Outline

- 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 Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics

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

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

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?
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)
  - 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
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: 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)
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>$\alpha$</td>
<td>$\alpha$</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$\beta$</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>12</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>23</td>
<td>10</td>
<td></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>$\alpha$</td>
<td>$\alpha$</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>23</td>
<td>10</td>
<td></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>$\alpha$</td>
<td>$\alpha$</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$\beta$</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>12</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>23</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

\( r \cup s: \)

\( r - s: \)

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

Must be compatible schemas

What about intersection?

Can be derived

Cartesian Product

Combine tuples from two relations

If one relation contains \( N \) tuples and the other contains \( M \) tuples, the result would contain \( N \times M \) tuples

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

Relation \( r, s \)

\[
\begin{array}{ccc}
A & B & C & D & E \\
\alpha & 1 & 10 & a \\
\beta & 2 & 10 & a \\
\gamma & 1 & 20 & b \\
\end{array}
\quad
\begin{array}{cccc}
A & B & C & D & E \\
\alpha & 1 & 1 & 10 & a \\
\alpha & 1 & \beta & 20 & b \\
\alpha & 1 & \gamma & 10 & b \\
\beta & 2 & \alpha & 10 & a \\
\beta & 2 & \beta & 10 & a \\
\beta & 2 & \gamma & 10 & b \\
\end{array}
\]

\( r \times s: \)
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></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

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

r ⋈ A = C s:

<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>α</td>
<td>1</td>
<td>α</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>1</td>
<td>10</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>20</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>γ</td>
<td>10</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Restriction on attributes A

<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>α</td>
<td>1</td>
<td>10</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>β</td>
<td>1</td>
<td>20</td>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>γ</td>
<td>10</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.
  
Example:

\[ \rho_X(E) \]

returns the expression \( E \) under the name \( X \)

If a relational-algebra expression \( E \) has arity \( n \), then

\[ \rho_X(A_1, A_2, ..., A_n)(E) \]

returns the result of expression \( E \) under the name \( X \), and with the attributes renamed to \( A_1, A_2, ..., A_n \).
Additional Operators

- Set intersection (∩)
  - $r \cap s = r - (r - s)$
  - SQL Equivalent: intersect

- Assignment (←)
  - A convenient way to right complex RA expressions
  - Essentially for creating “temporary” relations
    - $temp1 \leftarrow \Pi_{R \cdot S}(r)$
  - SQL Equivalent: “create table as...”

Additional Operators: Joins

- Natural join (⋈)
  - A Cartesian product with equality condition on common attributes
  - Example:
    - if $r$ has schema $R(A, B, C, D)$ and if $s$ has schema $S(E, B, D)$
    - Common attributes: $B$ and $D$
    - Then:
      
      \[
      r \bowtie s = \Pi_{r.A, r.B, r.C, r.D, s.E} (\sigma_{r.B = s.B \land r.D = s.D} (r \times s))
      \]

  - SQL Equivalent:
    - select $r.A, r.B, r.C, r.D, s.E$ from $r, s$ where $r.B = s.B$ and $r.D = s.D$, OR
    - select * from $r$ natural join $s$
Additional Operators: Joins

- **Equi-join**
  - A join that only has equality conditions

- **Theta-join** \( (\bowtie_\theta ) \)
  - \( r \bowtie_\theta s = \sigma_\theta(r \times s) \) (combines cartesian and select in single statement)

- **Left outer join** \( (\bow年\) )
  - Say \( r(A, B), s(B, C) \)
  - We need to somehow find the tuples in \( r \) that have no match in \( s \)
  - What is this? \( (r - \pi_{r.A, r.B}(r \bow年 s)) \)
  - \( r \bow年 s = (r \bow年 s) \cup \rho_{\text{temp}(A, B, C)}((r - \pi_{r.A, r.B}(r \bow年 s)) \times \{\text{NULL}\}) \)

