# Unit 3. Relational Database Model

Structure of RDBMS and Terminology, Database Schema and Schema Diagram, Keys: Super, Candidates, Primary, Foreign, Composite etc. and Relationship: Introduction to Relational Algebra, Relational Algebra Operations: Select, Project, Cartesian Product, Union, Set Difference, Natural Join, Outer Join.

# Relational Database Management System(RDBMS)

- A relational database management system (RDBMS) is a database management system that is based on a relational model.
- The relational model uses the basic concept of a relation or table.
- RDBMS is the basis for SQL, and for database systems like MS SQL Server, IBM DB2, Oracle, MySQL

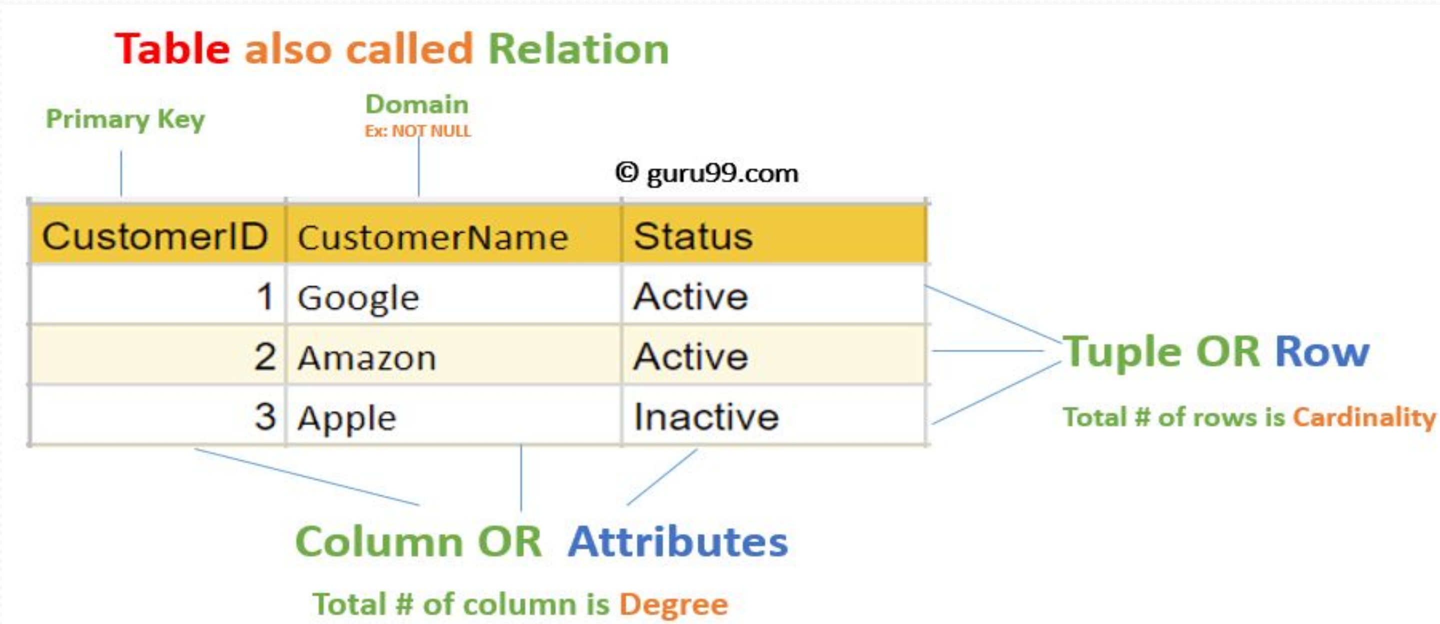
# Relational Model (RM)

- Relational Model represents how data is stored in Relational Databases.
- A relational database stores data in the form of relations (tables).
- Consider a relation STUDENT with attributes ROLL_NO, NAME, ADDRESS, PHONE and DOB as shown below.

