Relational Algebra

Introduction

1. Data Abstraction
    - Physical level - how data are actually stored
    - Logical level - what data are stored and what relationships exist among those data
    - View level - describes only part of the entire database
2. Instances and Schemas
    - The collection of information stored in the database at a particular moment is called an **instance** of the database.
    - The overall design of the database is called the database **schema**.
3. Data Models

    Data model is a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

    - Relational Model

        Use a collection of tables to represent both data and the relationships among those data

    - Entity-Relationship Model

        Use a collection of basic objects, called entities, and relationships among these objects.

    - Object-Based Data Model
    - Semistructured Data Model
4. Database Languages

    Data-Manipulation Language

    DML is a language that enables users to access or manipulate data as organized by the appropriate data model, including:

    - Retrieval
    - Insertion
    - Deletion
    - Modification

    Two types:

    - Procedural DMLs - need to specify **what** data are needed and **how** to get those data
    - Declaration DMLs - need to specify **what** data are needed without specifying how to get those data

    Data-Definition Language

    DDL is used to specify additional properties of the data. The data values stored in the database must satisfy certain *consistency constrains*, including:

    - *Domain Constraints* - A domain of possible values must be associated with every attribute (for example, integer types, character types, date/time types).
    - *Referential Integrity* - a value appears in one relation for a given set of attributes also appears in a certain set of attributes in another relation.
    - *Assertions* - any condition that the database must always satisfy
    - *Authorization* - differentiate among the users as far as the type of access they are permitted on various data values in the database.

5. Relational Databases

   A relational database is based on the relational model and uses a collection of tables to represent both data and the relationships among those data. It also includes a DML and DDL .

   The *SQL query language* is nonprocedural. A query takes as input several tables (possibly only one) and always **returns a single table**.

6. Database Design

   Three Phases:

   - Requirements analysis
   - Data modeling
   - *Normalization*

The Relational Algebra

1. Operations
   - The ***Select*** Operation
     - The ***select*** operation selects tuples that satisfy a given predicate.
     - Affected on Row

     > $R_1 := \sigma_C (R_2)$
     >
     > - C is a condition (as in "if" statements) that refers to attributes of R_2.
     > - R_1 is all those tuples of R_2 that satisfy C .

   - The ***Project*** Operation
     - Affected on Column

     > $R_1 := \pi_L(R_2)$
     >
     > - L is a list of attributes from the schema of R2.
     > - R_1 is constructed by looking at each tuple of R2, extracting the attributes on list L , in the order specified, and creating from those components a tuple for R1.
     > - Eliminate duplicate tuples, if any.

   - The ***Union*** Operation

     For a union operation r **U** s to be valid, we require that two conditions hold:

     1. The relations r and s must be of the same arity. That is, they must have the same number of attributes.
     2. The domains of the i^{th} attribute of r and the i^{th} attribute of s must be the same, for all i.

   - The ***Set-Difference*** Operation

     For a set-difference operation r − s to be valid, we require that the relations r and s be of the same arity, and that the domains of the i^{th} attribute of r and the i^{th} attribute of s be the same, for all i.

   - The ***Cartesian-Product*** Operation
     - Combine information from any two relations.
     - In general, if we have relations r_1 (R_1) and r_2(R_2), then $r_1 \times r_2$ is a relation whose schema is the concatenation of R_1 and R_2. Relation R contains all tuples t for which

there is a tuple t_1 in r_1 and a tuple t_2 in r_2 for which $t[R_1] = t_1[R_1]$ and $t[R_2] = t_2[R_2]$

- The ***Rename*** Operation
  - Given a relational-algebra expression E, the expression ρ_x (E) returns the result of expression E under the name x.
  - A second form of the rename operation is as follows: Assume that a relational-algebra expression E has arity n. Then, the expression $\rho_x(A_1, A_2, \ldots, A_n)$ (E) returns the result of expression E under the name x, and with the attributes renamed to $A_1, A_2, \ldots, A_n$.

2. Additional Relational-Algebra Operations
   1. The Set-Intersection Operation - ∩
   2. The ***Natural Join*** Operation
      - The ***natural join*** is a binary operation that allows us to combine certain selections and a Cartesian product into one operation.
        - The natural-join operation forms a Cartesian product of its two arguments, performs a selection forcing equality on those attributes that appear in both relation schemas, and finally removes duplicate attributes.

          $$r \bowtie s = \Pi_{R \cup S} (\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n} (r \times s))$$

          where $R \cap S = \{A_1, A_2, \ldots, A_n\}$
      - ***Theta-Join*** and ***Equi-Join***
        - The *θ*-join is a binary operator that is written as $R \bowtie_{a\,\theta\,b} S$ or $R \bowtie_{a\,\theta\,v} S$ where *a* and *b* are attribute names, *θ* is a binary relational operator in the set {<, ≤, =, ≠, >, ≥}, *v* is a value constant, and *R* and *S* are relations.

