Lecture Notes for Lecture 4 of CS 5200 (Database Management Systems) for the Summer 1, 2019 session at the Northeastern University Silicon Valley Campus.

SQL Overview

Philip Gust,
Clinical Instructor
Department of Computer Science

Lecture 3 Review

- Relational algebra provides a theoretical foundation for relational databases, and their query languages, such as SQL.
- Relational algebra takes instances of relations as input and yields instance of relations as output.
- Relations in relational algebra are sets of tuples, so includes basic set operations, such as subset, superset, union, intersection, difference, and cartesian product.
- A query is both an *imperative command* that produce a relational result, and a *declarative statement* that describes a desired relational result, without stating how to achieve it.
- A relational expression represents an *intentional relation*, while an *extensional relation* is one whose values are explicitly given. The two are equivalent in relational algebra.

Lecture 3 Review

- We also reviewed the following operations that are basic to relational algebra and looked at examples for a database using the Relational Algebra Calculator (RelX).
  - Select (σ) : select tuples that satisfy predicate
  - Project ($\prod$) : project columns that satisfy predicate
  - Union ($\cup$) : binary union between relations
  - Set difference ($-$) : tuples in one relation but not other
  - Rename (ρ): name output relation
  - Order (τ): order output relation by attribute
  - Group (γ): aggregate attributes into groups
  - Various join operations
    - Cross Join (X)
    - Natural join ($\bowtie$)
    - Left Outer Join ($⟕$)
    - Right Outer Join ($⟖$)
    - Full Outer Join ($⟗$)
    - Left Semijoin ($\ltimes$)
    - Right Semijoin ($\rtimes$)
    - Antijoin ($\triangleright$)

SQL Overview

What is SQL

- SQL (Structured Query Language) is a programming language for relational databases. It is often described as a declarative language, but it also includes procedural elements.
- Although designed over relational algebra and tuple relational calculus, it does not entirely adhere to Codd's relational model.
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and an International Organization for Standardization (ISO) standard in 1987
- Despite such standards, most SQL code is not completely portable among different database systems.

SQL Overview

Components of SQL

- SQL provides datatypes corresponding to numbers, strings, dates, and other specialized types.
- SQL consists of many kinds of statements, which may be informally classed as *sub-languages*, including
  - A *data definition language* (DDL) for designing and modifying database schema.
  - A *data manipulation language* (DML) that enable SQL to store and retrieve data from a database
  - A *data query language* (DQL) for making queries on the data stored in a database
  - A data control language (DCL) for managing and controlling access to data stored in a database

SQL Overview

SQL Datatypes

- Exact Numeric Data Types

DATA TYPE	FROM	TO
bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
tinyint	0	255
bit	0	1
decimal	-10^38 +1	10^38 -1
numeric	-10^38 +1	10^38 -1
money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
smallmoney	-214,748.3648	+214,748.3647

SQL Overview

SQL Datatypes

- Approximate Numeric Data Types

DATA TYPE	FROM	TO
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

SQL Overview

SQL Datatypes

- Date and Time Data Types

DATA TYPE	FROM	TO
datetime	Jan 1, 1753	Dec 31, 9999
smalldatetime	Jan 1, 1900	Jun 6, 2079
date	Stores a date like June 30, 1991	
time	Stores a time of day like 12:30 P.M.	

datetime has 3.33 milliseconds accuracy where as smalldatetime has 1 minute accuracy.

SQL Overview

SQL Datatypes

- Character String Data Types

DATA TYPE & Description
char Maximum length of 8,000 characters.(Fixed length non-Unicode characters)
varchar Maximum of 8,000 characters.(Variable-length non-Unicode data).
varchar(max) Maximum length of 231characters, Variable-length non-Unicode data (SQL Server 2005 only).
text Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters.

Some databases do not support varchar without length

SQL Overview

SQL Datatypes

- Binary Data Types

DATA TYPE & Description
binary Maximum length of 8,000 bytes(Fixed-length binary data)
varbinary Maximum length of 8,000 bytes.(Variable length binary data)
varbinary(max) Maximum length of 231 bytes (SQL Server 2005 only). (Variable length Binary data)
image Maximum length of 2,147,483,647 bytes. (Variable length Binary Data)

SQL Overview

Data Definition Language (DDL)

Important data definition language statements include

- **CREATE DATABASE** - creates a new database
- **ALTER DATABASE** - modifies a database
- **CREATE TABLE** - creates a new table
- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

SQL Overview

Data Definition Language (DDL)

SQL uses the following set of commands to define database schema:

