**Lesson 3**

* **Relational Algebra, a Primer:**
  + A procedural query language that:
    - Takes a relation as input
    - Applies relational algebra operator
    - Outputs another relation
  + Relation is like a **TABLE**, Tuple is like a **ROW**, and Attribute is like a **COLUMN**.
* **Symbols (check CS 331 notes for this):**
  + Projection, Selection, Union, Set difference, Cross product
* **Projection**:
  + **Used to project required attributes from a relation**.
  + **Note**: Removes duplicate values.

**Table:** R

|  |  |  |
| --- | --- | --- |
| A | B | C |
| 1 | 2 | 4 |
| 2 | 2 | 3 |
| 3 | 2 | 3 |
| 4 | 3 | 4 |

What is π(BC)?

|  |  |
| --- | --- |
| B | C |
| 2 | 4 |
| 2 | 3 |
| 3 | 4 |
|  |  |

* **Selection:**
  + **Used to select required tuples from a relation?**
    - Using the same relation

**Table:** R

|  |  |  |
| --- | --- | --- |
| A | B | C |
| 1 | 2 | 4 |
| 2 | 2 | 3 |
| 3 | 2 | 3 |
| 4 | 3 | 4 |

What is π (σ(C > 3)R )? **NEED TO INCLUDE PROJECTION to display**

**Table**: R

|  |  |  |
| --- | --- | --- |
| A | B | C |
| 1 | 2 | 4 |
| 4 | 3 | 4 |

* **Union:**
  + **Same as in set theory:**
    - Attributes (columns) needs to be the same
    - Results are relations and sets at the same time
    - UNION command (2 select statements)

TABLE: A

h y u

TABLE: B

h z q

* + A UNION OF B
    - U((h)A, (h)B)
  + U(A, B) 🡪 IF they have the **same amount** of attributes AND they have the **same datatype**.
    - So (h)A and (h)B have to have the same data type, (y)A and (z)B have to have the same data type, and (u)A and (q)B have to have the same datatype.
* **Set Difference (-):**
  + **Same as in set theory:**
    - Both relations should have the **same set of attributes**
    - EXCEPT operator
    - 2 SELECT statements
  + EX:

TABLE: A

h y u

1 1 1

TABLE: B

h z q

1 1 1

2 2 2

Relational Algebra:

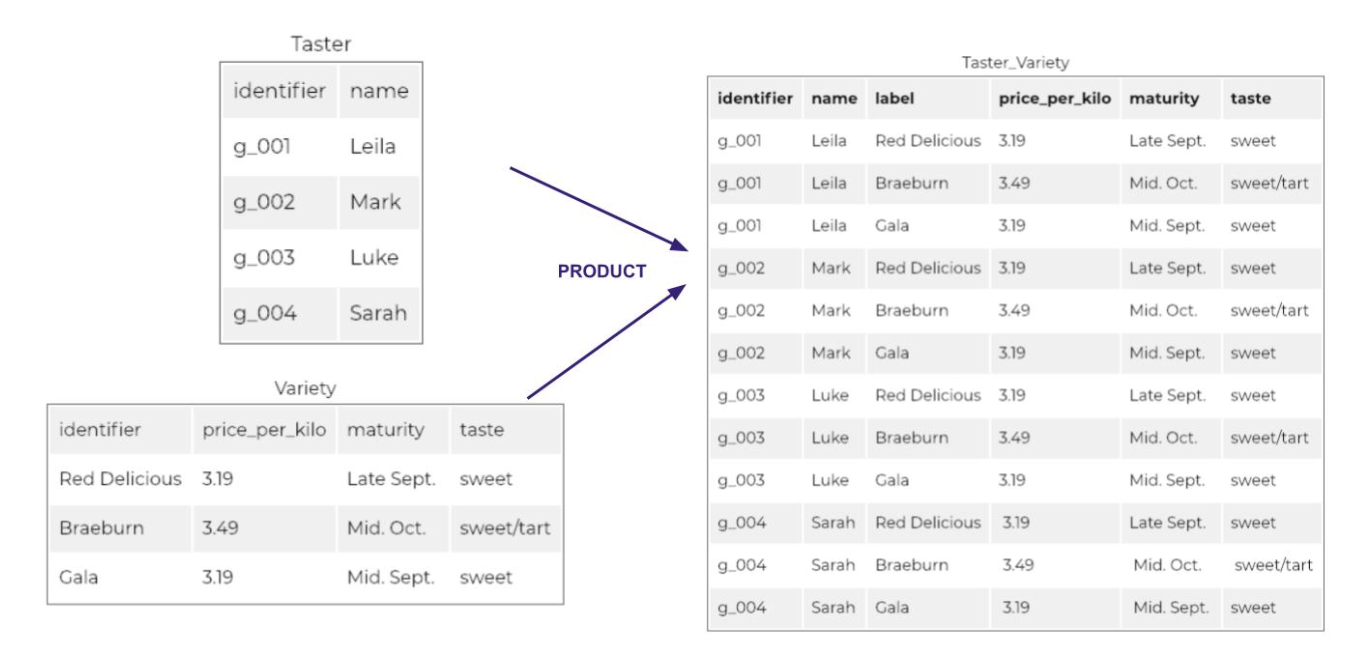
A – B = Nothing

SQL:

A EXCEPT B 🡪 returns 0 tuples (rows)

B EXCEPT A 🡪 returns 1 tuple (the 2nd row)

* **Cross Product(X):**
  + **Enforced by:**
    - Returns all attributes of A followed by each attribute of B
    - The “,” operator
    - **SELECT \* from TABLE\_A, TABLE\_B**
  + **Returns a table with a combination of every row**
  + EX:



* **SQL Constraints**:
  + **NOT NULL**:
    - **Enforces** that an attribute cannot have NULL value.
      * By default \_\_\_\_\_\_\_
    - **EX:**

Create table students (

Id int NOT NULL

);

* + **UNIQUE:**
    - **Prevents two identical records in an attribute**.
      * USE WISELY
      * Put the UNIQUE command after the attribute definition in your relation.
    - **EX:**

Create table students (

Id int NOT NULL UNIQUE

);

* + **Primary Key**:
    - **Unique key that identifies each tuple (row) in the relation (table)**
      * The column is going to be UNIQUE (automatically)
      * You need primary keys in your data model \_\_\_\_\_\_\_
    - EX:
  + **Foreign Key:**
    - **In the simplest way, a foreign key is a column in one table matches another column in another table.**
      * So in the main table it’s a **PRIMARY KEY** and in the current table, it’s a foreign key
    - EX:

Create table students (

Id INT PRIMARY KEY NOT NULL

);

create table student\_interests (

id INT PRIMARY KEY NOT NULL,

student\_ID INT references students(ID)

);

* **Check**:
  + **Enforces adherence to a condition defined in the schema**
    - .
  + EX:

Create table students (

Id INT PRIMARY KEY NOT NULL

Age int NOT NULL check (age > 0)

);

* **Drop Constraints:**
  + Enforced by:
    - Indexes, UNIQUE command, PRIMARY KEY constraint
  + EX:

ALTER TABLE students DROP CONSTRAINT students\_age\_check;

Create table students (

Id INT PRIMARY KEY NOT NULL

Age int NOT NULL check (age > 0)

);

Create table students (

id INT PRIMARY KEY NOT NULL

age int NOT NULL **check (age > 0)** //check is removed

);

* **Postgres Architecture**:
  + Client-Server Relationship;
  + pgAdmin – Monitor the server