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)
Recap: Definitions

**Relation Schema (or Schema)**

A list of attributes and their domains

*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></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

---

Recap: 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**

A special value used if the value of an attribute for a row 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 $K \subseteq R$
- $K$ 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

classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)

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)

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)

- **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\} \mid \exists s : \{A, B\} (R(A, B) \land s.B = 5) \} \}
  - **Domain relational calculus**
    - Similar to tuple relational calculus

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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>α</td>
<td>α</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>α</td>
<td>β</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>β</td>
<td>β</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>β</td>
<td>β</td>
<td>23</td>
<td>10</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>α</td>
<td>α</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>β</td>
<td>β</td>
<td>23</td>
<td>10</td>
</tr>
</tbody>
</table>

Project

Choose a subset of the columns (for all rows)
Denoted by \( \Pi \) in relational algebra

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>α</td>
<td>α</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>α</td>
<td>β</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>β</td>
<td>β</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>β</td>
<td>β</td>
<td>23</td>
<td>10</td>
</tr>
</tbody>
</table>

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

Relational algebra following “set” semantics – so no duplicates
SQL allows for duplicates – we will cover the formal semantics later
Set Union, Difference

Relation $r, s$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$A$</td>
<td>$B$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td>$\alpha$</td>
<td>2</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>2</td>
<td>$\beta$</td>
<td>3</td>
</tr>
<tr>
<td>$\beta$</td>
<td>1</td>
<td>$s$</td>
<td></td>
</tr>
</tbody>
</table>

$\mathbf{r \cup s;}$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$A$</td>
<td>$B$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1</td>
<td>$\alpha$</td>
<td>1</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>2</td>
<td>$\beta$</td>
<td>1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>3</td>
<td>$\beta$</td>
<td>1</td>
</tr>
</tbody>
</table>

$\mathbf{r - s;}$

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

Must be compatible schemas

What about intersection?

Can be derived

$r \cap s = r - (r - s);$
Joins

Combine tuples from two relations if the pair of tuples satisfies some constraint.
Equivalent to Cartesian Product followed by a Select.

Natural Join

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

department \( \bowtie \) instructor:

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>salary</th>
<th>dept_name</th>
<th>building</th>
<th>budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>1010</td>
<td>Srinivasan</td>
<td>65000</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>1212</td>
<td>Wu</td>
<td>90000</td>
<td>Finance</td>
<td>Painter</td>
<td>120000</td>
</tr>
<tr>
<td>1515</td>
<td>Mozart</td>
<td>40000</td>
<td>Music</td>
<td>Packard</td>
<td>80000</td>
</tr>
<tr>
<td>2222</td>
<td>Einstein</td>
<td>95000</td>
<td>Physics</td>
<td>Watson</td>
<td>70000</td>
</tr>
<tr>
<td>3234</td>
<td>El Said</td>
<td>60000</td>
<td>History</td>
<td>Painter</td>
<td>50000</td>
</tr>
<tr>
<td>3345</td>
<td>Gold</td>
<td>87000</td>
<td>Physics</td>
<td>Watson</td>
<td>70000</td>
</tr>
<tr>
<td>4556</td>
<td>Katz</td>
<td>75000</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>5898</td>
<td>Calieri</td>
<td>62000</td>
<td>History</td>
<td>Painter</td>
<td>50000</td>
</tr>
<tr>
<td>7654</td>
<td>Singh</td>
<td>80000</td>
<td>Finance</td>
<td>Painter</td>
<td>120000</td>
</tr>
<tr>
<td>7676</td>
<td>Crick</td>
<td>72000</td>
<td>Biology</td>
<td>Watson</td>
<td>90000</td>
</tr>
<tr>
<td>8382</td>
<td>Brandt</td>
<td>92000</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>9834</td>
<td>Kim</td>
<td>80000</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)

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...**

CMSC424: Database Design

**SQL**

Professor: Pete Keleher
keleher@cs.umd.edu
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 relation.
  - Security and authorization information for each relation.
  - The physical storage structure of each relation on disk.

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

```sql
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),
    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.