          $$R \bowtie_\theta S = \sigma_\theta(R \times S)$$

        - In case the operator *θ* is the equality operator (=) then this join is also called an **equi-join**.
        - **Natural join** is a special case of **equi-join**, in which the two attributes' name are the same.
   3. The ***Assignment*** Operation
      - The assignment operation, denoted by ←,works like assignment in a programming language.
      - The natural join r⋈s can be written as:

        $$\begin{aligned} temp1 &\leftarrow R \times S \\ temp2 &\leftarrow \sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n} (temp1) \\ result &= \Pi_{R \cup S} (temp2) \end{aligned}$$

   4. ***Outer join*** Operations

      The outer-join operation is an extension of the join operation to deal with missing information.

      1. Left Outer Join(R ⟕ S)

         All the tuples from the Left relation, R, are included in the resulting relation. If there are tuples in R without any matching tuple in the Right relation S, then the S-attributes of the resulting relation are made `NULL`.

      2. Right Outer Join: (R ⟖ S)

All the tuples from the Right relation, S, are included in the resulting relation. If there are tuples in S without any matching tuple in R, then the R-attributes of resulting relation are made `NULL`.

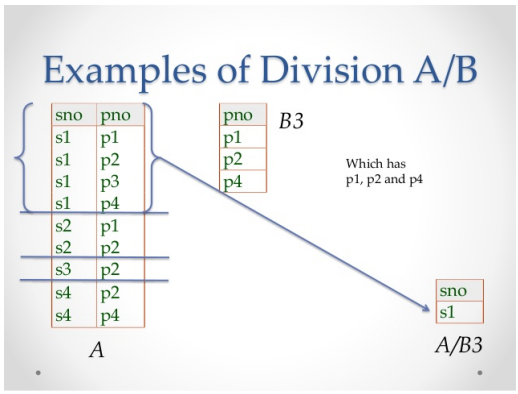
3. Full Outer Join: (R ⟗ S)

   All the tuples from both participating relations are included in the resulting relation. If there are no matching tuples for both relations, their respective unmatched attributes are made `NULL`.

5. **Division** Operations

r ÷ s is used when we wish to express queries with "all", for example:

- "Which persons have a loyal customer's card at ALL the clothing boutiques in town X?"
- "Which persons have a bank account at ALL the banks in the country?"
- "Which students are registered on ALL the courses given by Soini?"



- Select `sno` in A, which will give us A' (s1,s2,s3,s4);
- A'' = A' X B3;
- A''' = A - A''
- Select `sno` from A''', denoted as r1;
- Select `sno` from A, denoted as r2
- Result is r2 - r1

> Example:
>
> We want to find female students who take all the courses that student 40101 is taking.
>
> **All Female Students and the courses they are taking**
>
sid	sname	course_id
40112	Brita	456306
40112	Brita	456302
40113	Ann-Helen	456304
40113	Ann-Helen	456306
40128	Siru	G555
40128	Siru	456306
40128	Siru	456302
40240	Sara	456304
40240	Sara	456306
40240	Sara	456302
>
> | course_id |
> |---|
> | G55 |
> | 456306 |
> | 456302 |
>
> - $r \div s$ is defined as $\Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$
>
>   r - All Female Students and the courses they are taking
>
>   s - The course id student 40101 is taking.
>
>   - $\Pi_{R-S}(r) = \Pi_{sid,sname}$

sid	sname
40112	Brita
40113	Ann-Helen
40128	Siru
40240	Sara

- $(\Pi_{R\text{-}S}(r) \times s) = ((\Pi_{R\text{-}sid,sname}(r) \times s)$ all possible combination of students and the course 4101 is taking.

sid	sname	course_id
40112	Brita	456306
40112	Brita	G55
40112	Brita	456302
40113	Ann-Helen	456304
40113	Ann-Helen	40302
40113	Ann-Helen	G55
40113	Ann-Helen	456306
40128	Siru	G555
40128	Siru	456306
40128	Siru	456302
40240	Sara	456304
40240	Sara	456306
40240	Sara	456302
40240	Sara	G55

- $\Pi_{R\text{-}S,S}(r) = \Pi_{sid,sname,course\_id}(r)$ the actual course those female students are taking.

sid	sname	course_id
40112	Brita	456306
40112	Brita	456302
40113	Ann-Helen	456304
40113	Ann-Helen	456306
40128	Siru	G555
40128	Siru	456306
40128	Siru	456302
40240	Sara	456304
40240	Sara	456306
40240	Sara	456302

