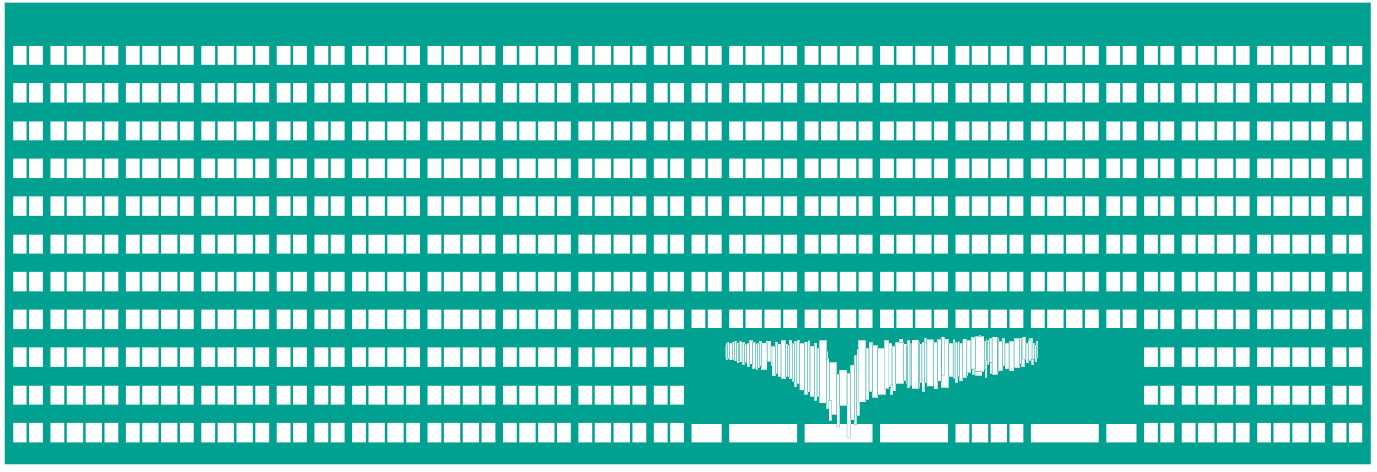
VŠB TECHNICKÁ UNIVERZITA OSTRAVA

VSB TECHNICAL UNIVERSITY OF OSTRAVA



www.vsb.cz

Database and Information Systems

db.cs@vsb.cz

Department of Computer Science
Faculty of Electrical Engineering and Computer Science
VSB - Technical University of Ostrava

2019/2020

Table `Student`

Note: unless noted otherwise all tasks are for the table `Student`:

```
CREATE TABLE Student (
  login     CHAR(6) PRIMARY KEY,
  fname     VARCHAR(30) NOT NULL,
  lname     VARCHAR(50) NOT NULL,
  email     VARCHAR(50) NOT NULL);
```

Task 1: Stored procedures and functions

1. Create stored procedure `AddStudent` with 4 parameters `p_login, p_fname, p_lname, p_email`, which will insert new record into table `Student`. Run procedure with command `EXECUTE`.
2. Create stored procedure `PAddStudent`, which will work the same way like the procedure `AddStudent` and it will return 'ok' in output parameter, if record is successfully inserted, otherwise it will return 'error' (Use `TRY CATCH`). Use `print` to print the output parameter.

Table Teacher

```
CREATE TABLE Teacher (
  login CHAR(6) NOT NULL PRIMARY KEY,
  fname VARCHAR(30) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  email VARCHAR(50) NOT NULL,
  department INT NOT NULL,
  specialization VARCHAR(30) NULL);
```

Task 2: Variables

1. Create stored procedure `StudentBecomeTeacher` with 2 parameters `p_login` a `p_department`, which will move student with login `p_login` from table `Student` into table `Teacher`.
2. Modify stored procedure `StudentBecomeTeacher` to be one transaction.

Table `Student`

Note: unless noted otherwise all tasks are for table `Student`:

```
CREATE TABLE Student (
  login     CHAR(6) PRIMARY KEY,
  fname     VARCHAR(30) NOT NULL,
  lname     VARCHAR(50) NOT NULL,
  email     VARCHAR(50) NOT NULL,
  tallness INT NOT NULL);
```

Task 3: Variables

1. Create stored procedure `AddStudent2` with 3 parameters `p_fname, p_lname, p_tallness`, which :
   - will change all characters of last name to lower case
   - will create login from last name (parameter `p_lname`) with adding '000'
   - will create email from login with adding '@vsb.cz'
   - will insert record into the table `Student`

Task 4: Control Structures

1. Add to table `Student` attribute `isTall`, which can be 0 or 1.
2. Create stored procedure `IsStudentTall` with one input parameter `p_login`, which will find record with current login and setup attribute `isTall` to 0 if attribute `tallness` is less than the average tallness, otherwise 1.
3. Create function `LoginExist` with one input parameter `p_login`, which will return true if record with current login `p_login` exists. Use function `LoginExist` to extend procedure `AddStudent2`, which will be generating new login until it will find unused login (use command WHILE).

