CMPS 277 – Principles of Database Systems

https://courses.soe.ucsc.edu/courses/cmps277/Fall11/01

The Influence of Relational Algebra and Relational Calculus on SQL
The basic SQL construct is:

```
SELECT <attribute list>
FROM <relation list>
WHERE <condition>
```

More formally,

```
SELECT R_{i1}.A1, ..., R_{im}.Am
FROM   R_1, ...,R_K
WHERE   \Psi
```

Restrictions:
- $R_{1}, ...,R_{K}$ are distinct relation names (no repetitions)
- Each $R_{ij}.A_j$ is an attribute of $R_{ij}$
- $\Psi$ is a condition with a precise (and rather complex) syntax.
### SQL vs. Relational Algebra

<table>
<thead>
<tr>
<th>SQL</th>
<th>Relational Algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>Projection $\pi$</td>
</tr>
<tr>
<td>FROM</td>
<td>Cartesian Product</td>
</tr>
<tr>
<td>WHERE</td>
<td>Selection $\sigma$</td>
</tr>
</tbody>
</table>

**Semantics of SQL via interpretation to Relational Algebra**

SELECT $R_{i1}.A1, ..., R_{im}.Am$
FROM $R_1, ..., R_K$
WHERE $\Psi$

$= \pi_{R_{i1}.A1, ..., R_{im}.Am}(\sigma_{\Psi}(R_1 \times ... \times R_K))$
SQL vs. Relational Algebra

Example: FACULTY(name, dpt, salary), CHAIR(dpt, name)
Find the salaries of department chairs
C-SALARY(dpt,salary) =

- Relational Algebra
  \[ \pi_{\text{dpt}, \text{Salary}}(\sigma_{\text{name} = \text{C.name} \land \text{dpt} = \text{C.dpt}}(\text{FACULTY} \times \text{CHAIR})) \]

- SQL
  
  ```sql
  SELECT FACULTY.dpt, FACULTY.salary
  FROM FACULTY, CHAIR
  WHERE FACULTY.name = CHAIR.name AND
    FACULTY.dpt = CHAIR.dpt
  ```
Self-Joins

Recall that the relation names in the FROM list must be distinct.

- **Question**: How do we then compute $R \times R$ in SQL?

- **Question**: Why do we care to compute $R \times R$?

- **Answer**: Many interesting queries involve self-joins, which, in turn, require computing $R \times R$. 
Self-Joins

Example: Given MANAGES(manager,employee), we want to compute 2-MANAGES(2manager,employee).

In relational algebra, we can reference columns by position number. So, 2-MANAGES is expressed by the relational algebra expression

\[ \pi_{1,4} \left( \sigma_{2 = 3} \left( \text{MANAGES} \times \text{MANAGES} \right) \right) \].

However, SQL does not support referencing columns by position number. Instead, SQL supports an aliasing mechanism.
Aliases in SQL

- SQL allows us to give one or more new names to a given relation; these are aliases of the given relation.

- **Rules for Aliases Creation**
  - Aliases are created in the FROM list
    - FROM <rel. name1> AS <rel. name2>, ...
    - The new names can be referenced in the SELECT list and in the WHERE clause.

- **Example:** Expressing $R \times R$ in SQL:
  ```sql
  SELECT * 
  FROM R as S, R as T
  ```
Aliases in SQL

**Example:** MANAGES(manager,employee)  
2-MANAGES(2manager,employee) in SQL:

```
SELECT R.manager as 2manager, T.employee as employee
FROM MANAGES AS R, MANAGES AS T
WHERE R.employee = T.manager
```

**Note:** This example also illustrates how SQL allows for the renaming of attribute names in the SELECT list.

**Note:** Aliases in SQL are used not only out of necessity (self-joins), but also for convenience in order to create short nicknames for relations.
Sets vs. Multisets

Informal Definition:

- A **set** is a collection of distinct objects viewed as a new object.
- A **multiset (bag)** is a collection of (not necessarily distinct) object viewed as a new object.
- The **multiplicity** \( m(a,B) \) of an element \( a \) of a multiset \( B \) is the number of occurrences of that element in the multiset.

