Today’s Plan

- SQL (Chapter 3, 4)
  - Views (4.2)
  - Triggers (5.3)
  - Transactions (4.3)
  - Integrity Constraints (4.4)
  - Functions and Procedures (5.2), Authorization (4.6), Ranking (5.5)
  - Return to / Finishing the Relational Algebra
  - E/R Diagrams

SQL Functions

- Function to count number of instructors in a department
  ```sql
  create function dept_count (dept_name varchar(20))
  returns integer
  begin
    declare d_count integer;
    select count(*) into d_count
    from instructor
    where instructor.dept_name = dept_name
    return d_count;
  end
  ```

- Can use in queries:
  ```sql
  select dept_name, budget
  from department
  where dept_count (dept_name) > 12
  ```
SQL Procedures

- Same function as a procedure:

  ```sql
  create procedure dept_count_proc (in dept_name varchar(20), out d_count integer)
  begin
    select count(*) into d_count
    from instructor
    where instructor.dept_name = dept_count_proc.dept_name
  end
  ```

- But use differently:

  ```sql
  declare d_count integer;
  call dept_count_proc( 'Physics', d_count);
  ```

HOWEVER: Syntax can be wildly different across different systems

- Was put in place by DBMS systems before standardization
- Hard to change once customers are already using
- This example **NOT** valid in your version of postgresql

Recursion in SQL

- Example: find which courses are a prerequisite, whether directly or indirectly, for a specific course

  ```sql
  with recursive rec_prereq(course_id, prereq_id) as (
    select course_id, prereq_id
    from prereq
    union
    select rec_prereq.course_id, prereq.prereq_id,
    from rec_prereq, prereq
    where rec_prereq.prereq_id = prereq.course_id
  )
  select *
  from rec_prereq;
  ```

Makes SQL Turing Complete (i.e., you can write any program in SQL)

But: Just because you can, doesn’t mean you should
**Ranking**

- Ranking is done in conjunction with an order by specification.

  - Consider: `student_grades(ID, GPA)`

  - Find the rank of each student.

```sql
select ID, rank() over (order by GPA desc) as s_rank
from student_grades
order by s_rank
```

- Equivalent to:

```sql
select ID, (1 + (select count(*)
    from student_grades B
    where B.GPA > A.GPA)) as s_rank
from student_grades A
order by s_rank;
```

---

**Authorization/Security**

- GRANT and REVOKE keywords
  - `grant select on instructor to U1, U2, U3`
  - `revoke select on branch from U1, U2, U3`

- Can provide select, insert, update, delete privileges

- Can also create “Roles” and do security at the level of roles

- Some databases support doing this at the level of individual “tuples”
  - PostgreSQL: [https://www.postgresql.org/docs/10/ddl-rowsecurity.html](https://www.postgresql.org/docs/10/ddl-rowsecurity.html)
Today’s Plan

> SQL (Chapter 3, 4)
>   - Views (4.2)
>   - Triggers (5.3)
>   - Transactions (4.3)
>   - Integrity Constraints (4.4)
>   - Functions and Procedures (5.2), Recursive Queries (5.4), Authorization (4.6), Ranking (5.5)
>   - Return to / Finishing the Relational Algebra
>   - E/R Diagrams

Relational Algebra, Again

> Procedural language
> Six basic operators
>   - select
>   - project
>   - union
>   - set difference
>   - Cartesian product
>   - rename
> Set semantics

The operators take one or more relations as inputs and give a new relation as a result.
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 \( A_1, A_2, ..., A_n \).

Relational Algebra

- Those are the basic operations

- What about SQL Joins?
  - Compose multiple operators together
    \[ \sigma_{A=C}(r \times s) \]

- Additional Operations
  - Set intersection
  - Natural join
  - Division
  - Assignment
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
    - $temp1 \leftarrow \Pi_{R \cap S}(r)$
  - 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 (⋈θ )**
  - \( r \bowtie_\theta s = \sigma_\theta(r \times s) \) (combines cartesian and select in single statement)

- **Left outer join (⟕)**
  - Say \( r(A, B), s(B, C) \)
  - We need to somehow find the tuples in \( r \) that have no match in \( s \)
  - Consider: \( (r - \pi_{r.A, r.B}(r \bowtie s)) \)
  - We are done:
    \[
    (r \bowtie s) \cup \rho_{temp (A, B, C)}( (r - \pi_{r.A, r.B}(r \bowtie s)) \times \{(NULL)\})
    \]