- CREATE
  - Creates new databases, tables and views from RDBMS.
    - CREATE DATABASE dbname
    - CREATE TABLE tblName (type1 field1 [, type2 field2 [, type3 field3 …]])
    - CREATE VIEW viewName AS selectStatement
  - **Example**:
    - CREATE database tutorialspoint;
    - CREATE table article (varchar(20) author, varchar(64) title, date pubDate);
    - CREATE view author_article AS SELECT author, title FROM article WHERE author = ’me’

SQL Overview

Data Definition Language (DDL)

SQL uses the following set of commands to define database schema:

- DROP
  - Drops commands, views, tables, and databases from RDBMS.
    - Drop object_type object_name;
  - **Example**:
    - Drop database publications;
    - Drop table article;
    - Drop view author_article;

SQL Overview

Data Definition Language (DDL)

SQL uses the following set of commands to define database schema:

- ALTER
  - Modifies database schema
    - ALTER object_type object_name parameters;
  - **Example**:
    - ALTER table article ADD subject varchar(64);

      adds an attribute in the relation **article** with the name **subject** of string type.

SQL Overview

Data Manipulation Language (DML)

Important Data Manipulation Language (DML) Statements

- **INSERT INTO** - inserts new data into a database
- **UPDATE** - updates data in a database
- **DELETE** - deletes data from a database

SQL Overview

Data Manipulation Language (DML)

SQL uses the following set of commands for all forms of data modification in the database:

- INSERT INTO/VALUES
  - Used for inserting values into the rows of a table (relation).
    - INSERT INTO table (column1 [, column2, column3 ...])
      VALUES (value1 [, value2, value3 ...])
    - INSERT INTO table VALUES (value1, [value2, ...])
  - Example:
    - INSERT INTO Author (Name, Subject)
      VALUES (‘anonymous’, ‘computers’);

SQL Overview

Example: Wikipedia – Relational Algebra Tables

```
create table Dept (
  DeptName varchar(16),
  Manager varchar(16)
);
create table Employee (
  Name varchar(16),
  EmpId int,
  DeptName varchar(16)
);

insert into Dept values ('Sales', 'Harriet');
insert into Dept values ('Production', 'Charles');

insert into Employee values ('Harry', 3415, 'Finance');
insert into Employee values ('Sally', 2241, 'Sales');
insert into Employee values ('George', 3401, 'Finance');
insert into Employee values ('Harriet', 2202, 'Sales');
insert into Employee values ('Tim', 1123, 'Executive');
```

SQL Overview

Data Manipulation Language (DML)

SQL uses the following set of commands for all forms of data modification in the database:

- UPDATE/SET/WHERE
  - Used for updating or modifying the values of columns in a table (relation).
    - UPDATE table_name
      SET column_name = value [, column_name = value ...] [WHERE condition]
  - **Example**:
    - UPDATE Author SET Name='Joe' WHERE Name='anonymous';

SQL Overview

Data Manipulation Language (DML)

SQL uses the following set of commands for all forms of data modification in the database:

- DELETE/FROM/WHERE
  - Used for removing one or more rows from a table (relation).
    - DELETE FROM table_name [WHERE condition];
  - **Example**:
    - DELETE FROM Author WHERE Name=‘unknown’;

SQL Overview

Data Query Language (DQL)

Important Data Modeling Language (DML) Statements

- **SELECT** - extracts data from a database

SQL Overview

Data Query Language (DQL)

SQL uses the following set of commands for all forms of queries in the database:

- SELECT/FROM/WHERE
  - **SELECT** : A fundamental query command of SQL; similar to the projection operation of relational algebra. Selects attributes based on condition described by WHERE clause.
  - **FROM** : Takes relation name as argument from which attributes are selected/projected. If more than one relation names are given, corresponds to Cartesian product.
  - **WHERE** : Defines predicate or conditions, which must match to qualify the attributes to be projected.

SQL Overview

Data Query Language (DQL)

SQL uses the following set of commands for all forms of data modification in the database:

- SELECT/FROM/WHERE
  - Used for selecting, projecting, and joining tuples
    - SELECT * | column1 [, column2, column3 ...]
      FROM table1 [, table2, table3 …]
      WHERE cond1 [op1 cond2 [op2 cond3 …]]
  - **Example**:
    - SELECT author_name
      FROM book_author
      WHERE age > 50;

    Yields names of authors from relation book_author whose age is greater than 50.

SQL Overview

Data Query Language (DQL)

SQL uses the following set of commands for all forms of data modification in the database:

- SELECT/FROM/JOIN/ON
  - **SELECT** : A fundamental query command of SQL; similar to the projection operation of relational algebra. Selects attributes based on condition described by WHERE clause.
  - **FROM** : Takes relation name as argument from which attributes are selected/projected. If more than one relation names are given, corresponds to Cartesian product.
  - **JOIN** : Takes relation name as argument to which the selected relation is joined.
  - **ON** : Defines predicate or conditions, which must match to qualify the attributes to be projected.