Examples:

- \( B = \{1,3,3,6,6,6,7\} \) is a multiset, but not a set (here, \( m(3,B)=2 \))
- \( PF(n) \) is the multiset representing the prime factorization of \( n \)
  - \( PF(90) = \{2,3,3,5\} \) (so, \( m(2,PF(90)) = 1 \), \( m(3,PF(90)) = 2 \))
  - \( PF(64) = \{2,2,2,2,2\} \) (so, \( m(2,PF(64)) = 6 \)).
Operations on Multisets

- **Union** $R \cup S$: \[ m(t, R \cup S) = m(t, R) + m(t, S) \]

- **Difference** $R - S$: \[ m(t, R - S) = \max\{m(t, R) - m(t, S), 0\} \]

- **Intersection** $R \cap S$: \[ m(t, R \cap S) = \min\{m(t, R), m(t, S)\} \]

- **Cartesian Product**: 
  If $t \in R$ and $t' \in S$, then \[ m(tt', R \times S) = m(t, R)m(t',S). \]

**Note:**
- If $R$ is a multiset, then $\pi_X(R)$ is also a multiset.
- If $R$ is a set, then $\pi_X(R)$ may be a multiset.
- If $R$ is a set, then $\sigma_\Theta (R)$ is a set.
- If $R$ is a multiset, then $\sigma_\Theta (R)$ may be a set.
Relations are sets (i.e., they consist of distinct tuples)

Relational algebra expressions take relations as arguments and return relations as values.
- In particular, all duplicates are eliminated in relational algebra.

The SELECT ... FROM ... WHERE construct of SQL does not eliminate duplicates, unless explicitly requested.
- In general, the SELECT ... FROM ... WHERE construct takes multisets as arguments and returns multisets as values.
- In particular, the SELECT ... FROM ... WHERE construct may return a multiset, even if the arguments are sets.
## Set Semantics vs. Multiset Semantics

The table below compares the set semantics and multiset semantics for the `SAVINGS` table.

<table>
<thead>
<tr>
<th>branch-name</th>
<th>acc-no</th>
<th>cust-name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aptos</td>
<td>153125</td>
<td>Vianu</td>
<td>3,450</td>
</tr>
<tr>
<td>Santa Cruz</td>
<td>123658</td>
<td>Hull</td>
<td>2,817</td>
</tr>
<tr>
<td>San Jose</td>
<td>321456</td>
<td>Codd</td>
<td>9,234</td>
</tr>
<tr>
<td>San Jose</td>
<td>334789</td>
<td>Codd</td>
<td>875</td>
</tr>
</tbody>
</table>

### Set Semantics

\[ \pi_{\text{cust-name}, \text{branch-name}}(\text{SAVINGS}) \]

<table>
<thead>
<tr>
<th>cust-name</th>
<th>branch-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vianu</td>
<td>Aptos</td>
</tr>
<tr>
<td>Hull</td>
<td>Santa Cruz</td>
</tr>
<tr>
<td>Codd</td>
<td>San Jose</td>
</tr>
</tbody>
</table>

### Multiset Semantics

\[ \text{SELECT cust-name,branch-name FROM } \text{SAVINGS} \]

<table>
<thead>
<tr>
<th>cust-name</th>
<th>Branch-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vianu</td>
<td>Aptos</td>
</tr>
<tr>
<td>Hull</td>
<td>Santa Cruz</td>
</tr>
<tr>
<td>Codd</td>
<td>San Jose</td>
</tr>
<tr>
<td>Codd</td>
<td>San Jose</td>
</tr>
</tbody>
</table>
Eliminating Duplicates in SQL

- **Question:** Why are not duplicates eliminated in SQL?