- $(\Pi_{R\text{-}S}(r) \times s) - \Pi_{R\text{-}S,S}(r)$

sid	sname	course_id
40112	Brita	G55
40113	Ann-Helen	456304
40113	Ann-Helen	G55
40113	Ann-Helen	G55
40240	Sara	456304
40240	Sara	G55

- $\Pi_{R\text{-}S}((\Pi_{R\text{-}S}(r) \times s) - \Pi_{R\text{-}S,S}(r))$

sid	sname
40112	Brita
40113	Ann-Helen
40240	Sara

- $\Pi_{R\text{-}S}(r) - \Pi_{R\text{-}S}((\Pi_{R\text{-}S}(r) \times s) - \Pi_{R\text{-}S,S}(r))$

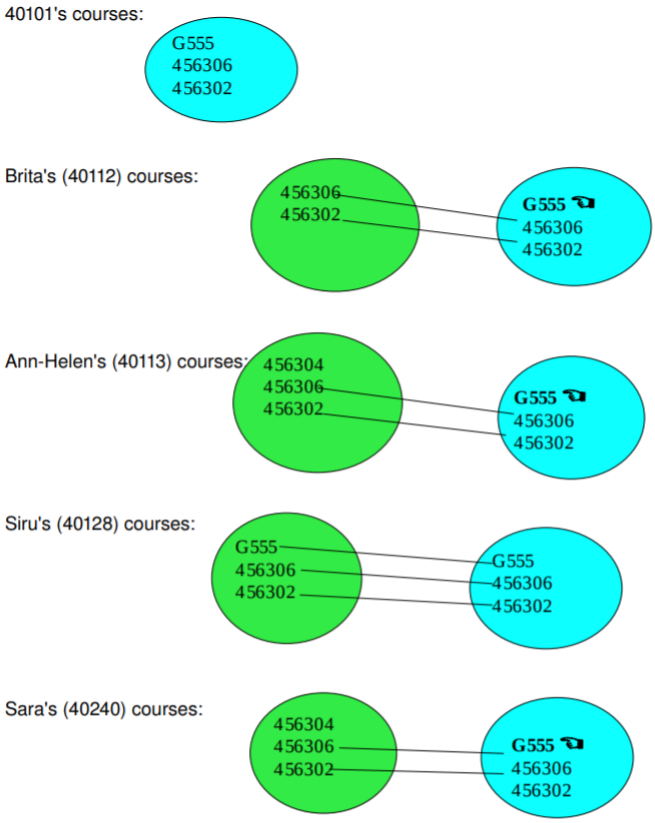
40128	Siru

Translate the previous example into SQL:

1. `Not exists` combined with `not in`: (“There may not be a course that 40101 takes that is not among the courses taken by the girl in question”)

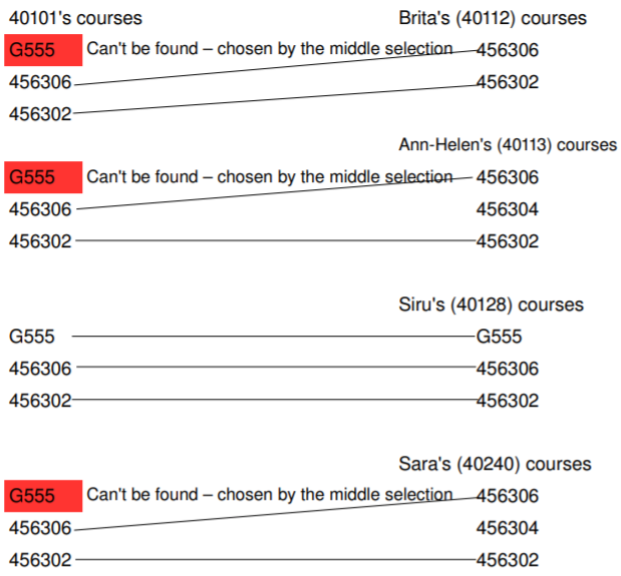
```
select distinct sid, sname
from student as R
where gender = 'female' and
      not exists (select course_id
                  from courses
                  where sid = 40101 and
                        course_id not in (select course_id
                                          from course as R2
                                    where R.sid = R2.sid));
```

   For the female student to be part of the result, the `not exists` clause for this student must be true (the list after the not exists must remain empty).