**Student**

| ROLL_NO | NAME | ADDRESS | PHONE | DOB |
|---|---|---|---|---|
| 1 | Ram | Kathmandu | 9841000000 | 1995 |
| 2 | Sita | Lalitpur | 9851100000 | 1997 |

- In the relational model, data are stored as **tables**
- **Tables** – In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.




| CustomerID | CustomerName | Status |
|---|---|---|
| 1 | Google | Active |
| 2 | Amazon | Active |
| 3 | Apple | Inactive |

- **Attribute:** Each column in a Table. Attributes are the properties which define a relation. ROLL_NO, NAME
- **Relation Schema:** a set of relational tables and associated items that are related to one another.
- **Relation Instance:** The set of tuples of a relation at a particular instance of time is called as relation instance. Table 1 shows the relation instance of STUDENT at a particular time. It can change whenever there is insertion, deletion or updation in the database.
- **Tuple:** Each row in the relation is known as tuple. The above relation contains 2 tuples

- **Key Integrity:** Every relation in the database should have at least one set of attributes which defines a tuple uniquely. Those set of attributes is called key. e.g.; ROLL_NO in STUDENT is a key. No two students can have same roll number. So a key has two properties:
  - It should be unique for all tuples.
  - It can't have NULL values.
- **Referential Integrity:** When one attribute of a relation can only take values from other attribute of same relation or any other relation, it is called referential integrity.
- **Degree:** The total number of attributes which in the relation is called the degree of the relation
- **Cardinality:** Total number of rows present in the Table.

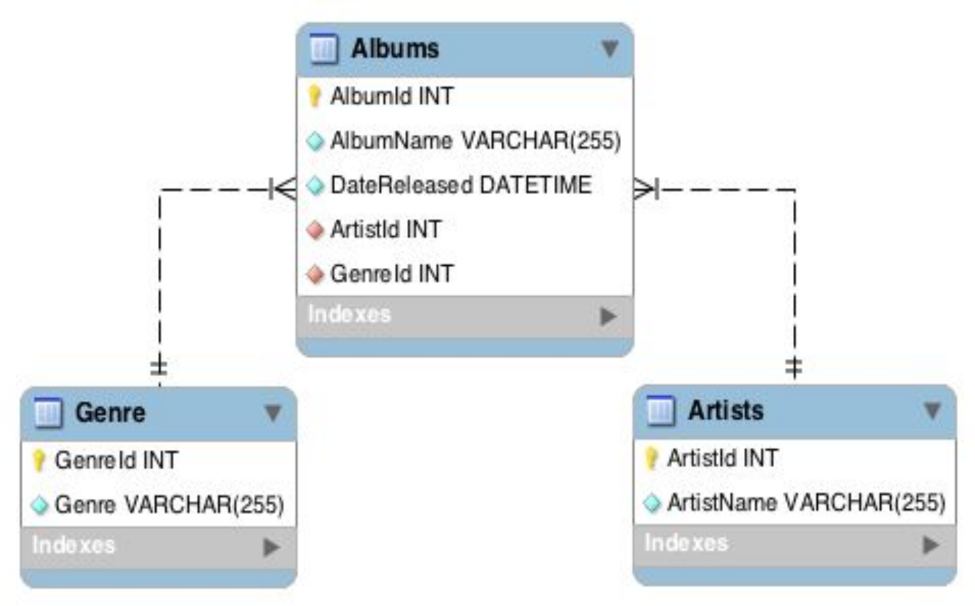
- **Column:** The column represents the set of values for a specific attribute.
- **Domain** – Every attribute has some pre-defined value and scope which is known as domain

