Project - Mini DBMS Requirements

2017 CS4710 Introduction to Database Systems
National Tsing Hua University

Project Stages

- The project will be developed in three stages:

    1. Table creation and insertion – **March 15 (Wed)**
    2. Intermediate SQL – **April 12 (Wed)**
    3. Indexing – **May 10 (Wed)**

- At the end of each stage, each group must present a **live demo** of the project functioning.
- The application must be robust and be able to handle a number of cases, as the T.A.s will use their own input data to test each stage.
- After each demo, the students must answer a questionnaire about each team member's contribution to the project development.

Evaluation

1. **Table creation and insertion**		(25%)
• Table Creation	10%	
• Data Insertion	10%	
• Questionnaire	5%	
2. **Intermediate SQL**		(30%)
• SQL File Import	5%	
• SQL Queries	20%	
• Questionnaire	5%	
3. **Indexing**		(45%)
• Individual query times	15%	
• Total query times	15%	
• Questionnaire	15%	

Stages Total : 100%

(30% of the total course grade)

Stage 1
Table Creation and Insertion
(25%)

Stage 1

- Live demo date: **Wednesday, March 15**
- In this stage, the application must be able to create a database with tables and attributes.
- The database must also support data insertion in this stage.
- Your application will be tested in real-time by the T.A.s using some arbitrary SQL statements.

Stage 1

- **(5%) Questionnaire**
- **(10%) Table Creation**
    - The DBMS must support the creation of up to 10 tables using standard SQL code.
    - Tables must be able to support primary keys (and no primary keys).
    - Each table must be able to contain up to 10 attributes.
    - The attribute types to be supported are:
        - **int** (4-byte integer from -2,147,483,648 to 2,147,483,647)
        - **varchar** (variable character string up to size 40).
- **Each group must be able to show each table created after running an SQL statement.**

Stage 1

Table Creation SQL Samples (see **4. Notes about SQL Syntax**):

```
CREATE TABLE Person (
     person_id int PRIMARY KEY,
     name varchar(20),
     gender varchar(1)
);
create table department (
     department_name varchar(20),
     type varchar(1),
     num_employees int,
     code varchar(10),
     city varchar(15)
);
Create    tAbLe     pRoduct
(
Product_id int PRImaRY KEY, product_name      varchar(26))
```

SQL is NOT case-sensitive

The number of spaces in a command doesn't matter. Even a new line (\n) is considered as a space.

Stage 1

- **(10%) Data Insertion**
  - The DBMS must support the insertion of at least 10 tuples per table.
  - Data insertion must check for the following constraints and validations, and display error messages when a user makes an error:
    - Primary key (cannot be null or repeated)
    - Data type validation (e.g. cannot insert string into int)
    - Column size validation (e.g. do not exceed the max. size of a varchar attribute)
- **Each group must be able to show each table and its corresponding data after running an SQL statement.**

Stage 2
Intermediate SQL
(30%)

Stage 2

- Live demo date: **Wednesday, April 12**
- In this stage, the application must be able to **support SQL file import.**
- The application must also be able to support **SQL queries** with several features:

  ① The '*' symbol

  ② The WHERE clause

  ③ Attribute prefixes and table aliases

  ④ Table joins

  ⑤ Aggregation functions

Stage 2

- **(5%) Questionnaire**
- **(5%) SQL File Import**
  - The DBMS must allow the user to **import** an SQL text file and execute it (the file extension will be “.sql”).
  - If a file contains more than one SQL statement, each statement will be separated by a semicolon ';' at the end of the statement.
  - All statements in an SQL file must be executed.
  - The system must support the insertion of at least 5,000 tuples in each table.

Stage 2

SQL File sample (see 4. Notes about SQL Syntax)**:**

```
INSERT INTO Student (studentId, name, age)
VALUES (1, 'John', 13);
INSERT INTO Student (studentId, name, age)
VALUES (2, 'Mary', 11);
INSERT INTO Student (studentId, name, age)
VALUES (3, 'Will', 12);
INSERT INTO Student (studentId, name, age)
VALUES (4, 'Sue', 13);
INSERT INTO Student (studentId, name, age)
VALUES (5, 'James', 14);
```

