CMSC424: Database Design
Relational Model; SQL

February 3, 2020

Instructor: Amol Deshpande
amol@cs.umd.edu
Today’s Plan

- Review of the Reading Homework 1
- Questions from Reading Homework 1
- Keys
  - Foreign keys vs Primary keys
- Relational Algebra
- SQL
  - Single-table queries
  - Joins

- Virtualization/Vagrant/Cloud Computing (last 20 mins)
- Still 14 (at least) who haven’t joined CampusWire
- **Foreign key**: Primary key of a relation that appears in another relation
  - \{ID\} from `student` appears in `takes, advisor`
  - `student` called **referenced** relation
  - `takes` is the **referencing** relation
  - Typically shown by an arrow from referencing to referenced

- **Foreign key constraint**: the tuple corresponding to that primary key must exist
  - Imagine:
    - Tuple: (‘student101’, ‘CMSC424’) in `takes`
    - But no tuple corresponding to ‘student101’ in `student`
  - Also called **referential integrity constraint**
Schema Diagram for University Database

- takes
  - ID
  - course_id
  - sec_id
  - semester
  - year
  - grade

- student
  - ID
  - name
  - dept_name
  - tot_cred

- section
  - course_id
  - sec_id
  - semester
  - year
  - building
  - room_no
  - time_slot_id

- time_slot
  - time_slot_id
  - day
  - start_time
  - end_time

- classroom
  - building
  - room_no
  - capacity

- course
  - course_id
  - title
  - dept_name
  - credits

- department
  - dept_name
  - building
  - budget

- advisor
  - s_id
  - i_id

- prerequisite
  - course_id
  - prerequisite_id

- teaches
  - ID
  - course_id
  - sec_id
  - semester
  - year

- instructor
  - ID
  - name
  - dept_name
  - salary
Schema Diagram for the Banking Enterprise

```
branch
- branch-name
- branch-city
- assets

account
- account-number
- branch-name
- balance

depositor
- customer-name
- account-number

customer
- customer-name
- customer-street
- customer-city

loan
- loan-number
- branch-name
- amount

borrower
- customer-name
- loan-number
```
Some of the languages are “procedural” and provide a set of operations
  ◦ Each operation takes one or two relations as input, and produces a single relation as output
  ◦ Examples: SQL, and Relational Algebra

The “non-procedural” (also called “declarative”) languages specify the output, but don’t specify the operations
  ◦ Relational calculus
  ◦ Datalog (used as an intermediate layer in quite a few systems today)
Choose a subset of the tuples that satisfies some predicate
Denoted by \((r)\) in relational algebra

Relation \(r\)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(A = B \land D > 5\) \((r)\)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Choose a subset of the columns (for all rows)
Denoted by in relational algebra

Relation r

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A,D(r)

<table>
<thead>
<tr>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

Relational algebra following “set” semantics – so no duplicates
SQL allows for duplicates – we will cover the formal semantics later
Must be compatible schemas

What about intersection?

Can be derived

\[ r \cap s = r - (r - s); \]
Cartesian Product

Combine tuples from two relations

If one relation contains N tuples and the other contains M tuples, the result would contain N*M tuples

The result is rarely useful – almost always you want pairs of tuples that satisfy some condition

Relation r, s

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>1</td>
<td>10</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>10</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>20</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>10</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>10</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>20</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>10</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>10</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Joins

Combine tuples from two relations if the pair of tuples satisfies some constraint

Equivalent to Cartesian Product followed by a Select

### Relation r, s

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

Relation \( r \bowtie_{A=C}s \):

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>b</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>b</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Natural Join

Combine tuples from two relations if the pair of tuples agree on the common columns (with the same name)

<table>
<thead>
<tr>
<th>dept_name</th>
<th>building</th>
<th>budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biology</td>
<td>Watson</td>
<td>90000</td>
</tr>
<tr>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>Elec. Eng.</td>
<td>Taylor</td>
<td>85000</td>
</tr>
<tr>
<td>Finance</td>
<td>Painter</td>
<td>120000</td>
</tr>
<tr>
<td>History</td>
<td>Painter</td>
<td>50000</td>
</tr>
<tr>
<td>Music</td>
<td>Packard</td>
<td>80000</td>
</tr>
<tr>
<td>Physics</td>
<td>Watson</td>
<td>70000</td>
</tr>
</tbody>
</table>

Figure 2.5 The department relation.

