend, als man bei einfachen Anfragen (zumindest bei PostgreSQL) auch nach Spalten sortieren kann, die man nicht ausgibt:

```
SELECT p.pres_name
FROM   president p
WHERE  p.years_serv >= 8
ORDER  BY p.birth_year
```

- In diesem Fall muss man die `ORDER BY`-Klausel wohl als Teil des `SELECT`-Blocks sehen.
- Man hat immer die Möglichlichkeit, eine `WITH`-Hilfsanfrage mit zusätzlichen Spalten zur Sortierung zu machen, und diese Spalten dann bei der Hauptanfrage nicht auszugeben (aber zur Sortierung zu verwenden).

UNION und ORDER BY (4)

- Es ist ein Syntaxfehler, in jeder Teilanfrage eine `ORDER BY`-Klausel zu schreiben:

```
SELECT s.state_name, COUNT(*) AS num_pres
FROM   state s, president p
WHERE  s.state_name = p.state_born
GROUP  BY s.state_name
ORDER  BY num_pres DESC, state_name
UNION  ALL
SELECT s.state_name, 0 AS num_pres
FROM   state s
WHERE  s.state_name NOT IN (SELECT state_born
                            FROM   president)
ORDER  BY num_pres DESC, state_name
```

```
ERROR: syntax error at or near "UNION", LINE 6: UNION ALL
```

UNION und ORDER BY (5)

- Wenn man die beide Teilanfragen in Klammern setzt, gibt es keinen Syntaxfehler mehr, aber aber eine Sortierung vor `UNION` bleibt nach dem `UNION` nicht notwendigerweise erhalten.
- Wenn man nur die erste Teilanfrage in Klammern setzt, gibt es auch keinen Syntaxfehler, und die `ORDER BY`-Klausel am Ende gilt für das Ergebnis der Vereinigung:
  - Dann bekommt man also die richtige Sortierung, aber die `ORDER BY`-Klausel in der geklammerten ersten Teilanfrage ist nutzlos.
  - `UNION` bindet also stärker (hat höhere Priorität) als `ORDER BY`.

Im Vergleich ist `UNION` wie Punktrechnung, `ORDER BY` wie Strichrechnung.

UNION und ORDER BY (6)

- Auch nach Duplikateliminierung kann man nur noch auf die explizit ausgegebenen Spalten zugreifen:

```
SELECT DISTINCT p.pres_name
FROM   president p
WHERE  p.years_serv >= 8
ORDER  BY p.birth_year
```

```
ERROR: for SELECT DISTINCT, ORDER BY expressions must appear
in select list.     LINE 4: ORDER BY p.birth_year
```

- Das ist insofern logisch, als es im allgemeinen für eine Ausgabezeile nach dem `DISTINCT` mehrere verschiedene Werte der übrigen („wegprojizierten“) Spalten geben könnte.
- Im Beispiel bewirkt die Duplikateliminierung allerdings nichts, da ein Schlüssel ausgegeben wird.

Inhalt

Phasen des DB-Entwurfs (1)

- Was sind die drei klassischen Phasen des DB-Entwurfs?

A. ER-Entwurf,
relationaler Entwurf,
`CREATE TABLE` Statements.

B. Konzeptioneller Entwurf,
Logischer Entwurf,
Physischer Entwurf

C. Anforderungs-Analyse,
Entwurf,
Implementierung

D. Inception,
Elaboration,
Construction

Phasen des DB-Entwurfs (2)

- Man sollte Datenbank-Schemata (mit mehr als zwei Tabellen) zunächst im ER-Modell entwerfen, und dann ins relationale Modell übersetzen.
- Welchen Grund finden Sie am überzeugensten?

A. Das ER-Modell hat eine übersichtliche graphische Notation.

   Auch etwas kompakter als eine graphische Darstellung eines relationalen Schemas mit Kanten für Fremdschlüssel.

B. Man ist nicht gleich auf ein bestimmtes DBMS festgelegt.

   Nicht einmal auf ein relationales DBMS. Blick auf reale Welt, nicht DB.

C. Es gibt Konstrukte, die das relationale Modell nicht hat.

   Z.B. Subklassen, aber erst in DB-Entwurfsvorlesung im Master.