**Properties of Relations**

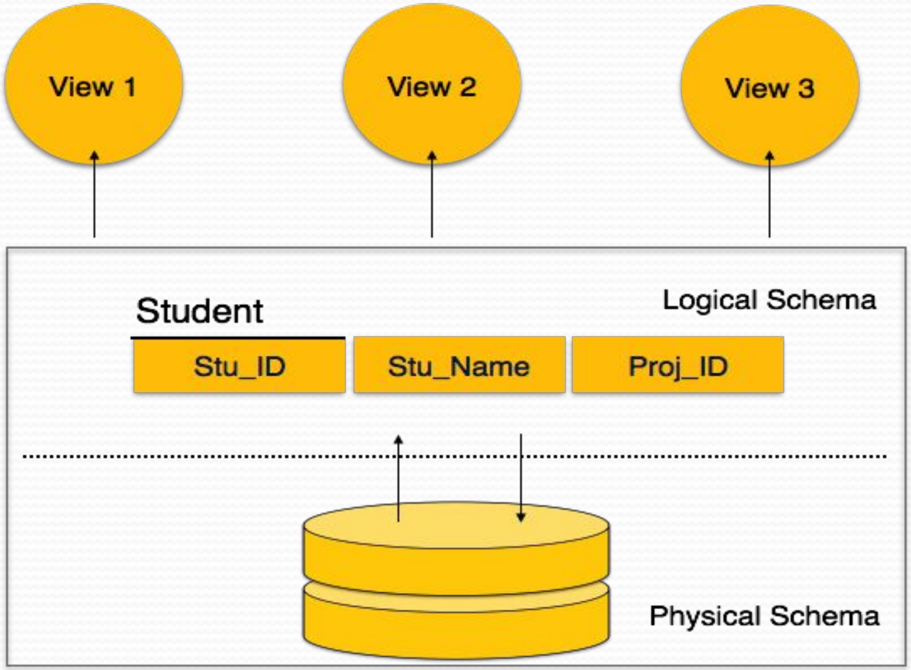
- Values are atomic.
- Column values are of the same kind.
- Each row is unique.
- The sequence of columns is insignificant.
- The sequence of rows is insignificant.
- Each column must have a unique name

# Database Schema

- A database schema is the skeleton structure that represents the logical view of the entire database.
- It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.
- **Example**:



- **Physical Database Schema** – This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.
- **Logical Database Schema** – This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.



# Integrity Constraints

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to the database.

# Types of Integrity Constraint

- Domain Constraints
  - Default constraints
  - Check constraints
  - Not null constraints
- Entity Integrity constraints
- Referential Integrity constraints
- Key constraints

# Domain Constraints

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

| ID | NAME | SEMENSTER | AGE |
|---|---|---|---|
| 1000 | Tom | 1$^{st}$ | 17 |
| 1001 | Johnson | 2$^{nd}$ | 24 |
| 1002 | Leonardo | 5$^{th}$ | 21 |
| 1003 | Kate | 3$^{rd}$ | 19 |
| 1004 | Morgan | 8$^{th}$ | A |

Not allowed. Because AGE is an integer attribute

- **NOT NULL constraint**
  - By default, a column can hold NULL values. If you do not want a column to have a NULL value, then you need to define such a constraint on this column specifying that NULL is now not allowed for that column.
  - A NULL is not the same as no data, rather, it represents unknown data.
- **DEFAULT constraint**
  - The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.
- **CHECK Constraint**
  - The CHECK Constraint enables a condition to check the value being entered into a record. If the condition evaluates to false, the record violates the constraint and isn't entered the table.

# Entity Integrity Constraints

- The **Entity integrity** constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.

