Databases

Part 2/2
SQL

Kai Brünnler
CAS Applied Data Science
University of Bern

Material adapted from: Silberschatz, Korth, Sudarshan: Database System Concepts. 6th Edition.

Comparison Procedural vs Declarative

- Example: Find the name of the instructor with ID 22222

```
import csv

with open('instructor.csv') as file:
    reader = csv.DictReader(file)
    for row in reader:
        if row['ID'] == '2222':
            print(row['name'])
```

```
select  name
from    instructor
where ID = '22222'
```

Comparison Procedural vs Declarative

- Example: Find the building of the instructor “Einstein”

```
import csv

with open('instructor.csv') as file:
    reader = csv.DictReader(file)
    for row in reader:
        if row['name'] == 'Einstein':
            dept_name = row['dept_name']

with open('department.csv') as file:
    reader = csv.DictReader(file)
    for row in reader:
        if row['dept_name'] == dept_name:
            print(row['building'])
```

```
select  building
from    instructor, department
where   instructor.dept_name =
        department.dept_name
        and
        name = 'Einstein'
```

Create Table

- Example:

```
create table instructor (
        ID              char(5),
        name            varchar(20),
        dept_name       varchar(20),
        salary          numeric(8,2))
```

- Example with Integrity Constraints:

```
create table instructor (
      ID              char(5),
      name            varchar(20) not null,
      dept_name       varchar(20),
      salary          numeric(8,2),
      primary key (ID),
      foreign key (dept_name) references department)
```

The select Clause

- By default SQL lists duplicate tupels:

  select *dept_name*
  from *instructor*

  - To force the elimination of duplicates**:**

    select distinct *dept_name*
    from *instructor*

  - To retain duplicates (default):

    select all *dept_name*
    from *instructor*

- An asterisk denotes “all attributes”

  select *
  from *instructor*

- Can contain arithmetic expressions:

  select *ID, name, salary / 12*
  from *instructor*

Natural Join

- Natural join joins two tables in the natural way:
  - By combining all rows that have the same values on the common attributes.

- List the names of instructors along with the course ID of the courses that they taught:
  - **select** *name*, *course_id*
    from *instructor, teaches*
    where *instructor.ID = teaches.ID*;

  - **select** *name*, *course_id*
    from *instructor* **natural join** *teaches*;

Natural Join Example

- **select** * **from** *instructor* **natural join** *teaches*;

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

Renaming, String Matching, Ordering

- Often we rename attributes or relations:

  select *ID, name, salary/12* **as** *monthly_salary*
  from *instructor*

- We can match strings using patterns containing:
  - percent (%). Matches any substring.
  - underscore (_). Matches any character.

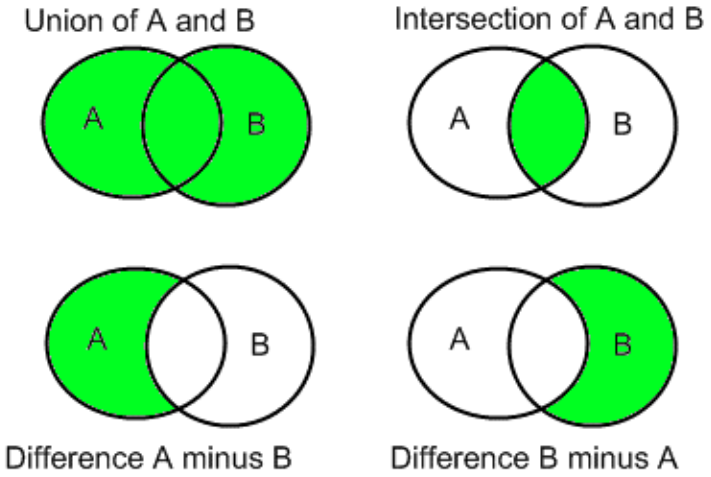
    select *name*
    from *instructor*
    where *name* **like** '_ _ _%stein%'

- List names in alphabetic order:

  select distinct *name*
  from *instructor*
  order by *name*

Set Operations: union, intersect, except

Keyword in SQL	Set Operation
union	union
intersect	intersection
except	difference



- Find courses that ran in 2009 or in 2010 or both:

  select *course_id* **from** *section* **where** *year* = 2009
  union
  select *course_id* **from** *section* **where** *year* = 2010