D. Ein objektorientierter Entwurf ist heute Standard.

E. Mich überzeugt keiner dieser Gründe.

UML Klassendiagramme (1)

- Kennen Sie UML Klassendiagramme?
  - A. Ja, gut.
  - B. Es war dran, aber ich erinnere mich nur verschwommen.
  - C. Ich habe davon gehört, aber es war nicht oder nur extrem kurz dran.
  - D. Nie gehört.
- Kennen Sie ER-Diagramme?
  - A. Ja, gut.
  - B. Etwas.
  - C. Nein.
- Wenn ja, woher? Welche Notation?

UML Klassendiagramme (2)

ER-Diagramm in Barker-Notation:

UML Klassendiagramm:

UML Klassendiagramme (3)

- Was haben UML Klassendiagramme, was das ER-Modell nicht hat?
  A. Attribute
  B. Methoden
  C. Schlüssel
  D. Relationships mit Kardinalitätsangaben (ggf. umbenannt)
- Was hat das ER-Modell, was UML Klassendiagramme (ohne Erweiterungen) nicht haben?
  A. Attribute
  B. Methoden
  C. Schlüssel

Zeichnen von ER-Diagrammen

- Handschriftlich, abfotographieren/scannen.
- Graphviz: Open Source Graph Visualization Software
  - Homepage: [http://www.graphviz.org/]
  - DOT Language: [https://graphviz.org/doc/info/lang.html]
  - User's Manual: [https://www.graphviz.org/pdf/dotguide.pdf]
  - Online Schnittstelle z.B.: [http://graphviz.it/]
  - Erstes Beispiel: [http://www.informatik.uni-halle.de/~brass/db20/lab/dozent.dot]
- Oracle SQL Developer Data Modeler: [https://www.oracle.com/de/database/technologies/appdev/datamodeler.html]

DOT Spezifikation für Graphviz (1)

```
strict digraph {
nodesep=1
splines=false
rankdir=LR

DOZENT [shape=rect style=rounded label=<
<table border="0" cellborder="0" cellpadding="0">
<tr><td align="center" cellpadding="5" colspan="2"
>DOZENT</td></tr>
<tr><td>#</td><td align="left">Name</td></tr>
<tr><td>*</td><td align="left">Tel</td></tr>
</table>
>];
```

DOT Spezifikation für Graphviz (2)

```
VORLESUNG [shape=rect style=rounded label=<
<table border="0" cellborder="0" cellpadding="0">
<tr><td align="center" cellpadding="5" colspan="2"
>VORLESUNG</td></tr>
<tr><td>#</td><td align="left">Nr</td></tr>
<tr><td>*</td><td align="left">Titel</td></tr>
</table>
>];
```

