

Dr. Stefan Brass

May 8, 2000



Note that it is not allowed to use the \FOREIGN KEY" keywords if you formulate the foreign key as a column constraint. Also, the parentheses around TABLE\_NAME are required, even though the foreign key consists only of single column in this case.

Some students used \VARCHAR(1)" for \NULLABLE

Two students wrote `NUMERIC(1000)`, because the exercise said that column numbers until 1000 are needed. However, the parameter is the number of digits, not the maximal value. One student wrote `INT(4)`, but `INT` does not allow a parameter. Three students declared `COLUMN_ID` as `VARCHAR(4)`, but this allows illegal values (non-numbers) in this column, and even if you use only numbers, you get a wrong order (e.g. `'2' > '10'`).

`NOT NULL` constraints were forgotten only in two cases.

**Exercise 3 (SQL)****18 Points**

The results of this exercise were quite disappointing: Only 3 students (out of 70) got the full points, 5 students lost 0.5 or 1 point, 10 students lost 1.5 or 2 points, 5 students lost 2.5 or 3 points, 21 students lost between 3.5 and 6 points, 18 students lost between 6.5 and 9 points, 7 students lost more than 9 points. The maximum lost was 14.5 points. Many students said that they were not prepared to do the step to the meta-data, and that a standarhomeenqithrme47-data,ary,the





The following query lists also indexes containing only one of the two columns:

```
SELECT INDEX_NAME      Wrong!
FROM   USER_IND_COLUMNS
AND    TABLE_NAME = 'STUDENTS'
AND    (COLUMN_NAME = 'FIRST_NAME'
OR     COLUMN_NAME = 'LAST_NAME' )
```

b) Which indexes are only on one column?

A very common error was to do the HAVING in the same query as the selection of the column name. This error came in two variations. 9 students (13%) wrote basically this query:

```
SELECT  INDEX_NAME, TABLE_NAME, COLUMN_NAME      Wrong!
FROM    USER_IND_COLUMNS
GROUP BY INDEX_NAME
HAVING  COUNT(*) = 1
```

This does not work, because all attributes used in the select list outside ag-

```
SELECT INDEX_NAME, TABLE_NAME, COLUMN_NAME
```

**Wrong!**

This is a legal SQL query. Using GROUP BY without an aggregation function is only a strange way of duplicate elimination. But in this case, there can be no





```
SELECT TABLE_NAME, COUNT(*)
```

- d) Define a view which contains table name, column name, and ID of all columns which are contained in an index.

This needs simply a join between COLS and USER\_INDEX\_COLUMNS:

```
CREATE VIEW INDEXED_COLUMNS AS
```





of USER\_IND\_COLUMNS





f) Use the views defined in d) and e) in a query which lists all columns (table name,



**Exercise 4 (FDs, BCNF)**

**6 Points**

18 students (26%) got the full points, 17 students (24%) lost 1 point, 17 students (24%)

d) Given the three functional dependencies, is "INDEX\_NAME, COLUMN\_NAME" an alternative key for the relation?

- Yes.  
 No.

15 students (21%) got this wrong. It is an alternative key because of the FDs

INDEX\_NAME ! TABLE\_NAME

INDEX\_NAME, COLUMN\_NAME ! COLUMN\_POSITION

The first FD is stronger than (implies)

INDEX\_NAME, COLUMN\_NAME ! TABLE\_NAME

so that the key attributes together determine all other attributes of the table.

e) Is the relation in BCNF? You may ignore the third FD for this question, since

21 students (30%) got this wrong: 6 checked the first box, and 15 the last.

**Exercise 5 (Security)**

**3 Points**

~~1~~ No.

2 Yes.

it wmakvtmv asputio,an 64 Td[0(wr33(Oracle)-310(an)27ouldDB2putio,requir19sen5.shaanv)28  
28 students (40%) got this wrong. If it were so easy to circumvent tgg.