Dr. Stefan Brass
Institut für Informatik
Universität Giessen

2. Juli 2002

Introduction to Databases (Summer 2002)
— Exam Solution —

The exam was written by 10 students. There were 90 minutes time.

Exercise	Correct	Small Error	Big Error	Not Done	Avg Points
1a	9	1	0	0	2.95/3 [98%]
1b	6	3	1	0	2.65/3 [88%]
1c	6	2	2	0	2.63/3 [87%]
1d	6	1	3	0	2.50/3 [83%]
1e	4	3	3	0	2.60/3 [87%]
1f	0	2	8	0	1.88/3 [62%]
2	9	1	0	0	5.95/6 [99%]
3	4	1	5	0	5.35/6 [89%]
4	0	1	7	2	2.35/4 [59%]

Example Database

An German online shop for DVDs (video films) uses the following three tables:

- The first table contains the most important data of the DVDs. Each DVD has a unique number. The database does not contain any null values.

DVD			
NO	TITLE	CATEGORY	PRICE
1	Snow White and the Seven Dwarfs	Animated Cartoon	27.99
2	Hot Shots! Part Deux	Comedy	25.99
3	Die Feuerzangenbowle (Fire Tongs Punch)	Comedy	24.99
4	Spaceballs	Comedy	19.99

- In contrast to VHS video tapes, DVDs often contain several audio tracks in different languages. The second table contains this information. `NO` is a foreign key that refers to `DVD`. Because this shop is in Germany, all DVDs in the database actually have a German audio track.

AUDIOTRACK	
NO	LANGUAGE
1	German
1	English
2	German
2	English
2	Spanisch
3	German
4	German
4	English
4	French
4	Italian

- Finally, there is a table with evaluations of the DVDs by customers or users of the website. A user can rate a DVD by giving between one and five stars. `NO` is again a foreign key that refers to the table `DVD`.

EVALUATION			
NO	UID	STARS	TEXT
1	sb	4	sweet
1	Nina	3	nice
1	Lisa	5	I watched it already 50 times.
2	sb	4	funny
3	sb	3	funny, but picture and sound quality not good.
4	sb	5	very funny
4	Nina	3	well, ok

Exercise 1 (SQL Queries) 18 Points

a) Print title and price of all DVDs in the category “Comedy”, which cost less than \$ 25. Sort the output by the price (lowest price first). In the example, the query result should look as follows:

TITLE	PRICE
Spaceballs	19.99
Die Feuerzangenbowle (Fire Tongs Punch)	24.99

- A correct solution is:

```
SELECT   TITLE, PRICE
FROM     DVD
WHERE    CATEGORY = 'Comedy' AND PRICE < 25
ORDER BY PRICE
```

- Of course, outside quotes, the case is not important. Three students used all lowercase, three students all uppercase, two wrote keywords in lowercase and column names in uppercase, and two used the natural German case: “`Select Title, Price ...`”.
- One can also write “`ORDER BY PRICE ASC`” (three students did that), but “`ASC`” is default.
- All solutions were correct, however one student wrote all string constants in the form `‚Comedy'` (with two different quotes). I took half a point off for this.

b) Print number and title of all films (DVDs) that have an English audio track and got 5 stars from the user "sb". In the example, the result should look as follows:

NO	TITLE
4	Spaceballs

- A correct solution is:

```
SELECT E.NO, TITLE
FROM   EVALUATION E, AUDIOTRACK A, DVD D
WHERE  D.NO = E.NO AND E.NO = A.NO
AND    UID = 'sb'
AND    LANGUAGE = 'English'
AND    STARS = 5
```

  Note that the tuple variable in `E.NO` (in the `SELECT`-clause) is necessary, although `A.NO` and `D.NO` would be equally correct. One student forgot the tuple variable.

- Of course, the names and sequence of the tuple variables declared under `FROM` is not important.

- In the above example, attribute references use tuple variables only when necessary (because the reference would otherwise not be unique). However, one can also use always tuple variables if this is clearer. Six students always used a tuple variable in attribute references, one only when necessary, one had an ambiguous reference (tuple variable missing), two had mixtures.

```
SELECT D.NO, D.TITLE
FROM   DVD D, AUDIOTRACK A, EVALUATION E
WHERE  D.NO = A.NO AND D.NO = E.NO
AND    A.LANGUAGE = 'English'
AND    E.UID = 'sb' AND E.STARS = 5
```

- One student used subqueries for the joins. I think that this looks unnecessarily complicated, but it is nevertheless correct:

```
SELECT NO, TITLE
FROM   DVD D
WHERE  EXISTS(SELECT * FROM AUDIOTRACK A
              WHERE  A.NO=D.NO
              AND    A.LANGUAGE = 'English')
AND    EXISTS(SELECT * FROM EVALUATION E
              WHERE  E.NO=D.NO
              AND    E.UID = 'sb' AND E.STARS=5)
```

- One student forgot the condition `UID = 'sb'`. I took half a point off for this.