SQL Overview

Data Query Language (DQL)

SQL uses the following set of commands for all forms of data modification in the database:

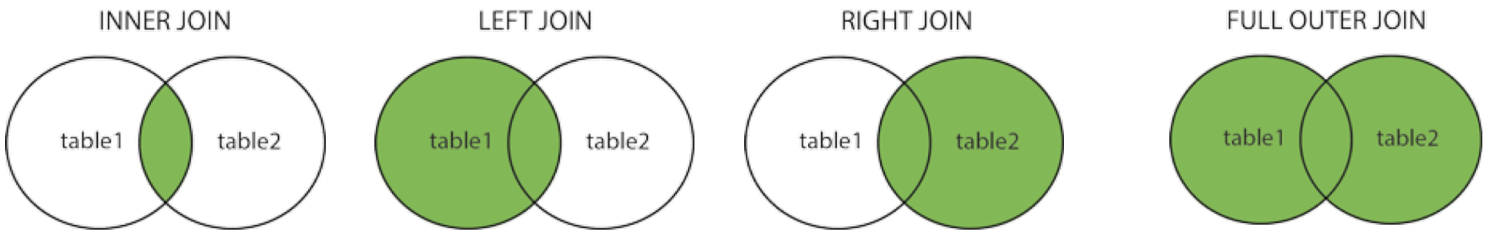
- SELECT/FROM/JOIN/ON
  - Used for selecting, projecting, and joining tuples
    - SELECT * | column1 [, column2, column3 ...]
      FROM table1 [, table2, table3 …] JOIN table
      ON cond1 [op1 cond2 [op2 cond3 …]]
  - **Example**:
    - SELECT book_author.author_name
      FROM book_author JOIN journal_author
      ON book_author.author_name = journal_author.author_name;

    Yields names of authors from the natural join of the relations book_author and journal_author on their author_name fields

SQL Overview

Data Query Language (DQL)

- **Join Types:**
  - **(INNER) JOIN**: Returns records that have matching values in both tables
  - **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
  - **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
  - **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table



SQL Overview

Data Query Language (DQL)

SQL uses the following set of commands for all forms of data modification in the database:

- UNION / INTERSECT / EXCEPT
  - Directly map to the relational algebra operators union (∪), intersection (∩) and subtraction (–).
  - Parentheses can be used to create more complex statements
  - **Examples**:
    - (SELECT * FROM S) UNION (SELECT * FROM T)

    Union of relations S and T
    - (SELECT * FROM S) INTERSECT(SELECT * FROM T)

    Intersection of relations S and T
    - (SELECT * FROM S) EXCEPT (SELECT * FROM T)

    Subtraction of relations S and T

SQL Overview

Data Query Language (DQL)

SQL uses the following set of commands for all forms of data modification in the database:

- Sort By
  - Directly map to the relational algebra operator tau (τ)
  - Can sort by ascending (ASC) or descending (DESC) -- default: ASC
  - **Example**:
    - SELECT * FROM Author SORT BY Name

    Sorts result by ascending Name attribute
    - SELECT * FROM Author SORT BY Name desc

    Sorts result by descending Name attribute
    - SELECT * FROM Author SORT BY Subject ASC

    Sorts result by ascending Subject attribute

SQL Overview

Data Query Language (DQL)

SQL uses the following set of commands for all forms of data modification in the database:

- Group By
  - Directly map to the relational algebra operator gamma (γ)
  - Used to group values of an attribute to aggregate them
  - **Example**:
    - SELECT count(*) AS Count FROM Author

    Counts the number of tuples in Author table
    - SELECT Name, count(*) AS Count FROM Author GROUP BY Name

    Counts the number of books published by each author
    - SELECT avg(Count) AS Avg FROM (SELECT count(*) AS Count FROM Name GROUP BY Name) AS CountPerAuthor

    Average number of books published by authors

SQL Overview

Data Control Language (DCL)

Important Data Control Language (DCL) Statements

- **GRANT** - grants permission to access database
- **REVOKE** - revokes permission to access database
- **DENY** - explicitly denies permission to access database

SQL Overview

Data Control Language (DCL)

SQL uses the following set of commands for all forms of control authorization in the database:

- GRANT/TO/ON
  - Used for adding new permissions to a database user.
    - GRANT privileges
      ON object
      TO user
      [WITH GRANT OPTION]
  - **Example**:
    - GRANT SELECT ON Author TO Joe;

      Joe can retrieve from Author table, but cannot grant other users permission because WITH GRANT OPTION clause not included.