Stage 2

- **(20%) SQL Queries**
- The DBMS must support the **SQL SELECT** statement
  - The system must allow SQL SELECT from up to 10 attributes in a table.
  - The '*' symbol must be supported, which indicates that all attributes from the table (or tables) in the FROM clause will be obtained.
  - The system must support SELECT queries from up to 2 tables simultaneously.
    - For this case, attribute prefixes and table aliases must also be supported.
    - For table aliases, the “AS” restricted word must be written between the table name and the alias in the FROM clause.
    - A table alias can be used as a prefix for an attribute (e.g. D.name).
    - If a table alias is not used and there is an ambiguous attribute between two tables (e.g. both the Student and Department tables have an attribute called “name”), then the table name has to be used as a prefix in the attribute (e.g. Student.name and Department.name).
    - A single SQL query can have a combination of table name prefixes, alias prefixes and no prefixes at all. Only attributes that are not ambiguous do not need a prefix (e.g. the attribute “studentId” only appears in the Student table)

Stage 2

SQL SELECT samples with 1 table
(see 4. Notes about SQL Syntax):

```
SELECT *
FROM Person;

SELECT person_id, name
FROM Person;

SELECT licensePlate, brand, model, year, engineSize
FROM Car;

SELECT *
FROM Car;
```

Stage 2

SQL SELECT samples with 2 tables

(see 4. Notes about SQL Syntax):

```
// Select all attributes from the Student and Department tables. If both tables have attributes with the same
name, then the query will always return the attributes from both tables
SELECT *
FROM Student, Department;

// Unambiguous attributes in both tables
SELECT studentId, name, licensePlate, brand
FROM Student, Car;

// The attribute “name” is ambiguous because it exists in the Student and Department tables, so a prefix is
required (in this case, an alias was given to both tables)
SELECT S.studentId, S.name, D.name
FROM Student AS S, Department AS D;

// In this case, an alias was only given to Department, but the “name” attribute always needs a prefix,
whether it's an alias or the name of the table itself. The attribute “studentId” doesn't need a prefix
because it only appears in the Student table (but it could optionally have a prefix)
SELECT studentId, Student.name, D.name
FROM Student, Department AS D;
```

Stage 2

- **(20%) SQL Queries**
- The DBMS must support the **WHERE** clause in SQL queries.
    - The WHERE clause must support up to two Boolean expressions composed of one of the following logical operators:

        ① '=': equals

        ② '<>': is not equal to

        ③ '<': is less than

        ④ '>': is greater than

    - Two Boolean expressions can be separated by one of the following: “AND” and “OR”.

Stage 2

SQL SELECT samples with WHERE clause
(see 4. Notes about SQL Syntax):

```
SELECT *
FROM Student
WHERE gender = 'M';

SELECT departmentId, name
FROM Department
WHERE departmentId > 1000 AND buildingNo = 101;

SELECT *
FROM Car
WHERE brand = 'Honda' OR year > 2010;
```

Stage 2

- **(20%) SQL Queries**
  - The DBMS must support **inner joins** in SQL queries between two tables.
    - Inner joins can be made using a Boolean expression in the WHERE clause, simulating a foreign key constraint.
    - The restricted word “INNER JOIN” is not required, since the joins will be done in the WHERE clause.
    - Inner joins can also be made between a same table (self-reference).

Stage 2

SQL SELECT samples with inner joins

(see 4. Notes about SQL Syntax):

// Select the student ID, student name and department name of all students (assuming all students belong to a department)

```
SELECT studentId, S.name, D.name
FROM Student AS S, Department AS D
WHERE S.departmentId = D.departmentId;
```

// Select the student ID, student name, student age and department name of all students belonging to departments which have a department ID greater than 1000

```
SELECT Student.studentId, Student.name, Student.age, Department.name
FROM Student, Department
WHERE Student.departmentId = Department.departmentId AND
Department.departmentId > 1000;
```

// Select the employee id and employee name of all employees who work for employee 1234, as well as the name of employee 1234.

```
SELECT Subordinate.employeeId, Subordinate.name, Boss.name
FROM Employee AS Subordinate, Employee AS Boss
WHERE Subordinate.bossId = Boss.employeeId AND Boss.employeeId =
1234;
```

Stage 2

- **(20%) SQL Queries**
  - The DBMS must support **simple aggregation functions** (COUNT and SUM).
    - The COUNT aggregation function must be supported by the system.
    - The SUM aggregation function must be supported by the system
      - The SUM and COUNT aggregation functions do not require a GROUP BY clause.
      - These two aggregation functions can be made in simple SELECT queries.