---

Additional Operators: Join Variations

*Tables: r(A, B), s(B, C)*

<table>
<thead>
<tr>
<th>name</th>
<th>Symbol</th>
<th>SQL Equivalent</th>
<th>RA expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>cross product</td>
<td>(\times)</td>
<td>select * from r, s;</td>
<td>(r \times s)</td>
</tr>
<tr>
<td>natural join</td>
<td>(\bowland)</td>
<td>natural join</td>
<td>(\pi_{r.A, r.B, s.C}\sigma_{r.B = s.B}(r \times s))</td>
</tr>
<tr>
<td>equi–join</td>
<td>(\bowdird_\theta)</td>
<td>(\text{(theta must be equality)})</td>
<td></td>
</tr>
<tr>
<td>theta join</td>
<td>(\bowdird_\theta)</td>
<td>from .. where (\theta);</td>
<td>(\sigma_\theta(r \times s))</td>
</tr>
<tr>
<td>left outer join</td>
<td>(r \bow年 s)</td>
<td>left outer join (with “on”);</td>
<td>(see previous slide)</td>
</tr>
<tr>
<td>full outer join</td>
<td>(r \bow年 s)</td>
<td>full outer join (with “on”);</td>
<td>–</td>
</tr>
<tr>
<td>(left) semijoin</td>
<td>(r \bowtimes s)</td>
<td>none</td>
<td>(\pi_{r.A, r.B}(r \bow年 s))</td>
</tr>
<tr>
<td>(left) antijoin</td>
<td>(r \bowtriangledown s)</td>
<td>none</td>
<td>(r - \pi_{r.A, r.B}(r \bowyear s))</td>
</tr>
</tbody>
</table>
**Additional Operators: Division**

- Assume \( r(R), s(S) \), for queries where \( S \subseteq R \):
  - \( r \div s \)
- Think of it as “opposite of Cartesian product”
  - \( r \div s = t \iff t \times s \subseteq r \)

<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>β</td>
<td>2</td>
<td>α</td>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>B</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>

\[ A \quad \div \quad C \quad D \quad E \]

\[ A \quad B \]

\[ = \]

\[ C \quad D \quad E \]

\[ A \quad 1 \]

\[ β \quad 2 \]

\[ α \quad 10 \quad a \]

\[ β \quad 10 \quad a \]

\[ β \quad 20 \quad b \]

\[ γ \quad 10 \quad b \]

**Relational Algebra Examples**

Find all loans of over $1200:

\[ \sigma_{\text{amount} > 1200} \text{(loan)} \]

Find the loan number for each loan of an amount greater than $1200:

\[ \Pi_{\text{loan-number}} (\sigma_{\text{amount} > 1200} \text{(loan)}) \]

Find names of all customers who have a loan, account, or both, from the bank:

\[ \Pi_{\text{customer-name}} (\text{borrower}) \cup \Pi_{\text{customer-name}} (\text{depositor}) \]
Relational Algebra Examples

Find names of customers who have a loan and an account at bank:

\[ \Pi_{\text{customer-name}} (\text{borrower}) \cap \Pi_{\text{customer-name}} (\text{depositor}) \]

Find names of customers who have a loan at the Perryridge branch:

\[ \Pi_{\text{customer-name}} (\sigma_{\text{branch-name} = \text{"Perryridge"}} (\sigma_{\text{borrower.loan-number} = \text{loan.loan-number}} (\text{borrower x loan})))) \]

Find largest account balance(balance), assume \{(1), (2), (3)\}

Rename the account relation to d

\[ \Pi_{\text{balance}} (\text{account}) - \Pi_{\text{account.balance}} (\sigma_{\text{account.balance} < d.\text{balance}} (\text{account x } \rho_d (\text{account}))) \]
Generalized Projection

- Extends the projection operation by allowing arithmetic functions to be used in the projection list.
  \[ \Pi_{F_1, F_2, ..., F_n}(E) \]

- \( E \) is any relational-algebra expression
- Each of \( F_1, F_2, ..., F_n \) are arithmetic expressions involving constants and attributes in the schema of \( E \).
- Given relation \( instructor(ID, name, dept\_name, salary) \) where salary is annual salary, get the same information but with monthly salary
  \[ \Pi_{ID, name, dept\_name, salary/12}(instructor) \]