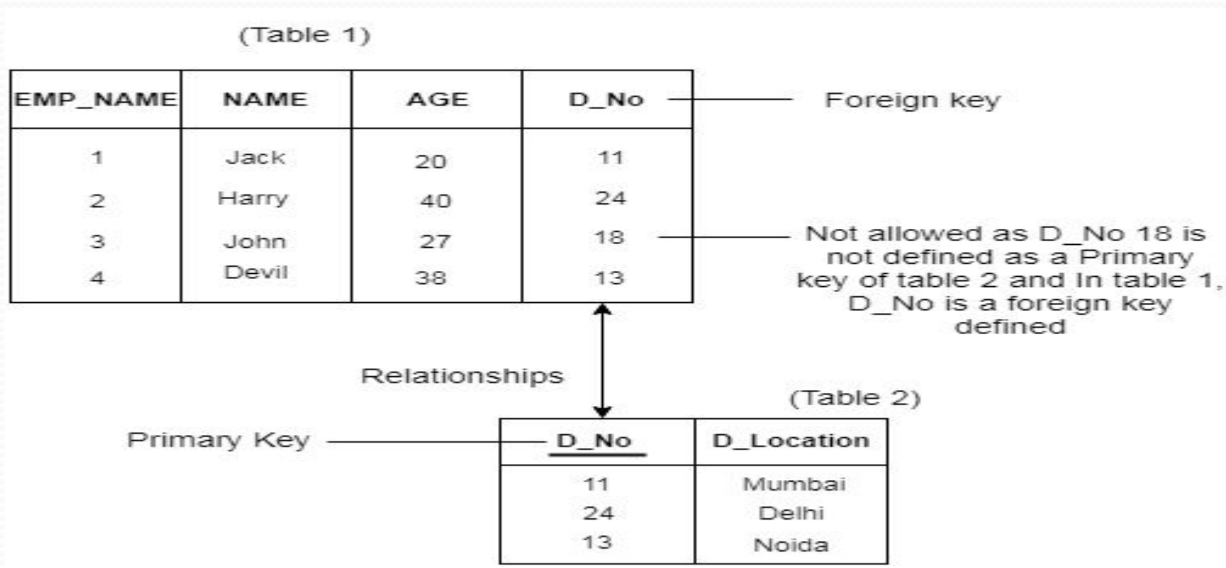
**EMPLOYEE**

| EMP_ID | EMP_NAME | SALARY |
|---|---|---|
| 123 | Jack | 30000 |
| 142 | Harry | 60000 |
| 164 | John | 20000 |
| | Jackson | 27000 |

Not allowed as primary key can't contain a NULL value

# Referential Integrity Constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.



# Key Constraints

- Keys are the entity set that is used to identify an entity within its entity set uniquely.

| ID | NAME | SEMENSTER | AGE |
|---|---|---|---|
| 1000 | Tom | 1st | 17 |
| 1001 | Johnson | 2nd | 24 |
| 1002 | Leonardo | 5th | 21 |
| 1003 | Kate | 3rd | 19 |
| 1002 | Morgan | 8th | 22 |

Not allowed. Because all row must be unique

# Keys in DBMS

- Key is an attribute or set of attributes which helps you to identify a row(tuple) in a relation(table).
- They allow you to find the relation between two tables.
- Keys help you uniquely identify a row in a table by a combination of one or more columns in that table.
- Key is also helpful for finding unique record or row from the table.

# Types of Keys in DBMS

- Candidate Key
- Super Key
- Primary Key
- Alternate Key
- Foreign Key
- Composite Key

# Candidate Key

- CANDIDATE KEY is a set of attributes that uniquely identify tuples in a table.
- Candidate Key is a super key with no repeated attributes.
- The Primary key should be selected from the candidate keys.
- Every table must have at least a single candidate key.
- A table can have multiple candidate keys but only a single primary key.


| StudID | Roll No | First Name | LastName | Email |
|---|---|---|---|---|
| 1 | 11 | Tom | Price | abc@gmail.com |
| 2 | 12 | Nick | Wright | xyz@gmail.com |
| 3 | 13 | Dana | Natan | mno@yahoo.com |

# Super Key

- The set of attributes which can uniquely identify a tuple is known as Super Key.
- Adding zero or more attributes to candidate key generates super key.
- A candidate key is a super key but vice versa is not true.

| EmpSSN | EmpNum | Empname |
|---|---|---|
| 9812345098 | AB05 | Shown |
| 9876512345 | AB06 | Roslyn |
| 199937890 | AB07 | James |

- In the above-given example, EmpSSN and EmpNum name are superkeys.

# Primary Key

- PRIMARY KEY is a column or group of columns in a table that uniquely identify every row in that table.
- The Primary Key can't be a duplicate meaning the same value can't appear more than once in the table.
- A table cannot have more than one primary key.

**Rules for defining Primary key:**

- Two rows can't have the same primary key value
- It must for every row to have a primary key value.
- The primary key field cannot be null.
- The value in a primary key column can never be modified or updated if any foreign key refers to that primary key.
- In the following example, StudID is a Primary Key.