2. Two times not `exists`: (“There may not be a course that 40101 takes that is not taken by the current girl”)

```
select distinct sid, sname
from student as R
where gender = 'Female' and
      not exists (select course_id
                  from course as S
                  where sid = 40101 and
                        not exists (select course_id
                                    from course as R2
                                    where R.sid = R2.sid and
                                          S.course_id =
R2.course_id ));
```



3. "Count and compare"

   The idea behind this method is to count how many different values there are in the divisor (i. e. all the values that one should have to be a part of the result). When this is applied to our example we first count how many courses 40101 takes. Then we count how many of these courses our female students take. If we get the same result, the girl in question has all the courses necessary and will appear in the result.

```
select sid, sanme
      from course natural join student
      where gender = 'Female' and
            course_id in (select course_id
                          from course
                          where sid = 40101)
      group by sid
      having count(course_id) = (select count(*)
                                 from (select course_id
                                       from course
                                       where sid = 40101) as
tab);
```

From Relational Model to SQL

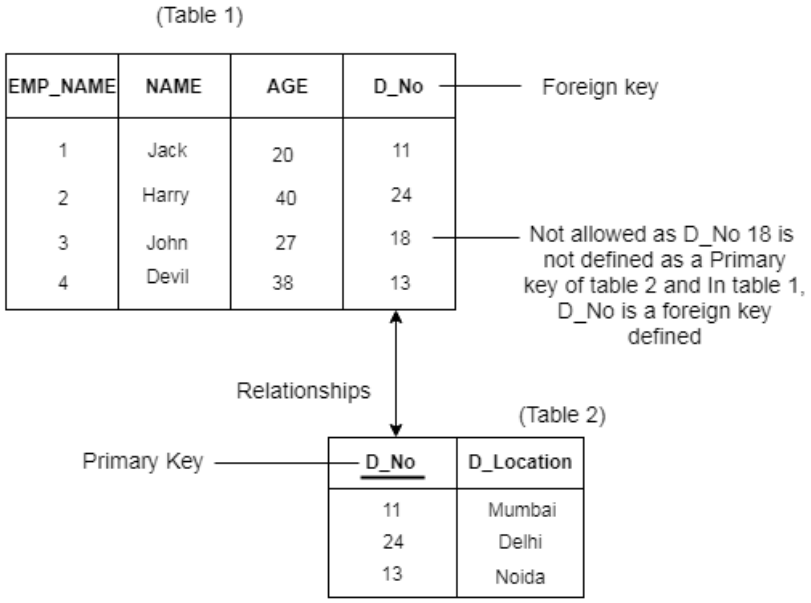
1. Structure of Relational Databases
   - ***Relation*** - Table
   - ***Tuple*** - Row
   - ***Attribute*** - Column
   - ***Relation Instance*** - a specific instance of a relation, i.e., containing a specific set of rows
   - ***Domain of attributes*** - a set of permitted values for each attribute
     - A domain is ***atomic*** if elements of the domain are considered to be indivisible units.
   - The ***null*** value is a special value that signifies that the value is unknown or does not exist.

2. Database Schema

- ***Database schema*** is the logical design of the database
  - We need a relation to describe the association between instructors and the class sections that they teach. The relation schema to describe this association is `teaches ( ID , course id, sec id, semester, year)`
- ***Database instance*** is a snapshot of the data in the database at a given instant in time
  - The relation filled with data

3. Keys

- ***Superkey*** is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation, e.g.,the *ID* attribute of the relation *instructor* is sufficient to distinguish one instructor tuple from another.
- **Candidate Key** can be any column or a combination of columns that can qualify as unique key in database. There can be multiple Candidate Keys in one table. Each Candidate Key can qualify as Primary Key.
- **Primary Key** is a column or a combination of columns that uniquely identify a record. Only one Candidate Key can be Primary Key.
  - A table can have multiple Candidate Keys that are unique as single column or combined multiple columns to the table. They are all candidates for Primary Key.
  - The primary key should be chosen such that its attribute values are never, or very rarely, changed.
  - If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s).
- ***Foreign Key*** is a column or a combination of columns whose values match a Primary Key in a different table.
  - The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table.
  - A ***referential integrity constraint*** requires that if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.



4. Relational Query Languages

- A ***query language*** is a language in which a user requests information from the database.
  - In a ***procedural language***, the user instructs the system to perform a sequence of operations on the database to compute the desired result.
    - Relational Algebra is a procedural language.
  - In a ***nonprocedural language***, the user describes the desired information without giving a specific procedure for obtaining that information.

5. Relational Operations
   - The ***relational algebra*** provides a set of operations that take one or more relations as input and return a relation as an output. Practical query languages such as SQL are based on the relational algebra, but add a number of useful syntactic features.

Symbol (Name)	Example of Use
σ	$\sigma_{\text{salary}>=85000}(instructor)$
(Selection)	Return rows of the input relation that satisfy the predicate.
Π	$\Pi_{ID,\,salary}(instructor)$
(Projection)	Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
$\bowtie$	$instructor \bowtie department$
(Natural join)	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
$\times$	$instructor \times department$
(Cartesian product)	Output all pairs of rows from the two input relations (regardless of whether or not they have the same values on common attributes)
$\cup$	$\Pi_{name}(instructor) \cup \Pi_{name}(student)$
(Union)	Output the union of tuples from the two input relations.