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

Figure 2.4 Unsorted display of the instructor relation.

department ⋈ instructor:

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>building</th>
<th>budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>Painter</td>
<td>120000</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>Packard</td>
<td>80000</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>Watson</td>
<td>70000</td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>History</td>
<td>Painter</td>
<td>50000</td>
</tr>
<tr>
<td>33456</td>
<td>Gold</td>
<td>Physics</td>
<td>Watson</td>
<td>70000</td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>58583</td>
<td>Califieri</td>
<td>History</td>
<td>Painter</td>
<td>50000</td>
</tr>
<tr>
<td>76543</td>
<td>Singh</td>
<td>Finance</td>
<td>Painter</td>
<td>120000</td>
</tr>
<tr>
<td>76766</td>
<td>Crick</td>
<td>Biology</td>
<td>Watson</td>
<td>90000</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>98345</td>
<td>Kim</td>
<td>Elec. Eng.</td>
<td>Taylor</td>
<td>85000</td>
</tr>
</tbody>
</table>

Figure 2.12 Result of natural join of the instructor and department relations.
Outline

- Overview of modeling
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Basic Data Definition (3.2)
  - Setting up the PostgreSQL database
  - Basic Queries (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)
IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory

Renamed Structured Query Language (SQL)

ANSI and ISO standard SQL:
- SQL-86, SQL-89, SQL-92

Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
- Not all examples here may work on your particular system.

Several alternative syntaxes to write the same queries
Different Types of Constructs

- **Data definition language (DDL):** Defining/modifying schemas
  - **Integrity constraints:** Specifying conditions the data must satisfy
  - **View definition:** Defining views over data
  - **Authorization:** Who can access what

- **Data-manipulation language (DML):** Insert/delete/update tuples, queries

- **Transaction control:**

- **Embedded SQL:** Calling SQL from within programming languages

- **Creating indexes, Query Optimization control...**
Data Definition Language

The SQL **data-definition language (DDL)** allows the specification of information about relations, including:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints
- Also: other information such as
  - The set of indices to be maintained for each relation.
  - Security and authorization information for each relation.
  - The physical storage structure of each relation on disk.
SQL Constructs: Data Definition Language

- `CREATE TABLE <name> ( <field> <domain>, ... )`

```sql
create table department
  (dept_name varchar(20),
   xyz varchar(20),
   building varchar(15),
   budget numeric(12,2) check (budget > 0),
   primary key (xyz, dept_name)
);

create table instructor
  (ID char(5),
   name varchar(20) not null,
   dept_name varchar(20),
   jx varchar(20),
   salary numeric(8,2),
   primary key (ID),
   foreign key (jx, dept_name) references department (xyz, dept_name)
)
```
SQL Constructs: Data Definition Language

- CREATE TABLE <name> ( <field> <domain>, ... )

```
create table department
    (dept_name varchar(20) primary key,
    building varchar(15),
    budget numeric(12,2) check (budget > 0)
);
```

```
create table instructor ( 
    ID char(5) primary key,
    name varchar(20) not null,
    d_name varchar(20),
    salary numeric(8,2),
    foreign key (d_name) references department
)
```
SQL Constructs: Data Definition Language

- drop table student
- delete from student
  - Keeps the empty table around
- alter table
  - alter table student add address varchar(50);
  - alter table student drop tot_cred;
SQL Constructs: Insert/Delete/Update Tuples