- One student used a wrong tuple variable in an attribute reference: `D.UID = 'sb'`, where `D` is declared as `DVD D`. I took half a point off for this.

- One solution contained severe syntax errors:

```
SELECT D.NO, D.TITLE
FROM   DVD D, AUDIOTRACK A, (SELECT X.NO
                              FROM   EVALUATION X
                              WHERE  X.NO = D.NO
                              AND    X.UID = 'sb'
                              AND    X.STARS = 5) Y
WHERE  A.NO = D.NO
AND    A.LANGUAGE = 'English'
AND    D.NO IN Y
```

  It is not legal to formulate the subquery as in `D.NO IN Y`, where the subquery itself is declared under `FROM`. If the student had moved the subquery from the `FROM`-clause to the right hand side of `IN`, the query would have worked. However, the condition `X.NO = D.NO` in the subquery would be superfluous, since the join is already done via `IN`. At least Oracle also forbids accesses to tuple variables of the same `FROM`-clause in subqueries used under `FROM`.

c) Print the title of all DVDs with an audio track in English and in Spanisch (both languages on the same DVD). In the example, the result would look as follows:

TITLE
Hot Shots! Part Deux

- In this exercise, one needs two tuple variables over the relation AUDIOTRACK. A correct solution is:

```
SELECT TITLE
FROM   DVD D, AUDIOTRACK X, AUDIOTRACK Y
WHERE  X.NO = Y.NO AND D.NO = X.NO
AND    X.LANGUAGE = 'Spanisch'
AND    Y.LANGUAGE = 'English'
```

  Five students used a solution of this type.

- Four students used a solution with a subquery under FROM. One was correct, two of them contained small errors, one a severe error. The correct solution is:

```
SELECT X.NO, TITLE
FROM   (SELECT NO, TITLE
        FROM   DVD D, AUDIOTRACK EN
        WHERE  D.NO = EN.NO
        AND    EN.LANGUAGE = 'English') X,
       AUDIOTRACK SP
WHERE  X.NO = SP.NO
AND    SP.LANGUAGE = 'Spanisch'
```

  In my view, subqueries under FROM are a complication that should be avoided if possible. E.g. they are not contained in the SQL-86 standard, so older/smaller DBMS might not support them. Even if they are supported, they might be evaluated in a slightly less efficient way than the standard solution above.

  Of course, subqueriesunder FROM are similar to views. If one wants to use views to construct the query step-by-step, the greater clearity might be worth a slight efficiency penalty. Also, views were already contained in the SQL-86 standard.

- The next two students had nearly the same solution, a correct version of it is:

```
SELECT TITLE
FROM   DVD D,
       (SELECT A1.NO
        FROM   AUDIOTRACK A1, AUDIOTRACK A2
        WHERE  A1.NO = A2.NO
        AND    A1.LANGUAGE = 'English'
        AND    A2.LANGUAGE = 'Spanisch') X
WHERE  D.NO = X.NO
```

- One of the two students wrote in the subquery "`SELECT A1.NO, A1.LANGUAGE`". I took half a point off for an unnecessary complication: "`A1.LANGUAGE`" is not used in the outer query, and furthermore it is constant, it can only have the value "`English`". Such result columns are normally not useful.
- The other sudent wrote "`SELECT NO`" in the subquery. This is an ambiguous tuple variable reference. It is not clear whether "`A1.NO`" or "`A2.NO`" is meant (although both have the same value, SQL requires that one specifies a tuple variable in this case).
- One student had a query that only checked for Spanisch. Since the two tuple variables over the same relation was the main complication of this query, I took off 1.5 points (maybe I even should have taken off two points).

d) Which DVDs have only a German audio track? Since this is a German online shop, you can assume that all DVDs on sale have a German audio track. You only have to test that it contains no other language besides German. Please print the title of all such DVDs. In the example, the result is:

TITLE
Die Feuerzangenbowle (Fire Tongs Punch)

- I would have used the following solution, which seems simplest to me. However, only three students tried it this way, and only two of them got it right:

```
SELECT TITLE
FROM   DVD D
WHERE  NO NOT IN (SELECT NO
                  FROM   AUDIOTRACK
                  WHERE  LANGUAGE <> 'German')
```