# Primary Key…

- In the following example, StudID is a Primary Key.

| studID | Roll No | First Name | LastName | Email |
|---|---|---|---|---|
| 1 | 11 | Tom | Price | abc@gmail.com |
| 2 | 12 | Nick | Wright | xyz@gmail.com |
| 3 | 13 | Dana | Natan | mno@yahoo.com |

# Alternate Key

- **ALTERNATE KEYS** is a column or group of columns in a table that uniquely identify every row in that table.
- A table can have multiple choices for a primary key but only one can be set as the primary key.
- All the keys which are not primary key are called an Alternate Key.

**Example:**

- In this table, StudID, Roll No, Email are qualified to become a primary key. But since StudID is the primary key, **Roll No, Email** becomes the alternative key.

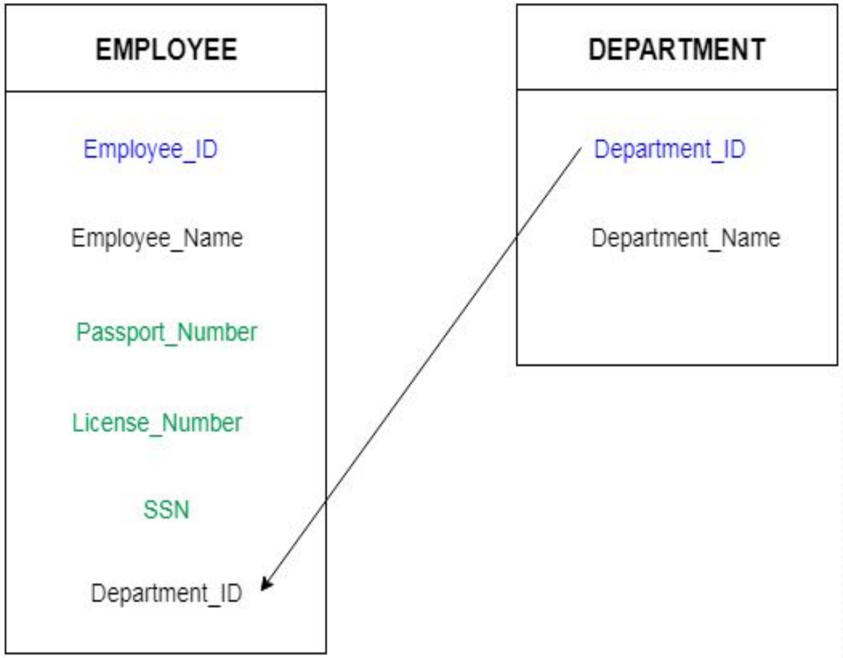
| studID | Roll No | First Name | LastName | Email |
|---|---|---|---|---|
| 1 | 11 | Tom | Price | abc@gmail.com |
| 2 | 12 | Nick | Wright | xyz@gmail.com |
| 3 | 13 | Dana | Natan | mno@yahoo.com |

# Foreign Key

- Foreign keys are the column of the table which is used to point to the primary key of another table.
- The purpose of Foreign keys is to maintain data integrity and allow navigation between two different instances of an entity.
- It acts as a cross-reference between two tables as it references the primary key of another table.

# Foreign Key…

- In a company, every employee works in a specific department, and employee and department are two different entities. So we can't store the information of the department in the employee table. That's why we link these two tables through the primary key of one table.
- We add the primary key of the DEPARTMENT table, Department_Id as a new attribute in the EMPLOYEE table.
- Now in the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.



# Composite Key

- **Composite key** is a combination of two or more columns that uniquely identify rows in a table. The combination of columns guarantees uniqueness, though individually uniqueness is not guaranteed. Hence, they are combined to uniquely identify records in a table.

| Suit | Value | No. times played |
|---|---|---|
| Hearts | Ace | 5 |
| Diamonds | Three | 2 |
| Hearts | Jack | 3 |
| Clubs | Three | 5 |
| Spades | Five | 1 |

One is not enough to identify, but both combined make a unique value

# Operations in Relational Model

- Four basic operations performed on relational database model are
- ***Insert, update, delete and select***
- Insert is used to insert data into the relation
- Delete is used to delete tuples from the table.
- Update allows you to change the values of some attributes in existing tuples.
- Select allows you to choose a specific range of data.