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}(r \bowland s) )</td>
</tr>
<tr>
<td>equi–join</td>
<td>( \bowland_\theta )</td>
<td>(theta must be equality)</td>
<td></td>
</tr>
<tr>
<td>theta join</td>
<td>( \bowland_\theta )</td>
<td>from .. where ( \theta );</td>
<td>( \sigma_\theta(r \times s) )</td>
</tr>
<tr>
<td>left outer join</td>
<td>( r \bowland s )</td>
<td>left outer join (with “on”)</td>
<td>(see previous slide)</td>
</tr>
<tr>
<td>full outer join</td>
<td>( r \bowland s )</td>
<td>full outer join (with “on”)</td>
<td>–</td>
</tr>
<tr>
<td>(left) semijoin</td>
<td>( r \times s )</td>
<td>none</td>
<td>(\pi_{r.A, r.B}(r \bowland s))</td>
</tr>
<tr>
<td>(left) antijoin</td>
<td>( r \bowland s )</td>
<td>none</td>
<td>( r - \pi_{r.A, r.B}(r \bowland 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”
  - \( r \div s = t \text{ iff } t \times s \subseteq r \)

\[
\begin{array}{|c|c|c|c|c|}
\hline
A & B & C & D & E \\
\hline
\alpha & 1 & \alpha & 10 & a \\
\alpha & 1 & \beta & 10 & a \\
\alpha & 1 & \beta & 20 & b \\
\alpha & 1 & \gamma & 10 & b \\
\beta & 2 & \alpha & 10 & a \\
\beta & 2 & \beta & 10 & a \\
\beta & 2 & \beta & 20 & b \\
\beta & 2 & \gamma & 10 & b \\
\hline
\end{array}
\]

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

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

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 the largest account balance:
*Rename the account relation to d*
\[ \Pi_{\text{balance}} (\text{account}) - \Pi_{\text{account.balance}} (\sigma_{\text{account.balance} < d.\text{balance}} (\text{account x } \rho_{d} (\text{account}))) \]

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.

\[ \Pi_{F_1, F_2, \ldots, F_n}(E) \]

- \( E \) is any relational-algebra expression

- Each of \( F_1, F_2, \ldots, F_n \) are arithmetic expressions involving constants and attributes in the schema of \( E \).

- Given relation \( \text{instructor}(ID, \text{name}, \text{dept\_name}, \text{salary}) \) where salary is annual salary, get the same information but with monthly salary

\[ \Pi_{ID, \text{name}, \text{dept\_name}, \text{salary}/12}(\text{instructor}) \]

Aggregate Functions and Operations

- **Aggregation function** takes a collection of values and returns a single value as a result.
  
  - \( \text{avg} \): average value
  - \( \text{min} \): minimum value
  - \( \text{max} \): maximum value
  - \( \text{sum} \): sum of values
  - \( \text{count} \): number of values

- **Aggregate operation** in relational algebra

\[ G_1, G_2, \ldots, G_n \bar{G} F_1(A_i), F_2(A_2), \ldots, F_n(A_n)(E) \]

\( E \) is any relational-algebra expression

- \( G_1, G_2, \ldots, G_n \) is a list of attributes on which to group (can be empty)
- Each \( F_i \) is an aggregate function
- Each \( A_i \) is an attribute name

- Note: Some books/articles use \( \gamma \) instead of \( \bar{G} \) (Calligraphic \( G \))
Aggregate Operation – Example

- Relation $r$:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>$\alpha$</td>
<td>7</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$\beta$</td>
<td>7</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>3</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>10</td>
</tr>
</tbody>
</table>

$G_{\text{sum}(c)}(r)$

<table>
<thead>
<tr>
<th>sum(c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
</tr>
</tbody>
</table>

Aggregate Operation – Example

- Find the average salary in each department $\text{ dept}_{\text{name}} G_{\text{avg}(\text{salary})}(\text{instructor})$

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>76766</td>
<td>Crick</td>
<td>Biology</td>
<td>72000</td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>Comp. Sci.</td>
<td>75000</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>Comp. Sci.</td>
<td>92000</td>
</tr>
<tr>
<td>96345</td>
<td>Kim</td>
<td>Elec. Eng.</td>
<td>80000</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
</tr>
<tr>
<td>76543</td>
<td>Singh</td>
<td>Finance</td>
<td>80000</td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>History</td>
<td>60000</td>
</tr>
<tr>
<td>58583</td>
<td>Califieri</td>
<td>History</td>
<td>62000</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>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dept_name</th>
<th>avg_salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biology</td>
<td>72000</td>
</tr>
<tr>
<td>Comp. Sci.</td>
<td>77333</td>
</tr>
<tr>
<td>Elec. Eng.</td>
<td>80000</td>
</tr>
<tr>
<td>Finance</td>
<td>85000</td>
</tr>
<tr>
<td>History</td>
<td>61000</td>
</tr>
<tr>
<td>Music</td>
<td>40000</td>
</tr>
<tr>
<td>Physics</td>
<td>91000</td>
</tr>
</tbody>
</table>
Aggregate Functions (Cont.)

