LAB 4 Notes

The Relational Algebra

- Any questions on the project (Discuss)
- In the previous lab we discussed how to convert an ER model into the Relational model of a specific database.
- Today we will discuss how to store and retrieve information in our database

Outline

1) Glace at Relational Algebra Operators. Emphasize on the most important aspects and then jump into examples
2) Examples on Relational Algebra.
3) A Glace at SQL Operators. Emphasize on the most important aspects and then jump into examples
5) Examples on SQL.
6) Putting all together: Relational Algebra & SQL

Relational Algebra (is the foundation for Structured Query Language SQL)

- One of 2 Formal Query Languages associated with the relational model
- An Algebra, is a formal structure consisting of *sets* and *operations* on those sets.
- Relational Algebra is based on set theory
- The Inputs/Outputs are relations (set of records)
- R.A is a closed language. The output of one operator is a RELATION (nested)

(Working Example in case they want one.)

Employee

ssn,	**name,**	**surname,**	**age**
12,	Jacob,	Jacobson,	30
13,	Mike,	Allison,	50
14,	Chris,	Mathews,	80

Department

did,	**dname,**	**mngrssn**
1,	A,	12
2,	B,	13
3,	C,	12

- **Unary Operators:** Operation done on just one relation

1. **Projection Π** (on columns), πdname(DEPARTMENT) = A, B, C
2. **Selection σ** (on rows) σage>50(EMPLOYEE) =
   13, Mike, Allison, 50
   14, Chris Mathews, 80

- **Binary Operators (Set Operators):**

3) **Union U** . RUS. R and S must be **union compatible (same # fields, domains)**
4) **Intersection** ^ . R^S. R and S must be **union compatible**
5) **Set Difference, R-S** all tuples in R but not in S
6) **Cross-Product (Cartesian Product) R x S.** All fields of R followed by all fields of S

12, Jacob, Jacobson,30, 1, A, 12

12, Jacob, Jacobson,30, 2, B, 13
12, Jacob, Jacobson,30, 3, C, 12

- AxB = BxA
- Number of tuples : MxN
- R $|><|_c$ S = σ_c(RxS) (*General*)
- Renaming operator: p(C(1→id1, 5→id2), Employee x Employee)

7) JOIN (just naming difference
Condition Join : If condition is anything (A$|><|_{a>b}$ B)
Equi-join : If condition is equality (A$|><|_{A.a=B.b}$ B)
Natural-join : An equijoin condition done on the attribute that has the same name
e.g A|><| B => No duplicate column

8) DIVISION
"Useful for queries like: Find name of sailors who reserved ALL boats"
In SQL this is represented by nested queries.
e.g.

Reservation	**Boat**
ssn, ~~name~~, boatID	boatID

Reservation/Boat => ssn of sailors who reserved ALL boats
Otherwise if name is included then different result

Aggregate operations e.g.

- "Find the minimum salary of all employees are not supported by standard Relational Algebra".
- Some extensions allow you to deal with them but in this course you don't consider them.
- We will see Aggregates in further extend in Chapter 5.

EXAMPLE

EMPLOYEE

FNAME	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNUM

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

WORKS_ON

ESSN	PNUM	HOURS

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

- Retrieve the names of all employees in department 5 who work more than 10 hours per week on the 'ProjectX' project.

ρ(EMP_W_X, $\sigma_{PNAME='ProjectX'}$(PROJECT)) |><|$_{PNUMBER=PNUM}$ (WORKS_ON))
ρ(EMP_WORK_10, (EMPLOYEE) |><|$_{SSN=ESSN}$ ($\sigma_{HOURS>10}$ (EMP_W_X))
$\pi_{LNAME, FNAME}$ ($\sigma_{DNUM=5}$(EMP_WORK_10))

- Retrieve the names of all employees who work on every project.
  ρ(PROJ_EMPS, $\pi_{PNUM, ESSN}$ (WORKS_ON))
  ρ(ALL_PROJS, $\pi_{PNUMBER}$ (PROJECT))
  ρ(EMPS_ALL_PROJS, PROJ_EMPS / ALLPROJS)
  $\pi_{LNAME, FNAME}$ (EMPLOYEE |><| EMP_ALL_PROJS)

- Find the names and addresses of all employees who work on at least one project located in Houston but whose department has no location in Houston.
  ρ (E_P_HOU, π_{ESSN}(WORKS_ON |><|$_{PNUM=PNUMBER}$ ($\sigma_{PLOCATION='Houston'}$(PROJECT))))
  ρ (D_NO_HOU, $\pi_{DNUMBER}$(DEPARTMENT) – $_{DNUMBER}$($\sigma_{DLOCATION='Houston'}$(DEPARTMENT))
  ρ (RESULT_EMPS, π_{SSN}(EMPLOYEE |><|$_{DNUM=DNUMBER}$ (D_NO_HOU))
  $\pi_{LNAME, FNAME, ADDRESS}$(EMPLOYEE |><| RESULT_EMPS)