- **Answer:**
  - **Efficiency** (duplicate elimination may take quadratic time).
  - **Necessity:** In certain cases, eliminating duplicates results to information loss and errors in computing averages.

- **Question:** What if we want to eliminate duplicates in SQL?

- **Answer:** Use the constuct:
  
  ```sql
  SELECT DISTINCT <attribute list>
  FROM   <relation list>
  WHERE  <condition>
  ```
The Relational Completeness of SQL

- **Fact:** SQL is a relationally complete database query language
- **Reason:**
  - The SELECT DISTINCT ... FROM ... WHERE ... construct of SQL makes it possible to express cartesian product, projection, and selection.
  - In addition, SQL has explicit constructs for union and difference:
    - The union \( R \cup S \) of two relations \( R \) and \( S \) is expressed by:
      ```
      (SELECT * FROM R) UNION (SELECT * FROM S)
      ```
    - The difference \( R - S \) of two relations \( R \) and \( S \) is expressed by:
      ```
      (SELECT * FROM R) EXCEPT (SELECT * FROM S)
      ```
  - UNION and EXCEPT eliminates duplicates! (Set semantics)
  - UNION ALL and EXCEPT ALL does not. (Multiset semantics)
Relational Calculus and SQL

- Relational calculus has influenced the design of SQL.
- In particular, existential and universal quantification are used in two different forms in SQL.
- Both these forms occur in the allowable conditions in the WHERE clause of the SELECT ... FROM ... WHERE construct.
- In addition, sets/multisets are allowed as operands in the WHERE clause (this is what makes existential and universal quantification meaningful).
Sets as Operands in SQL

- Sets are allowed as operands in the WHERE clause. Sets are defined by listing their elements (boring feature), or as the result of a SELECT ... FROM ... WHERE construct nested inside the WHERE clause of an outer SELECT ... FROM .. WHERE (interesting feature)

- This is what makes SQL a “structured” language, i.e., we have queries inside queries up to any finite depth of nesting.

- When sets are used as operands in a comparison clause:
  - We must use one of the keywords IN, NOT IN, SOME, ALL.
  - SOME and ALL must be preceded by one of the of comparison operators =, ≠, ≥, ≤, >, <.
  - The use of SOME and ALL is the first form of existential and universal quantification in SQL.
Sets as Operands in SQL

Example:  FACULTY(name,dpt,salary)

- Find the names of faculty who are in a department in which no member earns more than $175,000.

- SELECT name
  FROM FACULTY
  WHERE dpt NOT IN (SELECT dpt
      FROM FACULTY
      WHERE salary > 175,000)

Exercise: Express this query without using an SQL subquery.
SOME and ALL in SQL

- **Syntax:** In the WHERE clause, we can have subclauses of the form
  - `<attribute name>`  op  SOME T
  - `<attribute name>`  op  ALL T, where
    - op is one of the comparison operators =, ≠, ≥, ≤, >, <
    - T is the result of a nested SELECT ... FROM ... WHERE clause.

- **Semantics:**
  - `<attribute name>`  op  SOME T means:
    \[(\exists x)(x \in T \land <\text{attribute name}> \text{ op } x)\]
  - `<attribute name>`  op  ALL S means:
    \[(\forall x)(x \in T \rightarrow <\text{attribute name}> \text{ op } x)\].
SOME and ALL in SQL

- **Note:**
  - `<attribute name> = SOME T` is the same as IN
  - `<attribute name> ≠ ALL S` is the same as NOT IN

- **Note:** Earlier versions of SQL used ANY in place of SOME.
  - The use of ANY can be quite confusing and can lead to errors.
  - Even if the system supports ANY, it is better to avoid using it.
SOME and ALL in SQL

Example: FACULTY(name,dpt,salary)
- Find the highest paid faculty in CS
- SELECT name
  FROM FACULTY
  WHERE dpt = "CS" AND salary ≥ ALL (SELECT salary
  FROM FACULTY
  WHERE dpt = "CS").