- One of the two students wrote “DISTINCT” in the subquery. This is not necessary for the correctness of the query. The effect on the performance is unclear to me. Depending on the DBMS, the query might run faster or might run slower. In general, I would expect that a good optimizer automatically eliminates duplicates in “NOT IN” subqueries, if it thinks that this is advantageous. I would agree that current optimizers are less intelligent than one would expect, but still the “DISTINCT” there has more the effect of a hint to the optimizer (to be used by very experienced DB programmers in case of performance problems). In a first database course, one should use the simplest logically equivalent solution, which means that the “DISTINCT” should be left out.
- One student wrote “!=” for not equals. This works e.g. in Oracle, but it does not conform with the SQL standard. One should use “<>”. But I did not take any points off.
- Seven students tried to solve this task with an aggregation. Four had a correct solution, three of which used this version:

```
SELECT   TITLE
FROM     DVD D, AUDIOTRACK A
WHERE    A.NO = D.NO
GROUP BY A.NO, TITLE
HAVING   COUNT(*) = 1
```

- One of the incorrect solutions used only GROUP BY A.NO. Then it is a syntax error to list TITLE under SELECT. Although the TITLE actually is unique within the groups, SQL requires that all attributes used outside aggregations under SELECT must appear under GROUP BY. In order to understand that TITLE is unique, the SQL parser would have to analyze the WHERE-condition, find the A.NO = D.NO, and then get the information that `D.NO` is a key of `DVD`. When SQL was designed, this was probably considered as too difficult.

- The last correct solution used again a subquery under `FROM`. If one has understood how a hammer works, one is first temped to solve all tasks with a hammer, but this might not be the best solution:

```
SELECT TITLE
FROM   (SELECT   NO, COUNT(*) NUM_LANG
        FROM     AUDIOTRACK
        GROUP BY NO) X,
       DVD D
WHERE  D.NO = X.NO
AND    X.NUM_LANG = 1
```

  To be fair, if the subquery would be a view that already exists, it might be a good (or at least simple) solution to use this view. In general, the simplest/shortest solution is usually best.

- One wrong solution using an aggregation was:

```
SELECT TITLE
FROM   DVD
WHERE  NO IN (SELECT   NO
              FROM     AUDIOTRACK
              WHERE    LANGUAGE = 'German'
              GROUP BY NO
              HAVING   COUNT(*) = 1)
```

  Since `LANGUAGE` and `NO` together is a key of `AUDIOTRACK`, the `WHERE`-condition and the `GROUP BY` together imply that all groups consist of a single row, so the `HAVING` condition is automatically satisfied. The result will be that all DVDs are listed (assuming that all have a German audio track). Without the `WHERE`-condition, this solution would have worked.

  Unfortunately, it also contained a syntax error: The subquery selected the two reuslt columns `NO` and `LANGUAGE`. This is wrong for two reasons: If there is only one column on the left hand side of `IN` (the normal case), the subquery must have a single result column. `IN` tests whether the subquery has a result row that is equal to the left hand side. This works only if the two rows have the same number of columns (one). Second, one cannot list `LANGUAGE` there because it is not listed under `GROUP BY` (see above). Furthermore it would be wrong to add it to the `GROUP BY` clause, because then one has again groups consisting only of a single row each (one would group by a key).

  Finally, the solution contained an unnecessary join. If one corrects the other errors, the query looks as follows:

```
SELECT TITLE
FROM   DVD D, AUDIOTRACK A
WHERE  D.NO = A.NO
AND    D.NO IN (SELECT   NO
                FROM     AUDIOTRACK
                GROUP BY NO
                HAVING   COUNT(*) = 1)
```

The join `D.NO = A.NO` only ensures that there is an entry for the DVD in the relation `AUDIOTRACK`. However, if the `IN` condition is satisfied, this is automatically true. Therefore, the following query is provably equivalent:

```
SELECT TITLE
FROM   DVD
WHERE  NO IN (SELECT   NO
              FROM     AUDIOTRACK
              GROUP BY NO
              HAVING   COUNT(*) = 1)
```

Very probably this query will run slightly faster (e.g. the Oracle8 optimizer is not intelligent enough to discover the unnecessary join).

By the way, even without the `IN` condition, the join would be unnecessary if we have the integrity constraint that every DVD has at least one audio track. Above, it was stated that one can assume that there is a German audio track for every DVD in this store.

- A wrong solution using `NOT IN` was:

```
SELECT TITLE
FROM   DVD D, AUDIOTRACK A
WHERE  D.NO = A.NO
AND    LANGUAGE NOT IN (SELECT LANGUAGE
                        FROM   AUDIOTRACK
                        WHERE  LANGUAGE <> 'German')
```

This would give all DVDs that have an audio track in German, i.e. all DVDs in this store. The query does not check that there is no other audio track. The query is equivalent to

```
SELECT TITLE
FROM   DVD D, AUDIOTRACK A
WHERE  D.NO = A.NO
AND    LANGUAGE = 'German'
```

In general, if there is an equivalent query that is significantly simpler, this is an indication for an error (or at least some misconception).