SQL Overview

Data Control Language (DCL)

SQL uses the following set of commands for all forms of control authorization in the database:

- REVOKE/ON/FROM
  - Used for remove permissions from a database user.
    - REVOKE [GRANT OPTION FOR] permission
      ON object
      FROM user
      [CASCADE]
  - **Example**:
    - REVOKE SELECT ON Author FROM Joe;

      Joe can no longer retrieve from Author table.

SQL Overview

Data Control Language (DCL)

SQL uses the following set of commands for all forms of control authorization in the database:

- DENY/ON/TO
  - Used for explicitly prevent a user from receiving a permission.
    - DENY permission
      ON object
      TO user
  - **Example**:
    - DENY DELETE on Author to Matthew

      Matthew can never receive permission to delete information from the Author table.

SQL Overview

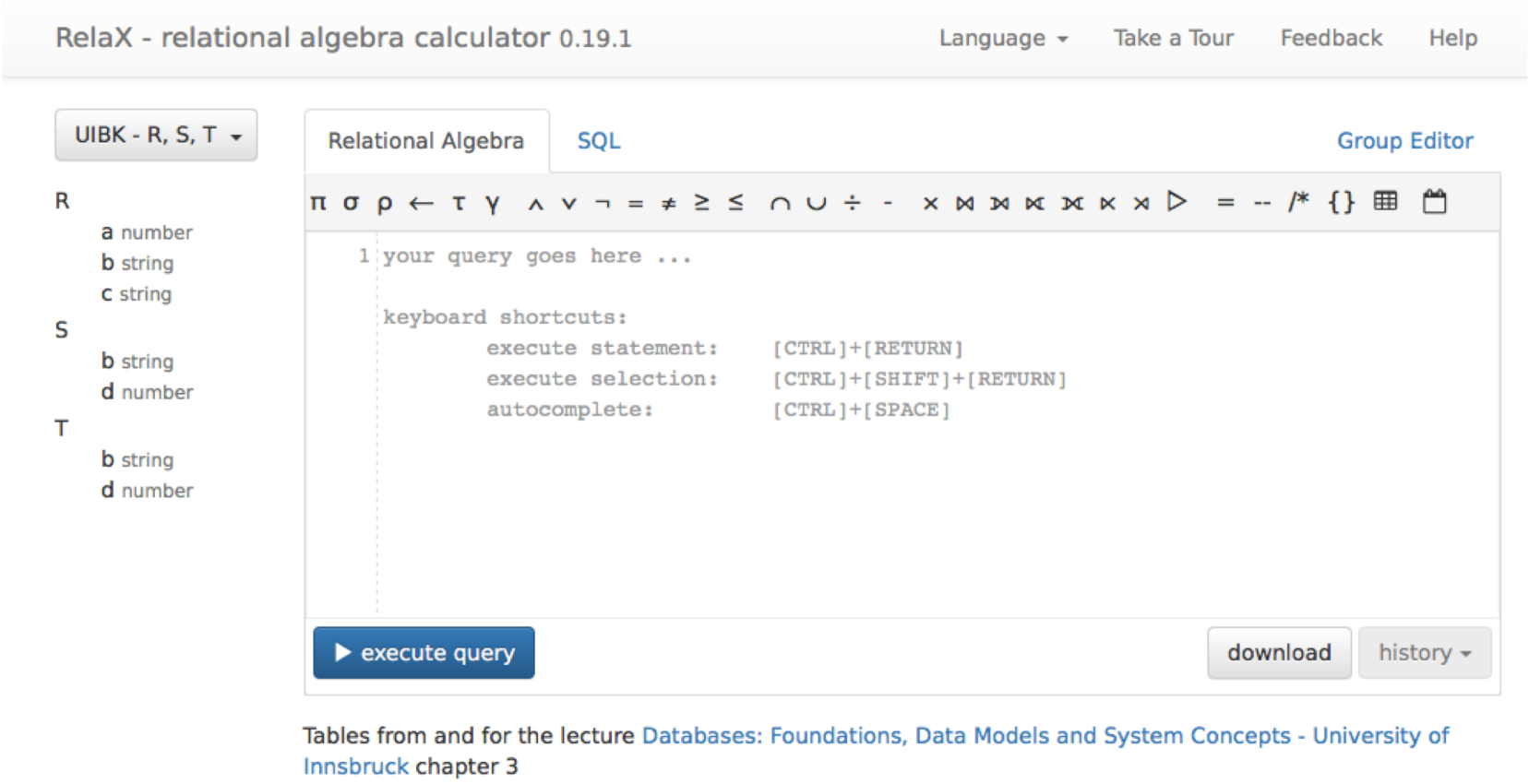
Demonstration of Relational Operators Using the *RelaX* Relational Algebra Calculator:

