CMSC424: Database Design
Relational Model/SQL

Instructor: Amol Deshpande
amol@cs.umd.edu
Today

- Overview of Reading Homework Topics
  - Will cover in more detail since first homework, and some of you don’t have textbook yet

- Relational Model

- No laptop use allowed in the class!!
Some To-Dos

- Sign up for Piazza!

- Set up the computing environment (project0), and make sure you can run Vagrant+VirtualBox, PostgreSQL, IPython, etc.

- Upcoming: Reading Homework 2, Project 1: SQL

- Meet the instructor (1%): Stop by to introduce yourself. Earlier the better.
Topics covered so far

- Why Databases
  - Data Modeling
  - Importance of abstraction/independence layers

- Relational Model
  - Relations, Tuples
  - Primary Keys, Foreign Keys
  - Referential Integrity Constraints

- Relational Algebra Operations

- SQL
  - Data Definition Language: How to create relations, change schemas, etc.
  - Data Manipulation Language: Simple single-table queries
Let $K \subseteq R$

- $K$ is a **superkey** of $R$ if values for $K$ are sufficient to identify a unique tuple of any possible relation $r(R)$
  - *Example:* \{ID\} and \{ID,name\} are both superkeys of instructor.

- Superkey $K$ is **a candidate key** if $K$ is minimal (i.e., no subset of it is a superkey)
  - *Example:* \{ID\} is a candidate key for Instructor

- One of the candidate keys is selected to be the **primary key**
  - Typically one that is small and immutable (doesn’t change often)

- Primary key typically highlighted (e.g., underlined)
Tables in a University Database

classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
Tables in a University Database

takes(ID, course_id, sec_id, semester, year, grade)

What about ID, course_id?

No. May repeat:

(“1011049”, “CMSC424”, “102”, “Fall”, 2015, null)

What about ID, course_id, sec_id?

May repeat:

(“1011049”, “CMSC424”, “101”, “Fall”, 2015, null)

What about ID, course_id, sec_id, semester?

Tables in a University Database

classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building, room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(ID, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)
Foreign key: Primary key of a relation that appears in another relation
- \{ID\} from \textit{student} appears in \textit{takes, advisor}
- \textit{student} called \textit{referenced} relation
- \textit{takes} is the \textit{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 \textit{takes}
  - But no tuple corresponding to ‘student101’ in \textit{student}
- Also called \textit{referential integrity constraint}
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
Examples

- Married(person1_ssn, person2_ssn, date_married, date_divorced)
- Account(cust_ssn, account_number, cust_name, balance, cust_address)
- RA(student_id, project_id, supervisor_id, appt_time, appt_start_date, appt_end_date)
- Person(Name, DOB, Born, Education, Religion, ...)
  - Information typically found on Wikipedia Pages
Examples

- Married(person1_ssn, person2_ssn, date_married, date_divorced)

- Account(cust_ssn, account_number, cust_name, balance, cust_address)
  - If a single account per customer, then: cust_ssn
  - Else: (cust_ssn, account_number)
    - In the latter case, this is not a good schema because it requires repeating information

- RA(student_id, project_id, supervisor_id, appt_time, appt_start_date, appt_end_date)
  - Could be smaller if there are some restrictions – requires some domain knowledge of the data being stored

- Person(Name, DOB, Born, Education, Religion, ...)
  - Information typically found on Wikipedia Pages
  - Unclear what could be a primary key here: you could in theory have two people who match on all of those
Outline

- Overview of modeling
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Setting up the PostgreSQL database
  - Data Definition (3.2)
  - Basics (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)
Relational Query Languages

- Example schema: $R(A, B)$
- Practical languages
  - **SQL**
    - `select A from R where B = 5;`
  - **Datalog** (sort of practical)
    - `q(A) :- R(A, 5)`
- Formal languages
  - **Relational algebra**
    - $\pi_A ( \sigma_{B=5} (R) )$
  - **Tuple relational calculus**
    - $\{ t : \{A\} | \exists s : \{A, B\} ( R(A, B) \land s.B = 5 ) \}$
  - **Domain relational calculus**
    - Similar to tuple relational calculus
Relational Operations