The query also contained two syntax errors that were corrected above: First, the student wrote “`SELECT *`” in the subquery. But then it has two result columns and is not comparable with a single column on the left hand side, see above. Second, the student wrote "`not`" instead of "`<>`".

e) Print for each DVD that was evaluated by at least two customers the title, the category, and the average evaluation (number of stars). The output column for the average evaluation should have the name "GRADE". In the example, the result would be:

TITLE	CATEGORY	GRADE
Snow White and the Seven Dwarfs	Animated Cartoon	4
Spaceballs	Comedy	4

- Six students had basically the following solution, but only two got it completely correct:

```
SELECT   TITLE, CATEGORY, AVG(STARS) AS GRADE
FROM     DVD D, EVALUATION E
WHERE    D.NO = E.NO
GROUP BY D.NO, TITLE, CATEGORY
HAVING   COUNT(*) >= 2
```

  The keyword "AS" in the SELECT-list ist not required (it is only "syntactic sugar" or a "noise word"). Also, one can use double quotes around column names, e.g. write "GRADE". However, the quotes are only required if the column name contains otherwise illegal characters (like a space) or is not all uppercase.

- Three students used only:

```
GROUP BY TITLE, CATEGORY
```

  This solution is not quite correct, since there can be several films with the same title in the same category. Then the evaluations of both are merged, which is not very fair. E.g. in Germany there are two films "Snow White and the Seven Dwarfs", both in the category "Animated Cartoon" (the Disney classic and one other film). I took off half a point in this case. Probably I should have mentioned this explicitly in the exercise. But there are always remakes of classic films.

- One student used

```
GROUP BY B.NO
```

  Then it is illegal to use TITLE and CATEGORY in the SELECT clause (outside of aggregations).

- One student listed the HAVING-clause before the GROUP BY-clause. This is syntactically not correct. However, I was surprised to note that Oracle accepts this.

- One student wrote

```
HAVING COUNT(B.STARS) >= 2
```

  This is not wrong: It actually does not matter what one counts, unless there are null values or one elminiates duplicates. But I believe that it is not very good style to explicitly list an attribute if it does not matter which one. The reader starts thinking why this attribute was chosen. Also, with duplicate elimination, i.e. COUNT(DISTINCT B.STARS), it would not be correct. If a film gets 5 stars from two different users, it should of course be listed.

- Two students used a subquery under FROM. As mentioned before, I believe that this is unnecessarily complicated, but nevertheless, there solution was correct:

```
SELECT TITLE, CATEGORY, GRADE
FROM   (SELECT   NO, AVG(STARS) "GRADE"
        FROM     EVALUATION E
        GROUP BY NO
        HAVING   COUNT(*) >= 2) X,
       DVD D
WHERE  D.NO = X.NO
```

  At least, this makes the GROUP BY-clause slightly simpler.

- One additional student used a similar solution, but checked the HAVING-condition in the outer query:

```
SELECT D.TITLE, D.CATEGORY, X.EY GRADE
FROM   (SELECT   NO, COUNT(*) EX, AVG(E.STARS) EY
        FROM     EVALUATION E
        GROUP BY NO) X
       DVD D
WHERE  D.NO = X.NO
AND    X.EX >= 2
```

  Actually, the student did the join D.NO = X.NO inside the subquery, which is not correct.

- Finally, a fourth student also tried to use a subquery under FROM, but the solution contains several severe mistakes and seems incomplete:

```
SELECT TITLE, CATEGORY, AVG(X.M) "GRADE"
FROM   DVD,
       (SELECT   UID, COUNT(*), SUM(STARS)
        FROM     EVALUATION
        GROUP BY UID
        HAVING   COUNT(*) >= 2)
```

f) Please print a list of all DVDs that shows title and price plus an additional column "`ENGLISCH`", that contains "`X`", if the DVD contains an English audio track, and a space otherwise. The list should be sorted by price.