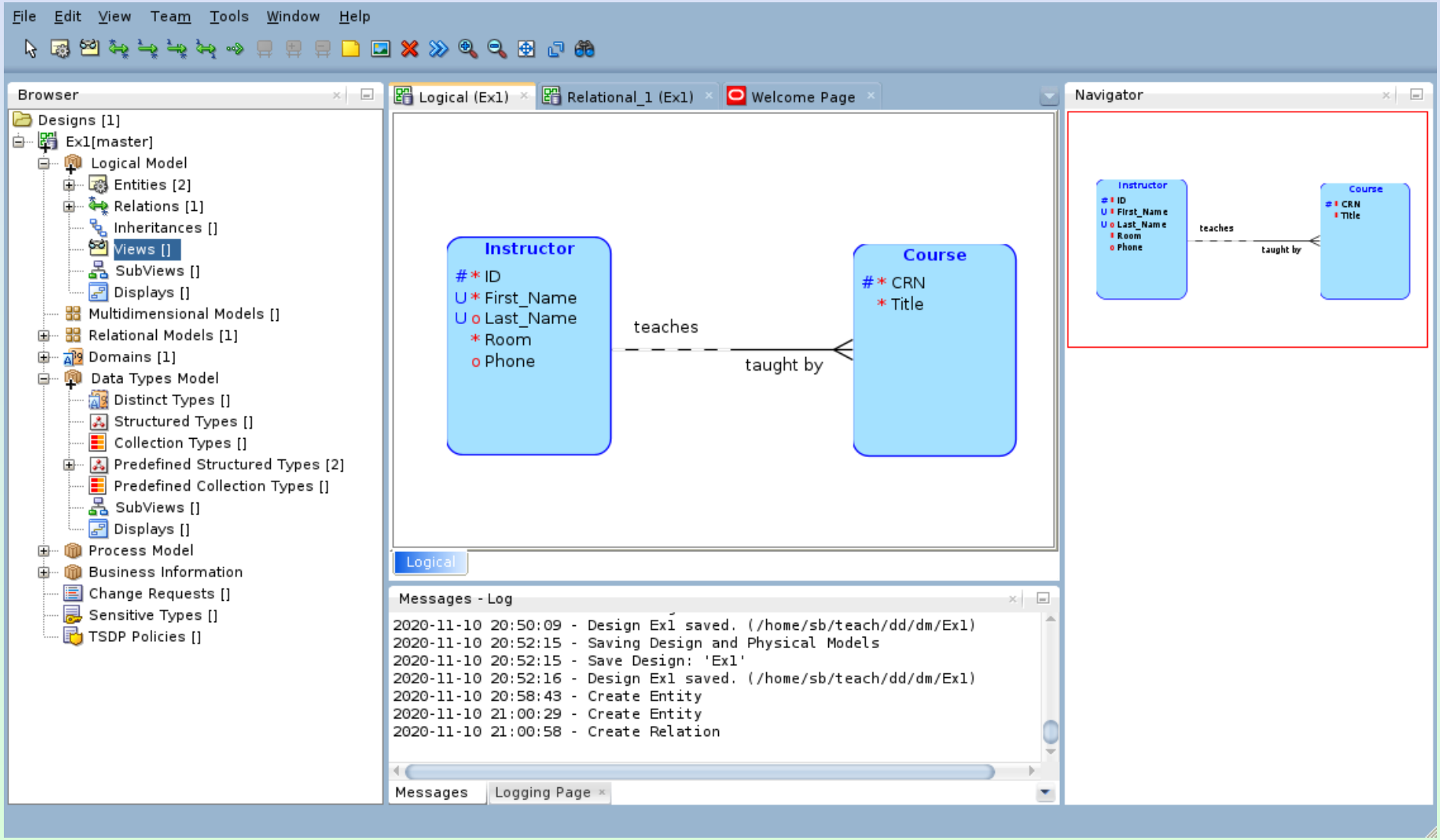
DOT Spezifikation für Graphviz (3)

```
node[shape=point height=0 color=black] mitte

DOZENT -> mitte [arrowhead=none style=dashed
                 label="hält"]
mitte -> VORLESUNG [arrowhead=crow style=solid
                    label="wird gehalten\nvon"]

}
```

Oracle SQL Developer Data Modeler



Zeichnen von Diagrammen: Weitere Möglichkeiten

- [draw.io] → [https://app.diagrams.net/]

  Schauen Sie sich auch die geschwungenen Kurven in der Palette für ER-Diagramme an. Dort verbergen sich passende Linien für Relationships.

- yEd: [https://www.yworks.com/products/yed]

- LaTeX pur oder mit Ti*k*Z

  TikZ&PGF Manual: [https://texdoc.org/serve/tikz/0]

  Claudio Fiandrino: „Drawing ER diagrams with TikZ“ (leider andere Notation): [https://www.guitex.org/home/images/ArsTeXnica/AT015/drawing-ER-diagrams-with-TikZ.pdf]

  Beispiel: [https://texample.net/tikz/examples/entity-relationship-diagram/]

- Dia: [http://dia-installer.de/]

  [https://sourceforge.net/projects/dia-installer/]

Inhalt

Check-Constraints (1)

- Die Bedingung in CHECK-Constraints wird vom DBMS überwacht:

```
CREATE TABLE AUFGABEN(
                ATYP    CHAR(1)      NOT NULL,
                ANR     NUMERIC(2)   NOT NULL,
                THEMA   VARCHAR(30)  NOT NULL,
                MAXPT   NUMERIC(3)   NOT NULL,
                PRIMARY KEY(ATYP, ANR),
                CHECK(ANR > 0),
                CHECK(ATYP IN ('H', 'Z', 'E')),
                CHECK(MAXPT >= 0))
```

- Man kann die Integritätsbedingung benennen:

```
CONSTRAINT ANR_MUSS_POSITIV_SEIN CHECK(ANR > 0)
```

Check-Constraints (2)

- Im `CREATE TABLE`: `CHECK(MAXPT >= 0)`.
- Diese Anweisung gibt dann einen Fehler:

```
INSERT INTO AUFGABEN
VALUES ('H', 5, 'Constraints', -1)
```

- Angenommen, `MAXPT` würde Nullwerte erlauben.
  Was passiert beim Einfügen einer Zeile mit Nullwert?
  A. Es gibt eine Fehlermeldung.
  B. Die Zeile wird eingefügt.
- Was kann man nicht in `CHECK`-Constraints verwenden?
  Kreuzen Sie die erste richtige Antwort an:
  A. `AND`, `OR`, `NOT`.
  B. `EXISTS`
  C. `CASE`

Überwachung von IBen mit Anfragen (1)

- "Die für eine Aufgabe vergebenen Punkte dürfen nicht größer sein als die Maximal-Punktzahl für die Aufgabe."