- http://dbis-uibk.github.io/relax/calc.htm

- Online tool for evaluating relational algebra expressions.
- Also evaluates SQL expressions and shows equivalent relational algebra expressions
- Includes a number of datasets with the ability to create and load new ones

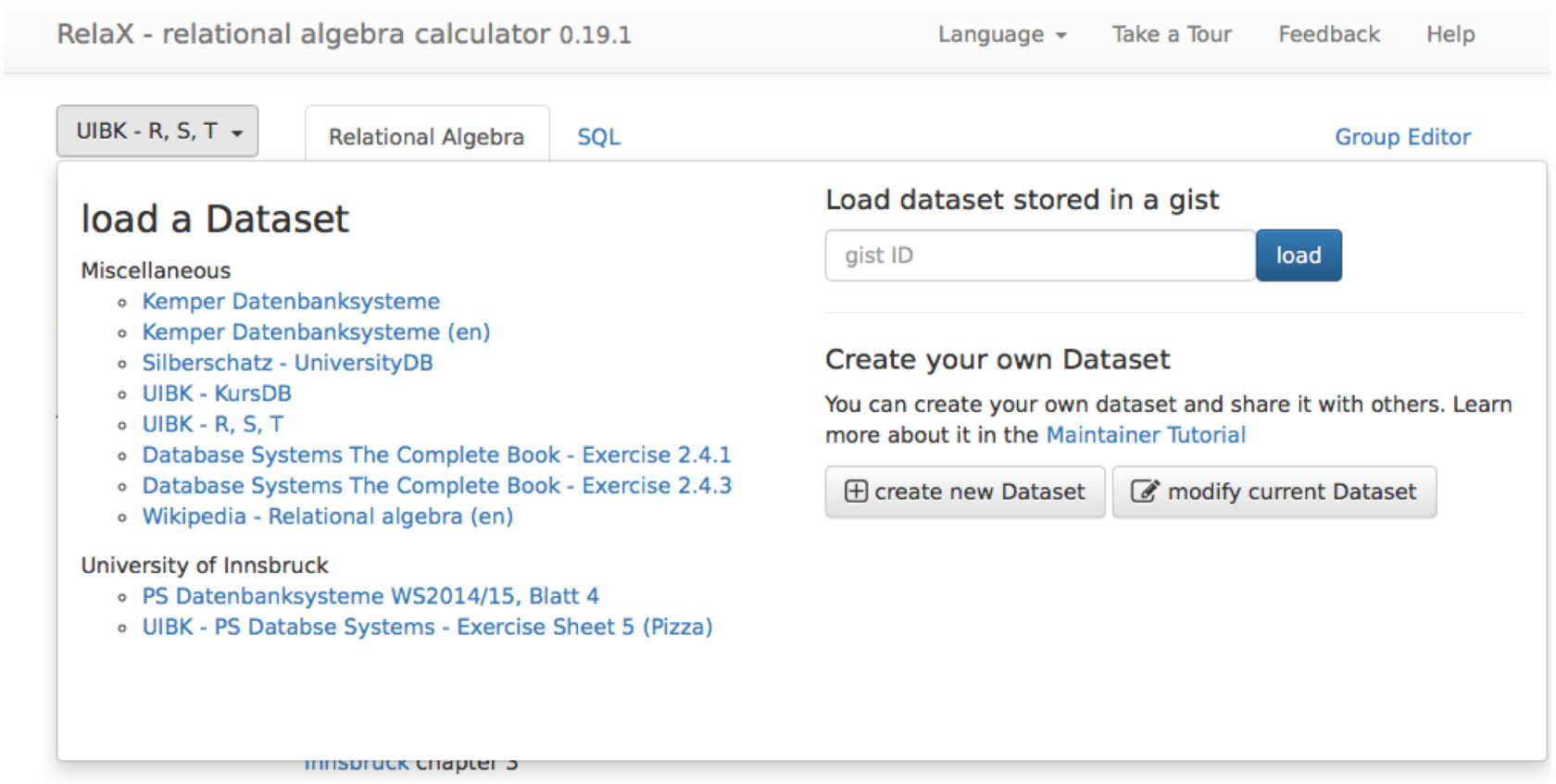
SQL Overview

RelaX Relational Algebra Calculator



SQL Overview

RelaX Relational Algebra Calculator



SQL Overview

Relation

- The Wikipedia relational algebra database provides the following relations:

Employee
- Name (string)
- Empid (number)
- DeptName ()