TITLE	PRICE	ENGLISH
Spaceballs	19.99	X
Die Feuerzangenbowle (Fire Tongs Punch)	24.99	
Hot Shots! Part Deux	25.99	X
Snow White and the Seven Dwarfs	27.99	X

- No student got this exercise completely right. Two students had only small mistakes. This exercise needs a `UNION` of two parts, one for the DVDs with English language audio track, and one for those without. Eight students discovered this and tried a solution with `UNION`. The following is a correct solution:

```
SELECT   TITLE, PRICE, 'X' ENGLISH
FROM     DVD D, AUDIOTRACK A
WHERE    D.NO = A.NO
AND      LANGUAGE = 'English'
UNION ALL
SELECT   TITLE, PRICE, ' ' ENGLISH
FROM     DVD D
WHERE    D.NO NO IN (SELECT NO
                     FROM   AUDIOTRACK
                     WHERE  LANGUAGE = 'English')
ORDER BY PRICE
```

- One can use `UNION` instead of `UNION ALL`, but that would be less efficient. Most DBMS will probably not notice that because the last column contains distinct values in the two queries, the operands to the `UNION` are disjoint. Then the DBMS will do a duplicate elimination, which is unnecessary, and can be avoided by writing `UNION ALL`. Seven students wrote `UNION ALL`, one student wrote `UNION`, and two students tried a solution without `UNION` (which was wrong).

- Of course one can also use `NOT EXISTS` in the second half:

```
...
UNION ALL
SELECT TITLE, PRICE, ' ' ENGLISH
FROM   DVD D
WHERE  NOT EXISTS (SELECT *
                   FROM   AUDIOTRACK A
                   WHERE  D.NO = A.NO
                   AND    LANGUAGE = 'English')
```

- One student did a join in the second half that was not required:

```
SELECT TITLE, PRICE, ' ' ENGLISH
FROM   DVD D, AUDIOTRACK A
WHERE  D.NO = A.NO
AND    D.NO NOT IN (SELECT NO
                    FROM   AUDIOTRACK
                    WHERE  AUDIOTRACK = 'English')
```

The join with `AUDIOTRACK A` in the main query is not required. Of course, it might make sense to exclude DVDs from the list that have no soundtrack entered into the database. However, in d) it is explicitly stated that one can assume that all DVDs have German sound. Then the join simply does nothing. Also the exercise did not mention that one should check that the DVD appears in `AUDIOTRACK`. I took off half a point in this case.

- One student tried the following in the second half:

```
SELECT TITLE, PRICE, ' ' ENGLISH
FROM   DVD D, AUDIOTRACK A
WHERE  D.NO = A.NO
AND    A.LANGUAGE <> 'English'
```

  This query returns all DVDs that contain an audio track in a language different from English. Since all DVDs in the database contain a German audio track, this are simply all DVDs. One can also understand that one needs `NOT IN`, `NOT EXISTS`, or maybe an aggregation by thinking about monotonic and nonmonotonic behaviour. This query behaves monotonically: If more rows are entered into the relation `AUDIOTRACK`, the old answers remain correct. Here one needs a constuct that behaves nonmonotonically: If there was no English audio track in the database, the DVD should be selected, but after one inserts such an audio track, the solution becomes wrong.

- The following query (used by three other students) has the same problem, but is significantly more complicated:

```
SELECT TITLE, PRICE, ' ' ENGLISH
FROM   DVD D, AUDIOTRACK A
WHERE  D.NO = A.NO
AND    A.LANGUAGE NOT IN (SELECT A.LANGUAGE
                          FROM   AUDIOTRACK A
                          WHERE  A.LANGUAGE = 'English')
```

  The subquery can actually return only the value `'English'`, and if the table contains at least one DVD with English language audio track, it will return this value. In general, I would say that using `NOT IN` with a subquery that can provably return only one value is at least strange (bad style).

- The following complicated version actually works (but is at least bad style):

```
SELECT TITLE, PRICE, ' ' ENGLISH
FROM   DVD D
WHERE  D.NO NOT IN (SELECT A.NO
                    FROM   AUDIOTRACK A
                    WHERE  A.LANGUAGE = 'English'
                    AND    A.NO = D.NO)
```

Because of the condition `A.NO = D.NO` in the subquery, the subquery can return only `D.NO`. So the result of the subquery can either be empty (if the DVD `D` has no Englisch audio track) or be the set consisting of the single element `D.NO` (if it has an English audio track). This is a bit strange for a `NOT IN` subquery. In general, correlated subqueries with `NOT IN` are normally bad style. In the example, the join condition is actually written wto times: Once in the subquery (`A.NO = D.NO`), and once with the `NOT IN`. This becomes even clearer when one translates the `NOT IN` into `NOT EXISTS`:

```
SELECT TITLE, PRICE, ' ' ENGLISH
FROM   DVD D
WHERE  NOT EXISTS
       (SELECT A.NO
        FROM   AUDIOTRACK A
        WHERE  A.LANGUAGE = 'English'
        AND    A.NO = D.NO -- given in the subquery
        AND    A.NO = D.NO -- Translation of NOT IN
       )
```

- One student forgot the `ORDER BY`. One student wrote the `ORDER BY`-clause in both parts of the query. That is a syntax error, `ORDER BY` is allowed only at the very end. It would also make little sense to order both parts separately.

