IS5102
Database Management Systems
Lecture 8: Introduction to SQL

Alexander Konovalov
alexander.konovalov@st-andrews.ac.uk

(with thanks to Susmit Sarkar)

2021

Introduction to SQL

- Overview of the SQL Query Language
- Data Definition
- Basic Query Structure

Integrity Constraints in CREATE TABLE

- `NOT NULL`
- `PRIMARY KEY (A1, ..., An)`
  primary key declaration on an attribute automatically ensures not null
- `FOREIGN KEY (Am, ..., An) REFERENCES r`

Example: updated declaration of `department` from the previous lecture:

```
CREATE TABLE department (
    dept_id     CHAR(5),
    dept_name   VARCHAR(20) NOT NULL,
    building    VARCHAR(15),
    budget      NUMERIC(12,2),
    PRIMARY KEY (dept_id)
);
```

Integrity Constraints in CREATE TABLE

Example: Declare `instr_id` as the primary key for instructor and `dept_id` a foreign key

```
CREATE TABLE instructor (
    instr_id    CHAR (5),
    instr_name  VARCHAR(20) NOT NULL,
    dept_id     VARCHAR(5),
    salary      NUMERIC (8,2),
    PRIMARY KEY (instr_id),
    FOREIGN KEY (dept_id) REFERENCES department);

INSERT INTO instructor
VALUES ('45797', 'Bob', 'CS', 28000),
       ('12355', 'Petro', 'MATH', 32000),
       ('23456', 'Alice', 'PHYS', 29500),
       ('45638', 'Sana', 'PHYS', 31500);
```

Warnings

Warning 1: In SQLite, we have to use

```
PRAGMA foreign_keys = TRUE;
```

to enforce foreign key constraints

Warning 2: In MariaDB, have to write

```
FOREIGN KEY (dept_id) REFERENCES department(dept_id)
```

And a few more relation definitions

```
CREATE TABLE student (
   stud_id   CHAR(5),
   name      VARCHAR(20) NOT NULL,
   dept_id   VARCHAR(20),
   tot_cred  NUMERIC(3,0),
   PRIMARY KEY (stud_id),
   FOREIGN KEY (dept_id) REFERENCES department);

INSERT INTO student
VALUES ('64545', 'Abdul', 'MATH', 180),
       ('79879', 'Tom', 'CS', 90),
       ('89675', 'Eilidh', 'PHYS', 120),
       ('96544', 'Sarah', 'PHYS', 180);
```

```
CREATE TABLE course (
   course_id  VARCHAR(8),
   title      VARCHAR(50),
   dept_id    VARCHAR(20),
   credits    NUMERIC(2,0),
   PRIMARY KEY (course_id),
   FOREIGN KEY (dept_id) references department);

INSERT INTO course
VALUES ('CS1234', 'Python', 'CS', 15),
       ('CS2234', 'Haskell','CS', 15),
       ('MT4665', 'Algebra', 'MATH', 15),
       ('PH3457', 'Photonics', 'PHYS', 30);
```

Altering a table: used to add or delete attributes from an existing relation

Syntax:

```
ALTER TABLE table r ADD A D
```

- where A is the name of the attribute to be added to relation r and D is the domain of A
- All tuples in the relation are assigned NULL as the value for the new attribute

```
ALTER TABLE r DROP A
```

- where A is the name of an attribute of relation r
- Warning: Dropping of attributes not supported by many systems

- The `SELECT` clause list the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra

- Example: find the names of all instructors:

```
SELECT instr_name FROM instructor
```

- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
  - E.g., `Select` = `SELECT` = `select`
  - We follow the SQL Style Guide: https://www.sqlstyle.guide/

The SELECT Clause (Cont.)

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword `DISTINCT` after `SELECT`

- Find the department names of all instructors, and remove duplicates

```
SELECT DISTINCT dept_id
  FROM instructor;
```

- The keyword `all` specifies that duplicates not be removed.

```
SELECT ALL dept_id
  FROM instructor;
```

- An asterisk in the SELECT clause denotes "all attributes"

```
SELECT *
  FROM instructor;
```

- The SELECT clause can contain arithmetic expressions involving the operations +, −, *, and /, and operating on constants or attributes of tuples.
- The query:

```
SELECT instr_id, instr_name, salary/12
  FROM instructor;
```

would return a relation that is the same as the instructor relation, except that the value of the attribute salary is divided by 12.

The WHERE Clause

- The `WHERE` clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra.

- To find all instructors in Physics dept with `salary > 30000`

```
SELECT instr_name
  FROM instructor
 WHERE dept_id = 'PHYS' AND salary > 30000;
```

- Comparison results can be combined using the logical connectives `AND`, `OR`, and `NOT`.
- Comparisons can be applied to results of arithmetic expressions.

The FROM Clause

- The FROM clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product instructor $\times$ teaches

```
SELECT *
  FROM instructor, teaches;
```

  - generates every possible instructor – teaches pair, with all attributes from both relations.
- Cartesian product not very useful directly, but useful combined with WHERE clause condition (selection operation in relational algebra).

- For all instructors who have taught courses, find their names and the course ID of the courses they taught.

```
SELECT instr_name, course_id
  FROM instructor, teaches
 WHERE instructor.instr_id = teaches.instr_id;
```

`NATURAL JOIN` matches tuples with the same values for all common attributes, and retains only one copy of each common column

```
SELECT * FROM instructor NATURAL JOIN teaches;
```

Compare this with

```
SELECT *
  FROM instructor, teaches
 WHERE instructor.instr_id = teaches.instr_id;
```

Also compare

```
SELECT instr_name, course_id
  FROM instructor NATURAL JOIN teaches;
```

with

```
SELECT instr_name, course_id
  FROM instructor, teaches
 WHERE instructor.instr_id = teaches.instr_id;
```

for listing the names of instructors along with the course ID of the courses that they taught.

{Are these equivalent? }

The Rename Operation

- The SQL allows renaming relations and attributes using the `AS` clause:
  `old-name AS new-name`

E.g.,

```
SELECT instr_id, instr_name, salary/12 AS monthly_salary
  FROM instructor;
```

- Find the names of all instructors who have a higher salary than some instructor in Physics:

```
SELECT DISTINCT T.instr_name
  FROM instructor AS T, instructor AS S
 WHERE T.salary > S.salary
   AND S.dept_id = 'PHYS';
```

- Keyword `AS` is optional and may be omitted
  `instructor AS T = instructor T`

Insertion – more examples

- Add a new tuple to course

```
INSERT INTO course
VALUES ('IS5102', 'DBMS','CS', 15);
```

- or equivalently

```
INSERT INTO course (course_id, title, dept_id, credits)
VALUES ('IS5102', 'DBMS','CS', 15);
```

- or with a different order of attributes

```
INSERT INTO course (course_id, title, credits, dept_id)
VALUES ('IS5040', 'HCI', 15, 'CS');
```

- Add new tuples to `student` with `tot_creds` set to `NULL` in two ways:

```
INSERT INTO student
VALUES ('65467', 'Emma', 'CS', NULL);
INSERT INTO student (stud_id, name, dept_id)
VALUES ('83456', 'Nick', 'MATH' );
```

- Chapter 3, Database System Concepts, Silberschatz, Korth and Sudarshan
- Chapter 6, Database Systems, Connolly and Begg
- Chapters 15-16, Database Design, Watt and Eng
- Useful sites
  - http://www.w3schools.com/sql/
  - http://sqlzoo.net/wiki/Main_Page