- Set operations eliminate duplicates
- To retain duplicates use **union all, intersect all** and **except all**

Null Values

- How would you add an instructor if you don't know his/her salary?
  - Use a special value? Like 0 or -1?
- There is the special value **null**, which signifies an unknown or non-existent value
- Every attribute can be **null** unless its explicitly forbidden

Null Values

- Rules for Null values:
  - The result of any arithmetic expression involving *null* is *null*
  - A comparison with *null* returns the special boolean value *unknown*
  - The **where** clause treats *unknown* as *false*

- What's the result of this?

  select *name*
  from *instructor*
  where *salary* = *null*

name	salary
Einstein	80000
Katz	null
Mozart	0

- Null values are often problematic.
  - Check for null with **is null**
  - Always consider the possibility that a value is null.

Aggregate Functions

- Find the average salary of instructors in the Computer Science department
  - **select avg** (*salary*)
    from *instructor*
    where *dept_name*= ’Comp. Sci.’;

- Find the number of tuples in the *course* relation
  - **select count** (*)
    from *course*;

- Find the number of instructors who taught a course in 2010
  - **select count** (**distinct** *ID*)
    from *teaches*
    where *year* = 2010;

Aggregate Functions – Group By

- Find the average salary of instructors in each department
  - **select** *dept_name*, **avg** (*salary*) as avg_salary
    from *instructor*
    group by *dept_name*;

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Nested Queries

- A **subquery** is a query inside another query.
- There are three kinds of subqueries:
  - in the **from**-clause,
  - in the **where**-clause, using **in** or **exists**
  - scalar subqueries (that can occur anywhere).

Subquery in the Where-Clause – in

- Find courses offered in 2009 and in 2010

```
select distinct course_id
from section
where year = 2009 and
course_id in (      select course_id
                    from section
                    where year = 2010 );
```

Subquery in the Where-Clause – exists

- **exists** *r* returns **true** iff *r* is nonempty.

- Find all courses taught in both 2009 and 2010

```
select course_id
from section as S
where year = 2009 and
        exists (select *
                from section as T
                where year = 2010 and S.course_id = T.course_id);
```

  - S is a **correlation variable**
  - the inner query is a **correlated subquery**

Scalar Subqueries

- A **scalar subquery** is a subquery which is used where a single value is expected
  - If it returns more than one tuple it causes a runtime error

- Find the instructors that cost more than 10% of their departments budget:

```
select name
from instructor
where  salary * 10 >
        (select budget  from department
          where department.dept_name = instructor.dept_name)
```

Deletion

- Delete all instructors

  delete from *instructor* ;

- Delete all instructors from the Finance department

  delete from *instructor*
  where *dept_name*= ’Finance’;

- Delete all tuples in the *instructor* relation for those instructors associated with a department located in the Watson building.

  delete from *instructor*
  where *dept_name* **in** (**select** *dept_name*
  from *department*
  where *building* = ’Watson’);

Deletion

- Delete all instructors whose salary is less than the average salary of instructors

  delete from *instructor*
  where *salary* < (**select avg** (*salary*) **from** *instructor*);

- Problem: as we delete tuples, the average salary changes
- Solution used in SQL:
  - First, compute avg salary and find all tuples to delete
  - Only then delete all those tuples

Insertion

- Add a new tuple to *course:*

  insert into *course*
  values (’CS-437’, ’Database Systems’, ’Comp. Sci.’, 4);

- Equivalent and better:

  insert into *course* (*course_id*, *title*, *dept_name*, *credits*)
  values (’CS-437’, ’Database Systems’, ’Comp. Sci.’, 4);

- Add all instructors to the *student* relation with tot_creds set to 0

  insert into *student*
  select *ID, name, dept_name, 0*
  from *instructor*

Updates

- Increase salaries of instructors by 5%:

```
update instructor
set salary = salary * 1.05;
```

- Problem: How to increase salaries of instructors whose salary is over $100,000 by 3%, and all others by 5%?
  - Write two **update** statements?

```
update instructor
    set salary = salary * 1.03
    where salary > 100000;
update instructor
    set salary = salary * 1.05
    where salary <= 100000;
```

Updates – case statement

- Problem as before, solution with case statement:

```
update instructor
   set salary =
         case
            when salary <= 100000 then salary * 1.05
                                  else salary * 1.03
         end
```