Question: What is the result of the following SQL query?
- SELECT name
  FROM FACULTY
  WHERE dpt = "CS" AND salary > ALL (SELECT salary
  FROM FACULTY
  WHERE dpt = "CS").
SOME and ALL in SQL

Question: What are the results of the following two SQL queries?

- SELECT name
  FROM FACULTY
  WHERE dpt = "CS" AND salary > SOME (SELECT salary
  FROM FACULTY
  WHERE dpt = "CS").

- SELECT name
  FROM FACULTY
  WHERE dpt = "CS" AND salary ≥ SOME (SELECT salary
  FROM FACULTY
  WHERE dpt = "CS").

Answer:
- The first returns all CS faculty who are not the lowest paid ones.
- The second returns all CS faculty.
EXISTS and NOT EXISTS in SQL

- **Syntax:**
  - SELECT ... FROM ... WHERE EXISTS (SELECT ... FROM ... WHERE)

- **Semantics:** The subquery (SELECT ... FROM ... WHERE) is evaluated and the resulting set is tested for emptiness:
  - If it is non-empty, then the condition in WHERE evaluates to “true”; otherwise, it evaluates to “false”.

- **Syntax:**
  - SELECT ... FROM ... WHERE NOT EXISTS (SELECT ... FROM ... WHERE)

- **Semantics:** The subquery (SELECT ... FROM ... WHERE) is evaluated and the resulting set is tested for emptiness:
  - If it is empty, then the condition in WHERE evaluates to “true”; otherwise, it evaluates to “false”.
**EXAMPLE:** FACULTY(name, dpt, salary)
Find the faculty in the CS dpt who are not the lowest paid ones.
SELECT R.name
FROM FACULTY as R
WHERE dpt = "CS" AND EXISTS (SELECT *
FROM FACULTY AS T
WHERE R.dpt = "CS" AND
R.salary > T.salary)

**Note:** This is an example of a correlated subquery:
- The subquery has to be evaluated separately for each tuple in the FROM list of the outer query.
- The tuple is kept or removed depending on the result of the EXISTS test.
EXISTS and NOT EXISTS in SQL

Example: FACULTY(name, dpt, salary)
Find the highest paid faculty in CS.
SELECT R.name
FROM FACULTY as R
WHERE dpt = "CS" AND NOT EXISTS (SELECT *
FROM FACULTY AS T
WHERE R.dpt = "CS" AND 
    R.salary < T.salary)

Note:
- SQL queries with SOME and ALL can be transformed to SQL queries with EXISTE and NOT EXISTS.
- SOME, ALL, EXISTE, NOT EXISTE are imported directly from relational calculus.
Beyond Relational Algebra: Aggregate Operators in SQL

SQL supports the aggregate operators avg, max, min, sum, count

- **Syntax:** In the SELECT line, we can use
  - `avg(<attribute name>)`
  - `avg(DISTINCT <attribute name>)`

- **Semantics:**
  - `avg(<attribute name>)` computes the average of the multiset of values occurring in the column of `<attribute name>`
  - `avg(DISTINCT <attribute name>)` computes the average of the set of distinct values in the column of `<attribute name>`

- Similar syntax and semantics for the other aggregate operators.
- `count(*)` computes the total number of rows in the table
- `count(DISTINCT *)` is not supported in SQL.
Aggregate Operators in SQL

- **Example:** FACULTY(name,dpt,salary)
  Find the max, min, and average salary of CS faculty
  - SELECT max(salary), min(salary), avg(salary)
    FROM FACULTY
    WHERE dpt = "CS"
  (the selection condition is applied before the aggregate operators)

- **Example:** Find the total number of faculty in CS and in CE
  - SELECT count(*)
    FROM FACULTY
    WHERE dpt = "CS" OR dpt = "CE"
  (in these examples, it is assumed that FACULTY is a set)
Aggregate Operators in SQL

