SQL Queries
Joins over Two or More Tables

SQL Queries

```
SELECT [DISTINCT] [SUM | COUNT | AVG] result_table
FROM table1, table2
[WHERE table_predicates AND join_conditions]
[GROUP BY grouping_attributes
    [HAVING agg_condition]]
[ORDER BY sorting_attributes]
[UNION [ALL]] [INTERSECT] [EXCEPT]
```

Cartesian Product

- R(A) = {1,1,2,3}
- S(B) = {1,3,4}
- R x S(A,B) = {

  (1,1),(1,3),(1,4),

  (1,1),(1,3),(1,4),

  (2,1),(2,3),(2,4),

  (3,1),(3,3),(3,4)}
- The result consists of pairs of one element from R and one from S
- Every element from R is paired with every element from S
- The number of elements in R x S is |R|*|S|, i.e., the size of R multiplied by the size of S

- select *

  from R, S
- The schema of the result is the **union** of the R schema and the S schema

Cartesian Product Generalization

- R(A) = {1,1,2,3}
- S(B) = {1,3,4}
- T{C} = {2,4}
- R x S(A,B) = {

  (1,1),(1,3),(1,4),

  (1,1),(1,3),(1,4),

  (2,1),(2,3),(2,4),

  (3,1),(3,3),(3,4)}
- select * from R, S

- R x S x T(A,B,C) = {

  (1,1,2),(1,3,2),(1,4,2),

  (1,1,2),(1,3,2),(1,4,2),

  (2,1,2),(2,3,2),(2,4,2),

  (3,1,2),(3,3,2),(3,4,2),

  (1,1,4),(1,3,4),(1,4,4),

  (1,1,4),(1,3,4),(1,4,4),

  (2,1,4),(2,3,4),(2,4,4),

  (3,1,4),(3,3,4),(3,4,4)}
- select * from R, S, T

Two-Table Join

- R(A) = {1,1,2,3}
- S(B) = {1,3,4}
- R $\bowtie_{A=B}$ S = {

  (1,1),~~(1,3)~~,~~(1,4)~~,

  (1,1),~~(1,3)~~,~~(1,4)~~,

  ~~(2,1)~~,~~(2,3)~~,~~(2,4)~~,

  ~~(3,1)~~,**(3,3)**,~~(3,4)~~} = {(1,1),(1,1),(3,3)}
- Join condition between attributes from the two tables
- Only those tuples from the Cartesian product that satisfy the join condition are included in the result

- select * from R, S

  where **A = B**
- Condition does not have to be equality
- select * from R, S

  where **A > B**
  - **{(2,1), (3,1)}**

Multiple-Table Join

- R(A) = {1,1,2,3}
- S(B) = {1,3,4}
- T{C} = {2,4}
- select * from R, S, T

  where **A=B and B>C**
- If there is no condition for a table, Cartesian product is performed for that table

- $R \bowtie_{A=B} S \bowtie_{B>C} T(A,B,C) = \{$

  ~~(1,1,2),(1,3,2),(1,4,2),~~

  ~~(1,1,2),(1,3,2),(1,4,2),~~

  ~~(2,1,2),(2,3,2),(2,4,2),~~

  ~~(3,1,2),~~(3,3,2),~~(3,4,2),~~

  ~~(1,1,4),(1,3,4),(1,4,4),~~

  ~~(1,1,4),(1,3,4),(1,4,4),~~

  ~~(2,1,4),(2,3,4),(2,4,4),~~

  ~~(3,1,4),(3,3,4),(3,4,4)~~} = {(3,3,2)}

Duplicate Attribute Names

- Product(maker, model, type)
- PC(model, speed, ram, hd, price)
- select * from Product, PC
  - schema: (maker, **Product.model**, type, **PC.model**, speed, ram, hd, price)
  - select **Product.model**, maker, price from Product, PC
  - select **P.model**, maker, PC.price from **Product P**, PC

Join Query Examples

- Product(maker, model, type)
- PC(model, speed, ram, hd, price)
- select * from Product P, PC

  where **P.model = PC.model**
- select P1.maker, PC.model AS pc_model, L.model AS laptop_model

  from **Product P1, Product P2**, PC, Laptop L

  where **P1.maker = P2.maker and P1.model = PC.model and P2.model = L.model and PC.price > L.price**
  - Find the (PCs, laptop) pairs produced by the same maker for which the PC price is larger than the laptop price
  - Multiple instances of a table can appear in a query. They have to be renamed as the attributes are renamed.

Abstract Evaluation Model

- select P1.maker, PC.model AS pc_model, L.model AS laptop_model

  from Product P1, Product P2, PC, Laptop L

  where P1.maker = P2.maker and P1.model = PC.model and P2.model = L.model and PC.price > L.price

-
  ```
  For each tuple P1 in table Product
      For each tuple P2 in table Product
          For each tuple PC in table PC
              For each tuple L in table Laptop
                  if P1.maker = P2.maker and P1.model = PC.model and P2.model =
                  L.model and PC.price > L.price
                  then add(P1.maker, PC.model, L.model) to the result
  ```

Abstract Evaluation Model for General Queries

SELECT [DISTINCT] [SUM | COUNT | AVG] result_table

FROM table$_1$, table$_2$, …

[WHERE table_predicates AND join_conditions]

[GROUP BY grouping_attributes

[HAVING agg_condition]]

[ORDER BY sorting_attributes]

[UNION [ALL]] [INTERSECT] [EXCEPT]

- **The evaluation model for joins is first applied to the entire WHERE clause**
- **Everything else is evaluated on the result of the join evaluation**

Examples

- Computers
- TPCH