- INSERT INTO <name> (<field names>) VALUES (<field values>)
  
  insert into instructor values (‘10211’, ’Smith’, ’Biology’, 66000);
  insert into instructor (name, ID) values (‘Smith’, ‘10211’);
  -- NULL for other two
  insert into instructor (ID) values (‘10211’);
  -- FAIL

- DELETE FROM <name> WHERE <condition>
  
  delete from department where budget < 80000;
  ◦ Syntax is fine, but this command may be rejected because of referential integrity constraints.
We can choose what happens:
(1) Reject the delete, or
(2) Delete the rows in Instructor (may be a cascade), or
(3) Set the appropriate values in Instructor to NULL

We need a relation to describe the association between instructors and the department. We look first at the relation.

Consider the Figure 2.5 shows a sample instance of the database schema.

We can choose what happens:
(1) Reject the delete, or
(2) Delete the rows in Instructor (may be a cascade), or
(3) Set the appropriate values in Instructor to NULL

We look first at the relation.

For example, one can obtain the set of all students in a department. Other set operations can also be performed.

The relation contains only the student identifiers and the department identifiers. The department identifiers are in turn references to the department's identifiers in the department relational schema.

Although it is important to know the difference between a relation schema and an instance, we simply use the relation name.

Where required, we explicitly refer to the schema or to the instance, for example, if we want to find the information about all the instructors who work in the Watson building.

Each course in a university may be offered multiple times, across different semesters, and in different sections.

Figure 2.6 shows a sample instance of the class. The schema is updated. In contrast, the schema of a relation does not generally change.

Figure 2.5 shows the department relation.

DELETE FROM <name> WHERE <condition>

delete from department where budget < 80000;

<table>
<thead>
<tr>
<th>dept_name</th>
<th>building</th>
<th>budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biology</td>
<td>Watson</td>
<td>90000</td>
</tr>
<tr>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>Elec. Eng.</td>
<td>Taylor</td>
<td>85000</td>
</tr>
<tr>
<td>Finance</td>
<td>Painter</td>
<td>120000</td>
</tr>
<tr>
<td>History</td>
<td>Painter</td>
<td>50000</td>
</tr>
<tr>
<td>Music</td>
<td>Packard</td>
<td>80000</td>
</tr>
<tr>
<td>Physics</td>
<td>Watson</td>
<td>70000</td>
</tr>
</tbody>
</table>

Figure 2.5  The department relation.

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

Instructor relation
DELETE FROM <name> WHERE <condition>

delete from department where budget < 80000;

create table instructor

  (ID         varchar(5),
   name       varchar(20) not null,
   dept_name  varchar(20),
   salary     numeric(8,2) check (salary > 29000),
  primary key (ID),
  foreign key (dept_name) references department

  on delete set null
);

We can choose what happens:
(1) Reject the delete (nothing), or
(2) Delete the rows in Instructor (on delete cascade), or
(3) Set the appropriate values in Instructor to NULL (on delete set null)
DELETE FROM <name> WHERE <condition>

- Delete all classrooms with capacity below average

```
delete from classroom where capacity <
(select avg(capacity) from classroom);
```

- Problem: as we delete tuples, the average capacity changes

- Solution used in SQL:
  - First, compute **avg** capacity and find all tuples to delete
  - Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)

- E.g. consider the query: **delete the smallest classroom**
UPDATE <name> SET <field name> = <value> WHERE <condition>

- Increase all salaries’s over $100,000 by 6%, all other receive 5%.
- Write two update statements:

  update instructor
  set salary = salary * 1.06
  where salary > 100000;

  update instructor
  set salary = salary * 1.05
  where salary <= 10000;

- The order is important
- Can be done better using the case statement
**UPDATE <name> SET <field name> = <value> WHERE <condition>**

- Increase all salaries’s over $100,000 by 6%, all other receive 5%.
- Can be done better using the **case** statement

```sql
update instructor
set salary =
    case
        when salary > 100000
            then salary * 1.06
        when salary <= 100000
            then salary * 1.05
    end;
```