- **Warning:** aggregate operators are not allowed as operands in the WHERE clause
- **Example:** Find all faculty in CS who are paid higher than average
  - The following is tempting, but **wrong**:
    ```sql
    SELECT name
    FROM FACULTY
    WHERE dpt = "CS" AND salary > avg(salary).
    ```
  - The **correct** way to express this query is as follows:
    ```sql
    SELECT name
    FROM FACULTY
    WHERE dpt = "CS" AND salary > (SELECT (avg(salary)
    FROM FACULTY
    WHERE dpt = "CS")
    ```
Fact:

- The syntax of relational algebra and relational algebra can be augmented with aggregate operators in a straightforward way.
- SQL, however, also supports the GROUP BY construct, which makes it possible to apply the aggregate operators to disjoint "clusters" of tuples in a relation.
- Furthermore, selection can be applied to these clusters using the HAVING construct.
- So, the full syntax of SQL is:
  ```sql
  SELECT <list of attributes>, <list of aggr. operator applications>
  FROM    <list of relation names>
  WHERE   <condition>
  GROUP BY <list of attributes> (must contain list in SELECT line)
  HAVING   <condition>
  ```
Aggregate Operators and the GROUP BY construct

Example: FACULTY(name,dpt,salary)

- Find the average and max salary paid in each department
  - SELECT dpt, avg(salary) as average-salary, max(salary) as max-salary
    FROM FACULTY
    GROUP BY dpt

- Find the average and max salary paid in each department in which the minimum salary is at least $90,000
  - SELECT dpt, avg(salary) as average-salary, max(salary) as max-salary
    FROM FACULTY
    GROUP BY dpt
    HAVING min(salary) > 90,000
Rigorous Semantics of GROUP BY

- **Question:** How can we give precise semantics to GROUP BY?

- **Answer:** The GROUP BY clause can be given rigorous semantics using the concepts of equivalence relation and equivalence classes in discrete mathematics.
Definition: A binary relation $R \subseteq A \times A$ is an equivalence relation on $A$ if $R$ is:
- Reflexive (for all $a$ in $A$, we have $(a,a) \in R$);
- Symmetric (if $(a,b) \in R$, then $(b,a) \in R$);
- Transitive (if $(a,b) \in R$ and $(b,c) \in R$, then $(a,c) \in R$).

Examples: “same birthday”, “same income”, $x \equiv y \mod(p)$, ...

Definition: If $R$ is an equivalence relation on $A$ and $a$ is an element of $A$, then the equivalence class of $a$ is the set $[a]_R = \{b \in A: R(a,b)\}$.

Example: The equivalence relation $x \equiv y \mod(p)$, with $p$ a prime number, has $p$ distinct equivalence classes.
Equivalence Relations and Partitions

**Definition:** A partition of a set $A$ is a collection of pairwise disjoint subsets of $A$ whose union is equal to $A$.

**Fact:**
- If $R$ is an equivalence relation on $A$, then the equivalence classes of $R$ form a partition of $A$.
- Conversely, every partition $P$ of $A$ gives rise to an equivalence relation $R$ on $A$, where
  - $R(x,y)$ if and only if $x$ and $y$ are in the same part of $P$. 
The sets $P_1, P_2, \ldots, P_k$ form a partition of the set $A$.

Equivalence Relation on $A$: “$a$ and $b$ are in the same part $P_i$”
The list of attributes in the GROUP BY clause induces an equivalence relation on the table (FROM ... WHERE ...):

- Two tuples of that table are equivalent if they agree on all values of attributes in the list of attributes in the GROUP BY clause.
- The equivalence classes of this equivalence relation partition the table (FROM ... WHERE ...) into disjoint sub-tables.

The aggregate operators in the SELECT line are applied to the attributes of each of the equivalence classes of this equivalence relation.

If the HAVING clause is present, then only the sub-tables that satisfy the condition in the HAVING clause are kept.