Dept
- DeptName (string)
- Manager (string)

Completed
- Student (string)
- Task (string)

DBProject
- Task (string)

Car
- CarModel (string)
- CarPrice (number)

Boat
- BoatModel (string)
- BoatPrice (number)

SQL Overview

RelaX Relational Algebra Calculator

Relational Algebra

Select (σ)

- The equivalent of specifying a relation in relational algebra is to select all rows in SQL:

  select * from Employee

Employee

Employee

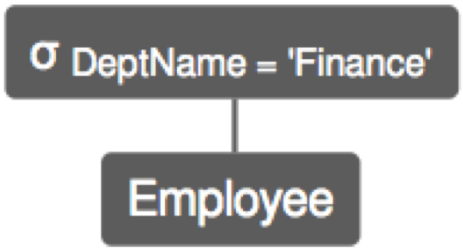
Employee.Name	Employee.EmpId	Employee.DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive

Relational Algebra

Select (σ)

- Select tuples from Employee with DeptName = Finance:

  select * from Employee where DeptName = 'Finance'



σ DeptName = 'Finance' Employee

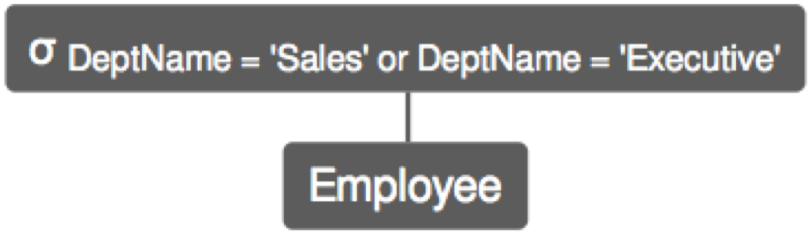
Employee.Name	Employee.EmpId	Employee.DeptName
Harry	3415	Finance
George	3401	Finance

Relational Algebra

Select (σ)

- Select tuples from Employee with DeptName = 'Sales' or DeptName ='Executive' :

  select * from Employee where DeptName = 'Sales' or DeptName = 'Executive'



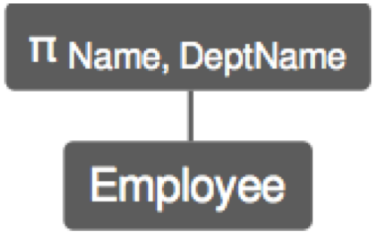
σ DeptName = 'Sales' or DeptName = 'Executive' Employee

Employee.Name	Employee.EmpId	Employee.DeptName
Sally	2241	Sales
Harriet	2202	Sales
Tim	1123	Executive

Relational Algebra

Project (π)

- Project the Name and DeptName attributes of Employee:
  select Name, DeptName from Employee



π Name, DeptName Employee

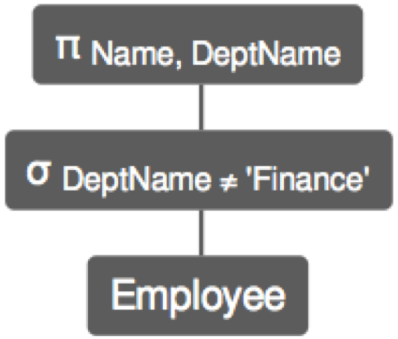
Employee.Name	Employee.DeptName
Harry	Finance
Sally	Sales
George	Finance
Harriet	Sales
Tim	Executive

Relational Algebra

Select (σ) and Project (π)

- Project the Name and DeptName attributes of the selection of Employee tuples with DeptName ≠ 'Finance'

  select Name, DeptName from Employee where DeptName != 'Finance'



π Name, DeptName σ DeptName ≠ 'Finance' Employee

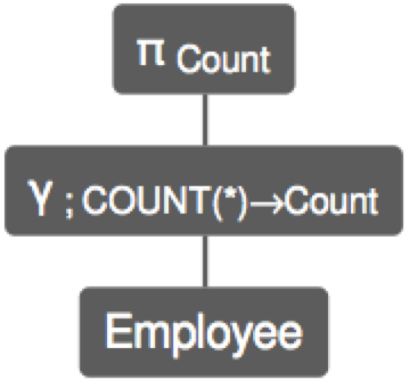
Employee.Name	Employee.DeptName
Sally	Sales
Harriet	Sales
Tim	Executive

Relational Algebra

Group (γ)

- Group to count total number of employee tuples.