- 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)
**Select Operation**

Choose a subset of the tuples that satisfies some predicate
Denoted by $\sigma$ 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>$\alpha$</td>
<td>$\alpha$</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$\beta$</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>23</td>
<td>10</td>
</tr>
</tbody>
</table>

$\sigma_{A=B \land D > 5}(r)$

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

Relation $r$

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

$\prod_{A,D} (r)$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>α</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>β</td>
<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
Set Union, Difference

Relation \( r, s \):

\[
\begin{array}{c|c}
A & B \\
\hline
\alpha & 1 \\
\alpha & 2 \\
\beta & 1 \\
\end{array}
\quad \begin{array}{c|c}
A & B \\
\hline
\alpha & 2 \\
\beta & 3 \\
\end{array}
\]

\( r \cup s \):

\[
\begin{array}{c|c}
A & B \\
\hline
\alpha & 1 \\
\alpha & 2 \\
\beta & 1 \\
\beta & 3 \\
\end{array}
\]

\( r - s \):

\[
\begin{array}{c|c}
A & B \\
\hline
\alpha & 1 \\
\beta & 1 \\
\end{array}
\]

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

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

\[ r \]

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

\[ s \]

\[ r \times s:\]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>1</td>
<td>α</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>α</td>
<td>1</td>
<td>β</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>α</td>
<td>1</td>
<td>β</td>
<td>20</td>
<td>b</td>
</tr>
<tr>
<td>α</td>
<td>1</td>
<td>γ</td>
<td>10</td>
<td>b</td>
</tr>
<tr>
<td>β</td>
<td>2</td>
<td>α</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>β</td>
<td>2</td>
<td>β</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>β</td>
<td>2</td>
<td>β</td>
<td>20</td>
<td>b</td>
</tr>
<tr>
<td>β</td>
<td>2</td>
<td>γ</td>
<td>10</td>
<td>b</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 $r, s$

$$
\begin{array}{|c|c|}
\hline
A & B \\
\hline
\alpha & 1 \\
\beta & 2 \\
\hline
\end{array}
$$

$$
\begin{array}{|c|c|c|}
\hline
C & D & E \\
\hline
\alpha & 10 & a \\
\beta & 10 & a \\
\beta & 20 & b \\
\gamma & 10 & b \\
\hline
\end{array}
$$

$$
\begin{array}{|c|c|c|c|c|}
\hline
A & B & C & D & E \\
\hline
\alpha & 1 & \alpha & 10 & a \\
\alpha & 1 & \beta & 10 & a \\
\alpha & 1 & \beta & 20 & b \\
\alpha & 1 & \gamma & 10 & b \\
\beta & 2 & \alpha & 10 & a \\
\beta & 2 & \beta & 10 & a \\
\beta & 2 & \beta & 20 & b \\
\beta & 2 & \gamma & 10 & b \\
\hline
\end{array}
$$
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>
<th>building</th>
<th>budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>History</td>
<td>60000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>Comp. Sci.</td>
<td>75000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>98345</td>
<td>Kim</td>
<td>Elec. Eng.</td>
<td>80000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>76766</td>
<td>Crick</td>
<td>Biology</td>
<td>72000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>58583</td>
<td>Califieri</td>
<td>History</td>
<td>62000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>Comp. Sci.</td>
<td>92000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33456</td>
<td>Gold</td>
<td>Physics</td>
<td>87000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>76543</td>
<td>Singh</td>
<td>Finance</td>
<td>80000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2.4** Unsorted display of the instructor relation.

**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)
History

- 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),
     building varchar(15),
     budget numeric(12,2) check (budget > 0),
     primary key (dept_name)
    )
```

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

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

```sql
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.
DELETE FROM <name> WHERE <condition>
delete from department where budget < 80000;

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
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)
SQL Constructs: Insert/Delete/Update Tuples

- **DELETE FROM <name> WHERE <condition>**
  - Delete all classrooms with capacity below average
    ```sql
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*
**SQL Constructs: Insert/Delete/Update Tuples**

- 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:
    ```sql
    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
**SQL Constructs: Insert/Delete/Update Tuples**

- **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;
```