Stage 2

SQL SELECT samples with COUNT aggregation
(see 4. Notes about SQL Syntax):

```
// Select the total number of students in the database
SELECT COUNT(*)
FROM Student;

// Also select the total number of students in the database (unless
the "name" attribute has null values)
SELECT COUNT(name)
FROM Student;

// Select the total number of employees whose salaries are more
than $1000.00
SELECT COUNT(*)
FROM Employee
WHERE salary > 1000;
```

Stage 2

SQL SELECT samples with SUM aggregation
(see 4. Notes about SQL Syntax):

```
// Select the sum of all employee salaries in the database
SELECT SUM(salary)
FROM Employee;

// Select the sum of all employee salaries in the database
who work in department 1040
SELECT SUM(salary)
FROM Employee
WHERE departmentId = 1040;
```

Stage 3
Indexing
(45%)

Stage 3

- Live demo date: **Wednesday, May 11**
- In this stage, the application must be able to support at least two indexing algorithms.
- **Specific requirements:**
  - The DBMS must now support the insertion of at most 50,000 tuples.
  - The data should be stored at disks (NOT in memory).
  - The DMBS must be able to support at least two types of indexing algorithms.
  - The DBMS must also allow the user to choose which indexing algorithm s/he wishes to use on a specific index.
  - The efficiency of the indexing algorithms will be measured by the speed of queries.

Stage 3

- The DBMS must allow the user to select one or two attributes per table for applying an index.
    - **Tree Type Indexing**
        - The DBMS must support at least one of the many tree-type indexing algorithms.
    - **Hashing Type Indexing**
        - The DBMS must support at least one hashing-type indexing algorithm.

Stage 3

- **Determining the speed of a query**
  - The DBMS must have a way to display the execution time of any SQL statement.
  - The execution time for any SQL statement must be displayed in milliseconds or seconds.
  - The execution time will first be tested by the graders to prove that the time displayed is possibly correct.
  - If there is a significant difference between the displayed time and the measured time, the measured time will be adapted for evaluation.
  - If there is no significant difference, the displayed time will be used for evaluation.

Stage 3

- **(15%) Questionnaire**
- **(30%) Demo time**
    - Too many, please read **notes about stage3.**

Outline

1. Introduction
2. Requirements
3. Project Stages
    - Evaluation
    - Stage 1 : Table Creation & Insertion
    - Stage 2 : Intermediate SQL
    - Stage 3 : Indexing
4. **Notes about SQL syntax**

Notes about SQL syntax

- **Spacing**
  - The SQL should not care about the number of spaces between tokens.
  - The new line character (\n) is also considered a space.
- **The following statements should be valid and are all the same:**

  -
    ```
    SELECT studentId, name
       FROM Student
    ```

  -
    ```
    SELECT studentId, name FROM Student
    ```

  -
    ```
    SELECT studentId, name FROM;
       Student
    ```

  -
    ```
    SELECT       studentID,name       FROM
       Student
    ```

  -
    ```
    SELECT
       studentId,
       name
       FROM
       Student
    ```

Notes about SQL syntax

- **Varchar attributes**
  - Varchar values are written inside single quotes (').
  - Varchar attributes should accept spaces inside them.
- **Case-sensitivity**
  - SQL commands, attribute names and table names are NOT case-sensitive.
  - The text inside varchar attributes is case-sensitive.
- **INSERT statements**
  - The INSERT statement has two different syntaxes:
    - One in which the name of the attributes are listed
    - One in which the name of the attributes are not listed, but the values must be input in the same order as they appear in the table.

Notes about SQL syntax

- **The following statements should be valid and are all the same:**

  -
    ```
    INSERT INTO Person
    VALUES (1234, 'John Smith')
    ```

  -
    ```
    insert into person
    values (1234, 'John Smith')
    ```

  -
    ```
    iNsErT iNtO pErSoN(personId, name)
    vAlUeS(1234, 'John Smith')
    ```

  -
    ```
    INSERT INTO PERSON VALUES(1234,'John Smith')
    ```

Notes about SQL syntax

- **Semicolons**
  - Your SQL needs a semicolon (;) between each command if there are several commands.
  - If only one command is being executed, then the semicolon (;) can be omitted.

Any Questions?
send a mail to us!