  select count(*) as Count from Employee



π Count γ ; COUNT(*)→Count Employee

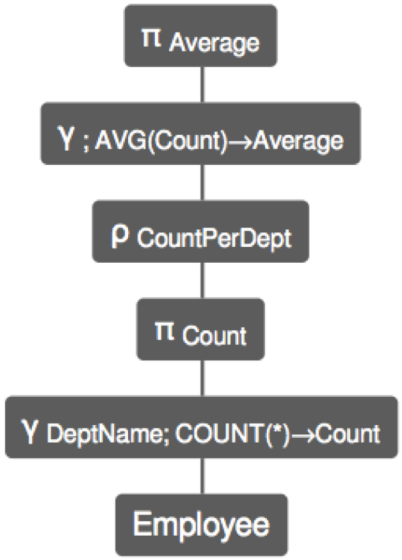
Count
5

Relational Algebra

Group (γ)

- Group to count average number of employees per department.

  select avg(Count) as Average from (select count(*) as Count from Employee group by DeptName) as CountPerDept



π Average γ ; AVG(Count)→Average ρ CountPerDept (π Count γ DeptName; COUNT(*)→Count Employee)

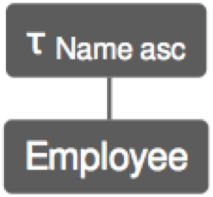
Average
1.5

Relational Algebra

Sort (τ)

- Sort Employee relation ascending (default) by Name.

  select * from Employee order by Name



τ Name asc Employee

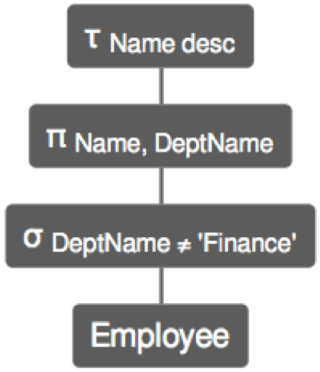
Employee.Name	Employee.EmpId	Employee.DeptName
George	3401	Finance
Harriet	2202	Sales
Harry	3415	Finance
Sally	2241	Sales
Tim	1123	Executive

Relational Algebra

Sort (τ)

- Sort the selection of Employee tuples with DeptName ≠ 'Finance' and projected by Name and DeptName descending by Name

  select Name, DeptName from Employee where DeptName != 'Finance' order by Name desc



τ Name desc π Name, DeptName σ DeptName ≠ 'Finance' Employee

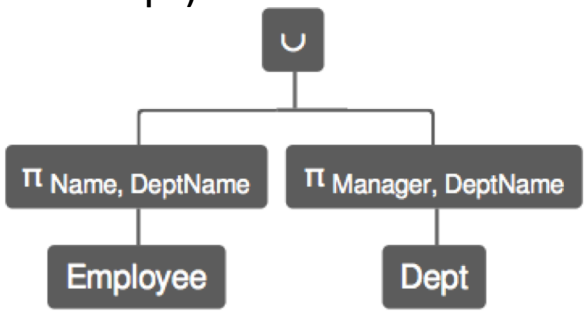
Employee.Name	Employee.DeptName
Tim	Executive
Sally	Sales
Harriet	Sales

Relational Algebra

Union (∪)

- Union of Name and Dept name fields from Employee and the Manager and DeptName fields from Dept

  (select Name, DeptName from Employee) union (select Manager, DeptName from Dept)



$(\pi_{\text{Name, DeptName}}\ \text{Employee}) \cup (\pi_{\text{Manager, DeptName}}\ \text{Dept})$

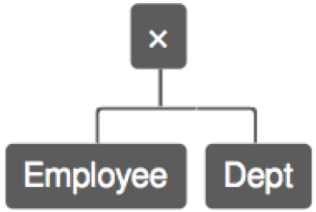
Employee.Name	Employee.DeptName
Harry	Finance
Sally	Sales
George	Finance
Harriet	Sales
Tim	Executive
Charles	Production

Relational Algebra

Cross Join (×)

- Cross join the Employee and Dept relations.

  select * from Employee, Dept



Employee × Dept

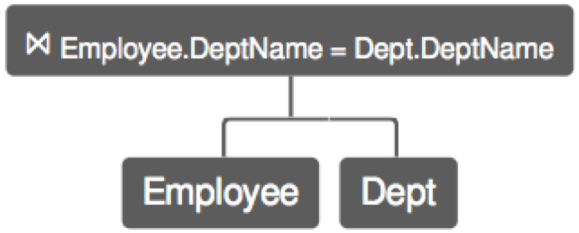
Employee.Name	Employee.EmpId	Employee.DeptName	Dept.DeptName	Dept.Manager
Harry	3415	Finance	Sales	Harriet
Harry	3415	Finance	Production	Charles
Sally	2241	Sales	Sales	Harriet
Sally	2241	Sales	Production	Charles
George	3401	Finance	Sales	Harriet
George	3401	Finance	Production	Charles
Harriet	2202	Sales	Sales	Harriet
Harriet	2202	Sales	Production	Charles
Tim	1123	Executive	Sales	Harriet
Tim	1123	Executive	Production	Charles