Exercise 2 (SQL CREATE TABLE) 6 Points

Please write a CREATE TABLE statement for the table "EVALUATION".

- Declare the usual constraints (keys, foreign keys, NOT NULL).
- Use a CHECK-constraint to ensure that the attribute "STARS" can only take the values 1, 2, 3, 4, 5.
- The attribute NO is a decimal number of up to five digits. UID is a string up to length 20. TEXT is a string of maximal length 2000. You can assume that your DBMS supports variable-length strings of this size.
- Null values are not allowed in any column.

All students performed very well on this exercise. Only one student lost half a point, all other students had a completely correct solution. One correct solution is:

```
CREATE TABLE EVALUATION(
             NO    NUMERIC(5)    NOT NULL
                                 REFERENCES DVD,
             UID   VARCHAR(20)   NOT NULL,
             STARS NUMERIC(1)    NOT NULL,
             TEXT  VARCHAR(2000) NOT NULL,
             PRIMARY KEY(NO, UID),
             CONSTRAINT STARS_VALID
                   CHECK(STARS BETWEEN 1 AND 5))
```

- It was not required that the CHECK-constraint is given a name, so

  ```
  CONSTRAINT STARS_VALID
  ```

  can be left out. However, the DBMS will probably give better error messages with it. Seven students gave the constraint a name, three did not. The constraint names used were: "VALID_STARS", "STARS_RANGE", "STARS_OK", "STARS_VALUES", "STARS_VALUE", "CHECK_S", "STARS_EVALUATION". Not all names give equally understandable error messages.

- Of course, it is also possible to give the key and the foreign key names. Since the error messages for constraint violations are normally understandable, this is not very important. However, for later ALTER TABLE statements, it might be useful. Four students assigned a name to all constraints.

- Of course, the constraint condition can also be written e.g. as

  ```
  CHECK(STARS <= 5 AND STARS > 0)
  ```

Still another possibility is

```
CHECK(STARS IN (1,2,3,4,5)).
```

- Also, in the above solution, the `CHECK`-constraint was formulated as a table constraint. Since it refers only to a single attribute, it can also be formulated as a column constraint.
- Of course, one can write the foreign key also as table constraint:

```
FOREIGN KEY (NO) REFERENCES DVD.
```

- Instead of e.g. `NUMERIC(5)`, one can also write `NUMERIC(5,0)`.
- The only case where I took off half a point was when a student added a `CHECK`-constraint for the column `NO`:

```
CHECK(NO > 0).
```

  It is of course true that the DVD numbers must be greater than 0, but this is already tested in the `CREATE TABLE` statement for the relation `DVD`. Because of the foreign key, the table `EVALUATION` can contain only DVD numbers that appear also in `DVD`. Then it logically follows that `EVALUATION` contains only positive DVD numbers. The `CHECK` constraint in `EVALUATION` is therefore unnecessary. Since it decreases performance and makes modifications more difficult (e.g. when somebody decides that valid DVD numbers must not start with a 0), one should not write this constraint.

Exercise 3 (ER-Diagram) 6 Points

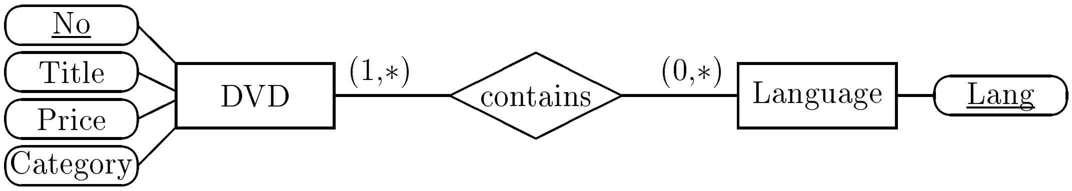
Draw an Entity-Relationship-Diagram for storing information about DVDs (number, title, price, category) and their audio tracks (available languages). I.e. the same information as in the first two example tables should be recorded in this database. In order to simplify the exercise, no evaluations (third table) have to be stored. Please define the key for each entity type. Also specify cardinalities for the relationship(s).

Each DVD must contain at least one language. If you want, you can permit to store languages or categories in the database that do not yet occur on any DVD. die noch auf keiner DVD vorhanden sind.

It is not required that if your ER-schema is translated back into the relational model, one gets exactly the two given tables (e.g. there could be one or two additional tables). You do not have to do the translation into the relational model (i.e. it suffices to draw an ER-diagram, `CREATE TABLE` statements are not required).