Whenever one of these operations are applied, integrity constraints specified on the relational database schema must never be violated.

# Insert Operation

- The insert operation gives values of the attribute for a new tuple which should be inserted into a relation.

| CustomerID | CustomerName | Status |
|---|---|---|
| 1 | Google | Active |
| 2 | Amazon | Active |
| 3 | Apple | Inactive |

INSERT

| CustomerID | CustomerName | Status |
|---|---|---|
| 1 | Google | Active |
| 2 | Amazon | Active |
| 3 | Apple | Inactive |
| 4 | Alibaba | Active |

# Update Operation

- Update allows you to change the values of some attributes in existing tuples.
- You can see that in the below-given relation table **CustomerName= 'Apple' is updated from Inactive to Active.**

| CustomerID | CustomerName | Status |
|---|---|---|
| 1 | Google | Active |
| 2 | Amazon | Active |
| 3 | Apple | Inactive |
| 4 | Alibaba | Active |

UPDATE

| CustomerID | CustomerName | Status |
|---|---|---|
| 1 | Google | Active |
| 2 | Amazon | Active |
| 3 | Apple | $\boxed{\text{Active}}$ |
| 4 | Alibaba | Active |

# Delete Operation

- Delete is used to delete tuples from the table.

| CustomerID | CustomerName | Status |
|---|---|---|
| 1 | Google | Active |
| 2 | Amazon | Active |
| 3 | Apple | Active |
| 4 | Alibaba | Active |


| CustomerID | CustomerName | Status |
|---|---|---|
| 1 | Google | Active |
| 2 | Amazon | Active |
| 4 | Alibaba | Active |

# Select Operation

- Select allows you to choose a specific range of data.

| CustomerID | CustomerName | Status |
|---|---|---|
| 1 | Google | Active |
| 2 | Amazon | Active |
| 4 | Alibaba | Active |

SELECT

| CustomerID | CustomerName | Status |
|---|---|---|
| 2 | Amazon | Active |

# Formal and informal term in relational model

| Informal term | Formal term |
|---|---|
| Table | Relation |
| Column/fields | Attributes |
| All possible column values | Domain |
| Row | Tuple |
| Table definition | Schema of Relation |

# Best Practices for creating a Relational Model

- Data need to be represented as a collection of relations
- Each relation should be depicted clearly in the table
- Rows should contain data about instances of an entity
- Columns must contain data about attributes of the entity
- Cells of the table should hold a single value
- Each column should be given a unique name
- No two rows can be identical
- The values of an attribute should be from the same domain

# Advantages of using Relational model

- **Simplicity**: A relational data model is simpler than the hierarchical and network model.
- **Structural Independence**: The relational database is only concerned with data and not with a structure. This can improve the performance of the model.
- **Easy to use**: The relational model is easy as tables consisting of rows and columns is quite natural and simple to understand
- **Query capability**: It makes possible for a high-level query language like SQL to avoid complex database navigation.
- **Data independence**: The structure of a database can be changed without having to change any application.
- **Scalable**: Regarding a number of records, or rows, and the number of fields, a database should be enlarged to enhance its usability.

# Disadvantages of using Relational model

- Few relational databases have limits on field lengths which can't be exceeded.
- Relational databases can sometimes become complex as the amount of data grows, and the relations between pieces of data become more complicated.
- Complex relational database systems may lead to isolated databases where the information cannot be shared from one system to another.

