CMSC424: Database Design
Introduction/Overview

Professor: Pete Keleher
keleher@cs.umd.edu

Outline

- 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)
Context

- **Data Models**
  - Conceptual representation of the data
- **Data Retrieval**
  - How to ask questions of the database
  - How to answer those questions
- **Data Storage**
  - How/where to store data, how to access it
- **Data Integrity**
  - Manage crashes, concurrency
  - Manage semantic inconsistencies

Relational Data Model

Introduced by Ted Codd (late 60’s – early 70’s)

- **Before** = “Network Data Model” (Cobol as DDL, DML)
- **Very contentious:** Database Wars (Charlie Bachman vs. Ted Codd)

Relational data model contributes:

1. Separation of logical, physical data models (data independence)
2. Declarative query languages
3. Formal semantics
4. Query optimization (key to commercial success)

1st prototypes:

- Ingres → CA
- Postgres → Illustra → Informix → IBM
- System R → Oracle, DB2
Key Abstraction: Relation

Account =

<table>
<thead>
<tr>
<th>bname</th>
<th>acct_no</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>A-101</td>
<td>500</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-201</td>
<td>900</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-217</td>
<td>500</td>
</tr>
</tbody>
</table>

Terms:

• Tables (aka: Relations)

Why called Relations?

*Closely correspond to mathematical concept of a relation*

Relations

Account =

<table>
<thead>
<tr>
<th>bname</th>
<th>acct_no</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>A-101</td>
<td>500</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-201</td>
<td>900</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-217</td>
<td>500</td>
</tr>
</tbody>
</table>

*Considered equivalent to…*

\[
\{ (Downtown, A-101, 500), \\
(Brighton, A-201, 900), \\
(Brighton, A-217, 500) \}
\]

*Relational database semantics defined in terms of mathematical relations*
Relations

Account =

<table>
<thead>
<tr>
<th>bname</th>
<th>acct_no</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>A-101</td>
<td>500</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-201</td>
<td>900</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-217</td>
<td>500</td>
</tr>
</tbody>
</table>

Considered equivalent to...

\{ (Downtown, A-101, 500),
   (Brighton, A-201, 900),
   (Brighton, A-217, 500) \}

Terms:
- Tables (aka: Relations)
- Rows (aka: tuples)
- Columns (aka: attributes)
- Schema (e.g.: Acct_Schema = (bname, acct_no, balance))

Definitions

Relation Schema (or Schema)

A list of attributes and their domains (elided here for space)
E.g. account(account-number, branch-name, balance)

Programming language equivalent: A variable (e.g. x)

Relation Instance

A particular instantiation of a relation with actual values
Will change with time

<table>
<thead>
<tr>
<th>bname</th>
<th>acct_no</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>A-101</td>
<td>500</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-201</td>
<td>900</td>
</tr>
<tr>
<td>Brighton</td>
<td>A-217</td>
<td>500</td>
</tr>
</tbody>
</table>

Programming language equivalent: Value of a variable
Definitions

Domains of an attribute/column

The set of permitted values
e.g., bname must be String, balance must be a positive real number
We typically assume domains are atomic, i.e., the values are treated as indivisible (specifically: you can’t store lists or arrays in them)

Null value
Used if attribute value is:
◦ unknown (e.g., don’t know address of a customer)
◦ inapplicable (e.g., “spouse name” attribute for a customer)
◦ withheld/hidden
Different interpretations all captured by a single concept – leads to major headaches and problems

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)
Keys

- Let $K \subseteq R$ (R is a set of attributes)
- 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)
  - Chosen by app/user
- Primary key typically highlighted (e.g., underlined)

Tables in a University Database

takes\(\text{ID, course\_id, sec\_id, semester, year, grade}\)

Is ID, course\_id a superkey?
- 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?
- Still no:
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)

Keys

- **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
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)
- 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)
    - Not a good schema because it requires repeating information
- RA(student_id, project_id, supervisor_id, appt_time, appt_start_date, appt_end_date)
- 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
    - $\{ t : \{A\} \mid \exists s : \{A, B\} ( R(A, B) \land s.B = 5) \}$
  - 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: Relational Algebra

- The “non-procedural” (also called “declarative”) languages specify the output, but don’t specify the operations
  - SQL, Relational calculus
  - Datalog (used as an intermediate layer in quite a few systems today)
Relational Algebra teaser

(set semantics)

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 $\Pi$ in relational algebra

<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>

Relation $r$  

<table>
<thead>
<tr>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>7</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>7</td>
</tr>
<tr>
<td>$\beta$</td>
<td>3</td>
</tr>
<tr>
<td>$\beta$</td>
<td>10</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 $r$, $s$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>1</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>2</td>
</tr>
<tr>
<td>$\beta$</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>2</td>
</tr>
<tr>
<td>$\beta$</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>2</td>
</tr>
<tr>
<td>$\beta$</td>
<td>1</td>
</tr>
</tbody>
</table>

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

$r \cap s = r - (r - s)$;
**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>Relation r, s</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>α</td>
<td>1</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>2</td>
<td>10</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>γ</td>
<td>10</td>
<td>b</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ r \times s: \]

<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>α</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>α</td>
<td>β</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>α</td>
<td>β</td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>α</td>
<td>γ</td>
<td>10</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>α</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>β</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>β</td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>γ</td>
<td>10</td>
<td>b</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

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

\[ r \bowtie_{A=C} s: \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>α</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>α</td>
<td>β</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>α</td>
<td>β</td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>α</td>
<td>γ</td>
<td>10</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>α</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>β</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>β</td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>γ</td>
<td>10</td>
<td>b</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>department</th>
<th>instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biology</td>
<td>Califieri</td>
</tr>
<tr>
<td>Comp. Sci.</td>
<td>Srinivasan</td>
</tr>
<tr>
<td>Elec. Eng.</td>
<td>Katz</td>
</tr>
<tr>
<td>Finance</td>
<td>Crick</td>
</tr>
<tr>
<td>History</td>
<td>Wu</td>
</tr>
<tr>
<td>Music</td>
<td>Gold</td>
</tr>
<tr>
<td>Physics</td>
<td>Kim</td>
</tr>
</tbody>
</table>

Figure 2.13 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...**
SQL: 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 relations.
  - 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
)
```
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
)
```

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 (DML)

- 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.

CMSC424: Database Design

SQL

Professor: Pete Keleher
keleher@cs.umd.edu
Recap: 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

- 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.

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
### SQL Constructs: Insert/Delete/Update Tuples

- **DELETE FROM <name> WHERE <condition>**
  - `delete from` department where `budget < 80000`;

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

- UPDATE <name> SET <field name> = <value> WHERE <condition>
  - Increase all salaries over $100,000 by 6%, all other receive 5%.
  - Write two update statements:

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

update instructor
set salary = salary * 1.06
where salary > 10000;
```

- The order is important
- Can be done better using the case statement

```
UPDATE instructor
SET salary =
CASE
  WHEN salary > 100000
    THEN salary * 1.06
  WHEN salary <= 100000
    THEN salary * 1.05
END;
```