Task 5: Cursor

1. Create stored procedure `SetStudentTallness`, which will set the attribute `isTall` for each student (see procedure `IsStudentTall`). The procedure will be parameterless. Use commands OPEN, FETCH and CLOSE.

Task 6: Table backup

1. Create stored procedure `CopyTableStructure` with two input parameters `p_table_schema, p_table_name`, which will create a copy (only attributes) of table with the name `p_table_name` from schema `p_table_schema`. New table will be empty, it will have suffix '_old', and it will have the same attributes with the same names (and types) like original table.

   Tip: Names and types of attributes select from system catalog [1] . Build the complete command `CREATE TABLE` (which will create a new table) into some string variable.

2. Create stored procedure `CopyTable` with same parameters, which will create a copy of table and then will copy a data from original table into backup table.

---

[1] SELECT * FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA = <login> AND TABLE_NAME = 'Student';

Task 7: Triggers

1. Create triggers which will record into table `Statistics` count of insert, update a delete operations on the table `Student`. Table `Statistics` has two attributes. First attribute `operation` is a type of operation and second attribute `operationCount` is a count of current operations .
2. Create table `Course (id, courseName, capacity)` and association table `StudyPlan` between tables `Student` and `Course`. Create trigger `controlCapacity`, which will generate exception in the case of capacity overload.

Task 8.1: Create tables

Create tables `Teacher` and `Department`:

- `Teacher` with attributes:
  - `login CHAR(5)`, primary key,
  - `fname VARCHAR(30)`,
  - `lname VARCHAR(50)`,
  - `email VARCHAR(50)`,
  - `department NUMBER`, foreign key, primary key of table `Department`.
- `Department` with attributes:
  - `id NUMBER`, primary key, use auto_incremented value,
  - `name VARCHAR(50)`,
  - `head CHAR(5)`, foreign key, primary key of table `Teacher`.

All attributes except `Department.head` are NOT NULL.

Task 8.1: Notes

- IO IDENTITY serves for automatic generation of value (mainly used for primary keys).
- Foreign key, f.e.:

```
...  FOREIGN KEY REFERENCES Teacher
```

Task 8.2: Drop tables

- Drop tables.
- Note: it is possible to solve it by adding/dropping of IO FOREIGN KEY.

Task 8.3: Insert records

1. Insert several records into table `Department` with automatic incrementation of primary key.
2. Insert several records into table `Teacher`.
3. Set the head for each department – update attribute `head` in the table `Department`.

Task 9: Print report

Write stored procedure `PrintReport()`, that will print out report of teachers at the departments with more than 1 teacher. Print login, name, surname, email and department_id for each teacher. The procedure has to contain only one cursor.

Task 10: Table backup

Write stored procedure `CopyTableDate(<scheme>, <table>)`, which will create the table with name `<table>_dcp` and will have the same attributes with the same names (and types) like original table + new attribute `date_copy`. footnoteSELECT * FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA = <login> AND TABLE_NAME = 'Student';. Copy all attributes from original table into the new table. The attribute `data_copy` will be set on current date.

Notes to date-time data types

- In previous task use data type DATE and function GETDATE() for getting current date or data type DATETIME and function CURRENT_TIMESTAMP.
- Study the functions and methods for date-time data types:
  - Variant 1:

    ```
    CAST(N'2017-08-08 03:36:00.0000000' AS
    DateTime2)
    ```

  - Variant 2:

    ```
    SELECT CAST('01-JAN-2009' AS DATETIME)
    SELECT CONVERT(DATETIME,'01/JAN/2009',101)
    print CONVERT(DATE,'01.01.2011')

    DECLARE @mDate DATETIME
    SELECT @MDate = '01/JAN/09'
    SELECT CONVERT(VARCHAR(20),@mDate,106)
    ```

    Where 106 is the format (see next slide).

Date-time formats

```
101 U.S. mm/dd/yyyy
102 ANSI yy.mm.dd
103 British/French dd/mm/yyyy
104 German dd.mm.yy
105 Italian dd-mm-yy
106 dd mon yy
107 Mon dd, yy
108 hh:mi:ss
109 mon dd yyyy hh:mi:ss:mmmAM
110 USA mm-dd-yy
111 JAPAN yy/mm/dd
112 Yymmdd
113 dd mon yyyy hh:mi:ss:mmm(24h)
114 hh:mi:ss:mmm(24h)
120 yyyy-mm-dd hh:mi:ss(24h)
121 yyyy-mm-dd hh:mi:ss.mmm(24h)
126 yyyy-mm-ddThh:mi:ss.mmm
127 yyyy-mm-ddThh:mi:ss.mmmZ (With Time Zone)
130 dd mon yyyy hh:mi:ss:mmmAM (Hijri)
131 dd/mm/yy hh:mi:ss:mmmAM (Hijri)
```