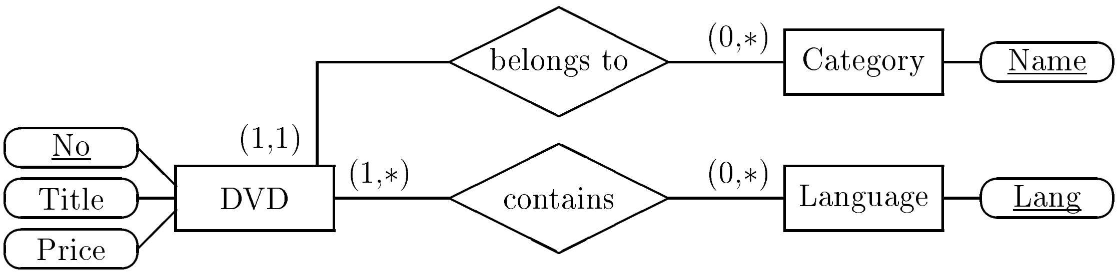
If it is not obvious what you meant with an attribute or another construct, please write a short explanation. Especially, an explanation would be important if you should modify the given information contents of the database.

- A correct solution is the following. It introduces "Language" as entity. The information of the table `AUDIOTRACK` corresponds then to the relationship "contains". The standard translation into the relational model would create an additional table "`LANGUAGES`" that lists all valid languages. This seems very reasonable.



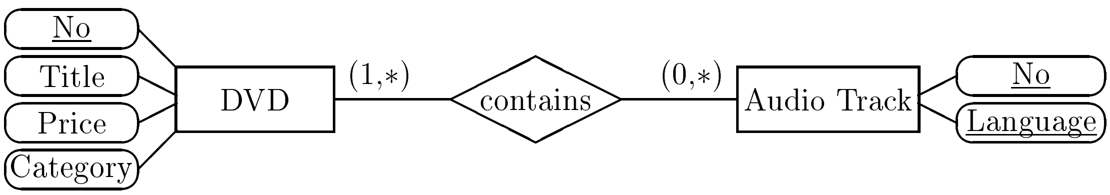
  Five students used this solution, of which only two were completely correct.

- In the same way, one could view the possibile categories as entities. Three students did this, of which two were completely correct.



One student added numbers as keys of Category and Language. This is possible, but means that later one will need more joins in the queries. E.g. the DVD table will then contain the category number as a foreign key referencing `CATEGORIES`. If one wants the category name, one must do a join with the table `CATEGORIES`. This will make the query more complicated and less efficient. However, since the `CATEGRIES` table is small and will soon be cached, the performance penality is probably not big.

- The following solution (used by two students) is incorrect:

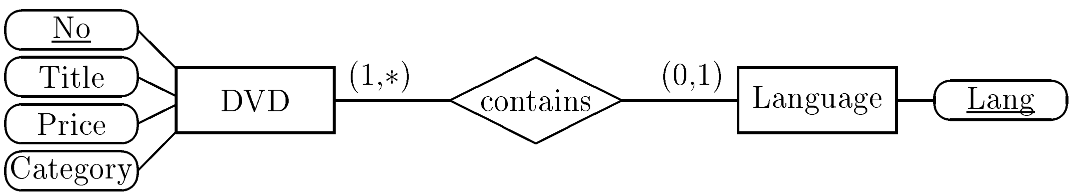


First, an "Audio Track" entity obviously belongs to a single DVD (identified by the DVD number that is part of the entity). Therefore, the correct cardinality of the relationship "contains" on the "Audio Track" side is "(1,1)".

Second, one needs in this case a constraint that if an DVD entity and an Audio Track entity are linked by the relationship "contains", their attributes "No" must have the same value. So the relationship would actually be redundant. Of course, in the ER-model, relationships are important, and one should not use attributes to reference other entities. It is a common error to use "foreign key" attributes in the ER-model (although one can declare any constraint, the ER-model has no special support for foreign keys). However, in this special case, "No" must be part of the key of "Audio Track" if one wants to map the given relation directly into the ER-model. Extended ER-models have the notion of a weak entity, which would have been the right solution in this case. Unfortunately, weak entities were not discussed in this course. I now believe that even a basic database course should treat them. It is also unfortunate that the exercise did not explicitly require constraints.

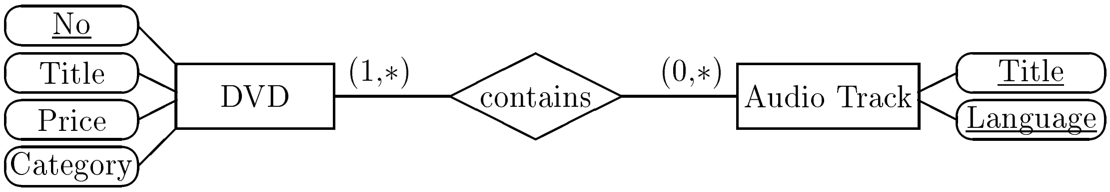
Probably it would also have been better if I had called the column "DNO" or "DVD-NO" instead of "NO". In one additional solution it was not clear to me whether an attribute "No" was a DVD-No or a Language-No.