Relational Algebra

Natural Join (⋈)

- Natural join of the Employee and Dept relations with matching DeptName attributes.

  select * from Employee join Dept on Employee.DeptName = Dept.DeptName



Employee ⋈ Employee.DeptName = Dept.DeptName Dept

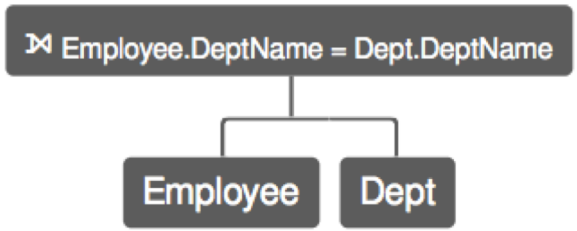
Employee.Name	Employee.EmpId	Employee.DeptName	Dept.DeptName	Dept.Manager
Sally	2241	Sales	Sales	Harriet
Harriet	2202	Sales	Sales	Harriet

Relational Algebra

Left Outer Join (⟕)

- Left Outer Join of the Employee and Dept relations on matching DeptName attributes. (similar for right and full outer join)

  select * from Employee left join Dept on Employee.DeptName = Dept.DeptName



Employee ⟕ Employee.DeptName = Dept.DeptName Dept

Employee.Name	Employee.EmpId	Employee.DeptName	Dept.DeptName	Dept.Manager
Harry	3415	Finance	*null*	*null*
Sally	2241	Sales	Sales	Harriet
George	3401	Finance	*null*	*null*
Harriet	2202	Sales	Sales	Harriet
Tim	1123	Executive	*null*	*null*

Relational Algebra

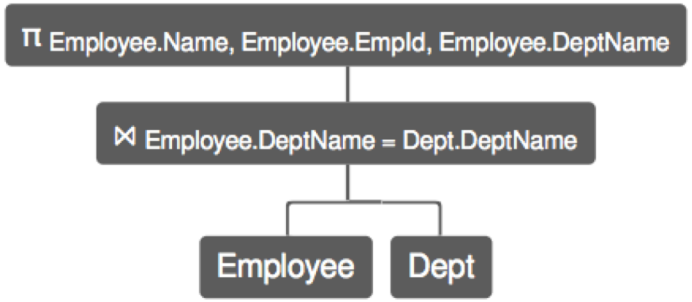
Left Semijoin (⋉)

- SQL has no Left Semijoin It is often implemented using EXISTS as

  select * from Employee where exists (select * from Dept where Employee.DeptName = Dept.DeptName)

- If EXISTS is not available, can also be implemented using JOIN as

  select * from Employee left join Dept on Employee.DeptName = Dept.DeptName



π Employee.Name, Employee.EmpId, Employee.DeptName Employee ⋉ Employee.DeptName = Dept.DeptName Dept

Employee.Name	Employee.EmpId	Employee.DeptName
Sally	2241	Sales
Harriet	2202	Sales

Relational Algebra

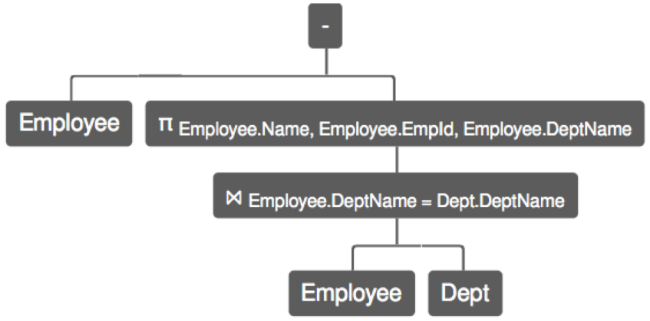
Antijoin (▷)

- SQL has no Antijoin. It is often implemented using NOT EXISTS as

  select * from Employee where not exists (select * from Dept where Employee.DeptName = Dept.DeptName)

- If EXISTS not available, can also be implemented using EXCEPT as

  (select * from Employee) except (select Employee.Name, Employee.EmpId, Employee.DeptName from Employee join Dept on Employee.DeptName = Dept.DeptName)



(Employee) - (π Employee.Name, Employee.EmpId, Employee.DeptName Employee ⋈ Employee.DeptName = Dept.DeptName Dept)

Employee.Name	Employee.EmpId	Employee.DeptName
Harry	3415	Finance
George	3401	Finance
Tim	1123	Executive