- Result of aggregation does not have a name
  - Can use rename operation to give it a name
  - For convenience, we permit renaming as part of aggregate operation

\[ \text{dept\_name } \bigstar \text{avg(salary) as avg\_sal}^{\text{instructor}} \]

Modification of the Database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating
- All these operations can be expressed using the assignment operator

\[
\begin{align*}
temp1 & \leftarrow R \times S \\
temp2 & \leftarrow \sigma_{r.A_1 = s.A_1 \land r.A_2 = s.A_2 \land \ldots \land r.A_n = s.A_n} (temp1) \\
result & = \Pi_{R \cup S} (temp2)
\end{align*}
\]

The result of \( R \times S \) potentially has duplicated attributes. For example, \( r(A,B) \times s(B,C) \) results in tuples w/ attributes \((A, B, B, C)\). \( \Pi_{R \cup S} \) gets rid of the extra \( B \). Duplicated tuples are an entirely different thing, and are not present in the relational algebra.
Multiset Relational Algebra

- Pure relational algebra removes all duplicates
  - e.g. after projection
- Multiset relational algebra retains duplicates, to match SQL semantics
  - SQL duplicate retention was initially for efficiency, but is now a feature
- Multiset relational algebra defined as follows
  - selection: has as many duplicates of a tuple as in the input, if the tuple satisfies the selection
  - projection: one tuple per input tuple, even if it is a duplicate
  - cross product: If there are $m$ copies of $t_1$ in $r$, and $n$ copies of $t_2$ in $s$, there are $m \times n$ copies of $t_1.t_2$ in $r \times s$
  - Other operators similarly defined
    - E.g. union: $m + n$ copies, intersection: $\min(m, n)$ copies
    - difference: $\min(0, m - n)$ copies

Today’s Plan

- Quiz 3
- Exam
- Entity-Relationship Diagrams
Terms

• **monotonic** queries
  • Do not lose tuples as more data arrives (think streaming queries)

• **table function**
  • returns set of tuples
  • used anywhere a table is used

Quiz 3: tough problems

• 2. `( (NULL = 20) or (10 = 10) ) and ( (NULL = 10) is unknown)`
  • true

• 9. “select A, max(B) from R” mixing scalar and non-scalar (no groupby for A)

• 12.

> Are the following two queries equivalent? Why or Why not? Assume R.a is an integer attribute.
> 1. select * from R where R.a > 1;
> 2. (select * from R) except (select * from R where R.a <= 1);

• Check what happens if all A are nulls
Quiz 3: tough problems

Q15
0.2 Points

For three relations R(A, B), S(B, C), T(C, D), write relational algebra expressions to generate the following relations:

1. Q1(A, D) where R and S are joined on condition R.B > S.B, and S and T have a natural join.
2. Q2(A, C) to find all (A, C) pairs such that R.B = S.B, and S.C does not have a matching tuple in T.

In both cases, use only the basic relational operations.

EXPLANATION

Q1 ← \( \pi_{A,D} (\sigma_{R.B > S.B}(R \times S) \bowtie T) \)
Q2 ← \( \pi_{A,C} ((R \bowtie S) \bowtie T) \)
Entity-Relationship Model

Two key concepts

• **Entities:**
  - An object that *exists* and is *distinguishable* from other objects
    - Examples: Bob Smith, BofA, CMSC424
  - Have **attributes** (people have names and addresses)
  - Form **entity sets** with other entities of the same type that share the same properties
    - Set of all people, set of all classes
  - Entity sets may overlap
    - Customers and Employees
Two key concepts

- **Relationships:**
  - Relate 2 or more entities
    - E.g. Bob Smith *has account at* College Park Branch
  - Form *relationship sets* with other relationships of the same type that share the same properties
    - Customers *have accounts at* Branches
  - Can have attributes:
    - *has account at* may have an attribute *start-date*
  - Can involve more than 2 entities
    - Employee *works at* Branch *at* Job

**Baby ER Diagrams** (illustration only, do not use)

- Rectangles: entity sets
- Diamonds: relationship sets
- Ellipses: attributes
Rest of the class

- Details of the ER Model
  - How to represent various types of constraints/semantic information etc.

- Design issues

- A detailed example

Next: Relationship Cardinalities

- We may know:
  - One customer can only open one account
  - OR
  - One customer can open multiple accounts

- Representing this is important

- Why?
  - Better manipulation of data
    - If former, can store the account info in the customer table
  - Can enforce such a constraint
    - “Application logic will handle it” NOT GOOD
  - If not represented in conceptual model, domain knowledge can easily be lost
Mapping Cardinalities

- Express the number of entities to which another entity can be associated via a relationship set
- Most useful in describing binary relationship sets
- N-ary relationships?
  - More complicated
  - Details in the book

- One-to-One
- One-to-Many
- Many-to-One
- Many-to-Many
Next: Types of Attributes

- Simple vs Composite
  - Single value per attribute?
    - Are parts accessed separately?
    - e.g. accessing first and last names from name

- Single-valued vs Multi-valued
  - E.g. Phone numbers are multi-valued

- Derived
  - If date-of-birth is present, age can be derived
  - Can help in avoiding redundancy, enforcing constraints etc...