- One student used the first solution with a wrong cardinality:



The same language can be on several DVDs, therefore the maximum cardinality 1 on the "Language" side is wrong.

- One student used a cardinality that permits one DVD to be contained in several categories. This might be reasonable (there are always mixtures of different categories of films), but the student would have had to explicitly state that he improved the given schema in this way. Without explanation, I counted it simply as a wrong cardinality.
- One student had this solution:



I did not understand the meaning of the attribute "Title" of the entity type "Audio Track". Is it the DVD title? That would be wrong, and it would also be not reasonable to use the title here when DVDs are otherwise identified by number.

Exercise 4 (XML) 4 Points

Encode the information about the first two DVDs in XML. Again, only the basic data and the audio track information is required, not the evaluations. You do not have to specify a DTD.

- A possible solution is:

```
<?xml version="1.0" ?>
<DVDLIST>
    <DVD no="1" title="Snow White and the Seven Dwarfs"
            category="Animated Cartoon" price="27.99">
        <AUDIOTRACK language="German" />
        <AUDIOTRACK language="English" />
    </DVD>
    <DVD no="2" title="Hot Shots! Part Deux"
            category="Comedy" price="25.99">
        <AUDIOTRACK language="German" />
        <AUDIOTRACK language="English" />
        <AUDIOTRACK language="Spanish" />
    </DVD>
</DVDLIST>
```

- Equally correct, but much longer is to use elements instead of attributes:

```
<?xml version="1.0" ?>
<DVDLIST>
    <DVD>
        <NO>1</NO>
        <TITLE>Snow White and the Seven Dwarfs</TITLE>
        <CATEGORY>Animated Cartoon</CATEGORY>
        <PRICE>27.99</PRICE>
        <AUDIOTRACK>German</AUDIOTRACK>
        <AUDIOTRACK>English</AUDIOTRACK>
    </DVD>
    <DVD>
        <NO>2</NO>
        <TITLE>Hot Shots! Part Deux</TITLE>
        <CATEGORY>Comedy</CATEGORY>
        <PRICE>25.99</PRICE>
        <AUDIOTRACK>German</AUDIOTRACK>
        <AUDIOTRACK>English</AUDIOTRACK>
        <AUDIOTRACK>Spanish</AUDIOTRACK>
    </DVD>
</DVDLIST>
```

- Only two students wrote the XML declaration. They got an extra half point. It is not a mistake to write no XML declaration, but it is better style to do it. One should also specify the character encoding there.

- Six students used no root element, i.e. they simply wrote the two DVD elements, but no enclosing container. I took off half a point for this error.

- One student wrote

```
<AUDIOTRACKS>German, English, Spanish</AUDIOTRACKS>
```

  This is not good and violates the spirit of XML because one must parse the string (search substrings) in order to find specific audio tracks. While the "sequence of words" structure as required here is probably supported in XML query languages, database people would try to use a markup that describes the complete structure. This corresponds to the idea that attribute values should be atomic in the relational model.

- Five students wrote something like

```
<AUDIOTRACKS>
    <A1>German</A1>
    <A2>English</A2>
    <A3>Spanish</A3>
</AUDIOTRACKS>
```

  Normally, element types such as `A1`, `A2`, `A3` must be declared in a DTD. In this case, it is not clear (at least not to me) what is a reasonable limit for the number of audio tracks on a DVD, i.e. how many of these elements one should declare. Also, queries will be complicated, because if will often be necessary to test each of these elements.

- One of the five students wrote:

```
<AUDIOTRACKS>
    <1>German</1>
    <2>English</2>
    <3>Spanish</3>
</AUDIOTRACKS>
```

  This is a syntax error. Element names must start with a letter (uppercase or lowercase) or an underscore or a colon.

- One student also wrote:

```
<AUDIOTRACKS>
    <A No=1>German</A>
    <A No=2>English</A>
    <A No=3>Spanish</A>
</AUDIOTRACKS>
```

This is also a syntax error. Attribute values must always be enclosed in quotes (single or double quotes).

- Furthermore, XML defines a sequence on the child elements of an element node, namely the document sequence (from top to bottom). Therefore, it is not necessary to explicitly use numbers to define the first, second, etc. audio track. On the contrary, the given relational database does not define what is the first, second, etc. audio track.