- **SQL (Structured Query Language)**
- Widely used relational database language
- SQL – Query Language **but has several other aspects**
  1) **DDL** (Definition Language) Create/delete/Alter tables & Views. Creating indexes/ deleting indexes
  2) **DML** (Manipulation Language) Insert/Delete/ Update Rows
  3) **Embedded and Dynamic SQL (will be covered as part of the project)**
     Allows SQL code to be executed from a host language such as C or Java.

Paradox is that in SQL we call:

- SELECT -> Projection π and
- WHERE -> Selection σ

A) SQL BASIC QUERY BLOCK

```
SELECT [DISTINCT] select-list
FROM from-list
WHERE qualification;
```

Sailors(<u>sid</u>, name, rating, age)

1	Chris	20
2	Chris	35
3	Chris	20
4	John	15

SELECT DISTINCT name, age
FROM Sailors;
➔ selects all the distinct pairs
i.e. chris, 20
chris, 35

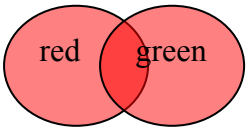
Relational Algebra => $\pi_{name,age}$ (Sailors)

B) #1 Set Manipulation constructs: **SQL UNION, INTERSECT AND EXCEPT**

+ Set Manipulation constructs extend the basic query form
+ Union compatible

```
(SELECT [DISTINCT] select-list-X
FROM from-list
WHERE qualification)
UNION/INTERSECT/EXCEPT (MINUS)
(SELECT [DISTINCT] select-list-X
FROM from-list
WHERE qualification)
```

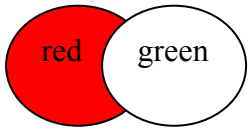
Sailors who reserved Red or green boat




```
SELECT *
FROM SailorsReserveBoats
WHERE color=red OR color=green;
```

```
SELECT *
FROM SailorsReserveBoats
WHERE color=red
UNION
SELECT *
FROM SailorsReserveBoats
WHERE color=green
```

Sailors who reserved Red but not green boat




```
SELECT *
FROM SailorsReserveBoats
WHERE color=red
EXCEPT
SELECT *
FROM SailorsReserveBoats
WHERE color=green
```

C) #2 Set Manipulation constructs: **Correlated Nested and nested IN, EXIST**

```
(SELECT [DISTINCT] select-list
FROM from-list
WHERE attribute [NOT] IN
                (SELECT attribute
                 FROM from-list
                 WHERE condition)
```

Union compatible

Example:
NOT CORELLATED IN (work well by optimizer)

: Select sailors who reserved boat 103

```
SELECT *
FROM EMPLOYEE
WHERE sid IN
        (SELECT R.sid
         FROM RESERVES R)
```

```
SELECT *
FROM EMPLOYEE E, RESERVES R
WHERE E.sid = R.sid;
```

CORELLATED EXISTS (ARE NOT optimized adequately)

Allows us to check whether a set is empty or not.
e.g. usually helpful in correlated queries.

```
(SELECT [DISTINCT] select-list
FROM from-list
WHERE EXISTS
```

```
(SELECT attribute
 FROM from-list
 WHERE condition)
```

e.g. select the employees with the highest salary

```
SELECT *
FROM EMPLOYEE E1
WHERE EXISTS (SELECT MAX(E2.salary)
                    FROM EMPLOYEE E2
                    WHERE E2.id = E.id)
```

D) AGGREGATE OPERATORS

```
SELECT [COUNT, SUM, AVG, MAX, MIN(attribute)]
FROM from-list
WHERE qualification
```

Putting it all together: Simple Example

Consider the following schema:

Suppliers(sid: integer, sname: string, address: string);
Parts(pid: integer, pname: string. color: string);
Catalog(sid: integer, pid: integer, cost: real);

Execute the script called lab4.sql on my web site to create the tables. Load data into the tables found in catalog.txt, parts.txt, suppliers.txt. Refer to the postgres manual if you have any question regarding this process.

Write the Relational Algebra statements and execute the corresponding SQL statements for the following queries:

- Find the pid of parts with cost lower than 10$
- Find the name of parts with cost lower than 10$
- Find the address of the suppliers who supply "Fire Hydrant Cap"
- Find the name of the suppliers who supply green parts