  Wenn man so eine Bedingung im Rahmen des Datenbank-Entwurfs aufstellt, sollte man darüber nachdenken, ob vielleicht ein Zusatzpunkt möglich wäre. Oder auch mehrere? Im Folgenden sei die Bedingung aber so gegeben.

- Man kann ein Skript mit SQL-Anfragen schreiben, um z.B. jede Woche nach Integritäts-Verletzungen zu suchen:

```
SELECT 'Zu viele Punkte für ' ||
       S.VORNAME || ' ' || S.NACHNAME ||
       ' in Aufgabe ' || A.ATYP || '-' || A.ANR'
FROM   STUDENTEN S, BEWERTUNGEN B, AUFGABEN A
WHERE  S.SID = B.SID
AND    B.ATYP = A.ATYP AND B.ANR = A.ANR
AND    B.PUNKTE > A.MAXPT
```

Überwachung von IBen mit Anfragen (2)

- Das ist zumindest eine formale Spezifikation der Bedingung.

  Es ist also eindeutig definiert, was genau die Integritätsbedingung ist. Es gibt keinen Interpretationsspielraum.

- Allerdings werden Verletzungen nicht beim Einfügen der Daten gefunden, sondern irgendwann später.

  Die verkehrten Daten könnten dann schon benutzt worden sein, um die Studienleistung zu bestätigen. Außerdem kann man den Fehler leichter korrigieren, wenn man noch dabei ist, die Daten einzugeben. Dann weiß man ja, was man tun will.

  In der Vorlesung „Datenbank-Programmierung“ werden für dieses Beispiel Trigger entwickelt (sie laufen automatisch beim Einfügen).

- Die obige Anfrage liefert ein leeres Ergebnis, wenn alles in Ordnung ist. Die Anfrage berechnet ja Fehlermeldungen.

  Wenn das Ergebnis leer ist, gibt es keine Verletzungen der Integritätsbedingung.

Überwachung von IBen mit Anfragen (3)

- Wenn man statt der Fehlermeldung umgekehrt ein „ok“ haben möchte, muss man fordern, dass die obige Anfrage ein leeres Ergebnis liefert:

```
WITH   FEHLER1 AS (SELECT 'Zu viele Punkte' ...)
SELECT 'ok'
WHERE  NOT EXISTS (SELECT * FROM FEHLER1)
```

  In PostgreSQL ist es möglich, eine Anfrage mit `SELECT` und `WHERE`, aber ohne `FROM` zu schreiben: Es gibt dann 0 oder 1 Ausgabezeilen, je nachdem, ob die `WHERE`-Bedingung erfüllt ist. Formal gibt es in diesem Fall also genau eine Variablenbelegung (für die leere Menge von Variablen).
  Bei MySQL könnte man z.B. schreiben: „`FROM (SELECT 1) X`“.
  Oracle hat eine Systemtabelle `DUAL`, die garantiert genau eine Zeile hat.

- Ziel der Präsenzaufgabe ist eine Anfrage, die Fehlermeldungen für eine IB berechnet, aber im positiven Fall kein leeres Ergebnis liefert, sondern „`ok`“ (also obige Anfragen verbindet).

Inhalt

Integritätsüberwachung mit Anfragen

Schreiben Sie folgende Anfrage in SQL (3 Punkte):

- Es soll die Integritätsbedingung überwacht werden, dass kein Angestellter mehr als sein direkter Vorgesetzter verdient.
- `emp(empno, ename, job, mgr°→emp, hiredate, sal, comm°, deptno°→dept)`
- Schreiben Sie eine Anfrage, die Fehlermeldungen folgender Form liefert, wenn die Bedingung verletzt wird:

errmsg
SCOTT verdient 3000, sein Vorgesetzter JONES nur 2975
FORD verdient 3000, sein Vorgesetzter JONES nur 2975

- Falls es keine Interitätsverletzungen gibt, soll die Anfrage eine Zeile mit dem Wert „`ok`“ in der Spalte „`errmsg`“ liefern.

Zum Test können Sie die IB modifizieren: Max. 50 $ mehr erlaubt.