# Relational Algebra

- Relational Algebra is a procedural query language used to manipulate relational databases. It is a theoretical foundation for relational databases, and its operators define how to extract and manipulate data from a relational database.
- It consists of a set of operations that take one or two relations as input and produce a new relation as their result.
- It uses operators to perform queries. An operator can be either **unary** or **binary**.
- **Types of operations in relational algebra**
  1. Basic Operations
  2. Derived Operations

- **Basic operations:**
  - ***Selection*** (σ) Selects a subset of rows from relation.
  - ***Projection*** (π) Selects a subset of columns from relation.
  - ***Cross-product*** (×) Allows us to combine two relations.
  - ***Set-difference*** (−) Tuples in reln. 1, but not in reln. 2.
  - ***Union*** (U) Tuples in reln. 1 and in reln. 2.
  - ***Rename***( ρ) Use new name for the Tables or fields.
- **Derived operations:**
  - ***Intersection*** (∩)
  - ***join***(⋈),
  - ***division*(÷)**

# Notation

The operations have their own symbols.

| Operation | Symbol |
|---|---|
| Union | $\cup$ |
| Intersection | $\cap$ |
| Set difference | - |

| Operation | Symbol |
|---|---|
| Projection | $\pi$ |
| Selection | $\sigma$ |
| Cartesian product | $\times$ |
| Join | $\bowtie$ |
| Left outer join | ⟕ |
| Right outer join | ⟖ |
| Full outer join | ⟗ |

Schema:

branch( b_name, assets)

Depositor(cname,acc_no)

Loan( loan_no, b_name,amount)

Borrower(cname,loan_no)

Account ( acc_no, balance)

Customer (cname,c_street,c_city)

Branch

| B_name | Assets |
|---|---|
| New road | 1500000 |
| Bhaktapur | 1000000 |
| Lagankhel | 1200000 |

Account

| Acc_no | Balance |
|---|---|
| 100 | 10000 |
| 101 | 15000 |
| 102 | 25000 |

Depositor

| Cname | Acc_no |
|---|---|
| Geeta | 102 |
| Hari | 100 |
| Ram | 101 |

Customer

| Cname | C_Street | C_city |
|---|---|---|
| Ram | S1 | Ktm |
| Geeta | S5 | ktm |
| Hari | S2 | Lalitpur |
| Rita | S3 | Bhaktapur |
| John | S1 | Bhaktapur |

Loan

| Loan_no | B_name | Amount |
| --- | --- | --- |
| L100 | Bhaktapur | 100000 |
| L101 | Lalitpur | 200000 |
| L102 | Bhaktapur | 50000 |

Borrower

| Cname | Loan_no |
| --- | --- |
| Hari | L102 |
| Rita | L100 |
| John | L101 |

# Select Operation(σ)

- The select operation selects tuples that satisfy a given predicate.
- It is denoted by sigma (σ).
- Notation: σ p(r)
- **Where:**
- **σ** is used for selection prediction
  **r** is used for relation
  **p** is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like =, ≠, ≥, <, >, ≤.

1.Find the customers who live in bhaktapur.

RA expression:

σ c_city = “Bhaktapur”(Customer)

Output:

Customer

| Cname | C_Street | C_city |
|---|---|---|
| Rita | S3 | Bhaktapur |
| John | S1 | Bhaktapur |

Customer

| Cname | C_Street | C_city |
|---|---|---|
| Ram | S1 | Ktm |
| Geeta | S5 | ktm |
| Hari | S2 | Lalitpur |
| Rita | S3 | Bhaktapur |
| John | S1 | Bhaktapur |

2. Find the customers who live in S5 street in Ktm.

σ c_city = “ktm” AND c_street= “S5” (Customer)

Output:

Customer

| Cname | C_Street | C_city |
|---|---|---|
| Geeta | S5 | ktm |

- Find the loans that have amount greater than 50000.

σ amount>50000(Loan)

Loan

| Loan_no | B_name | Amount |
|---|---|---|
| L100 | Bhaktapur | 100000 |
| L101 | Lalitpur | 200000 |
| L102 | Bhaktapur | 50000 |

Output:

Loan

| Loan_no | B_name | Amount |
|---|---|---|
| L100 | Bhaktapur | 100000 |
| L101 | Lalitpur | 200000 |

- Find the loans that have either loan less than 100000 or loan issued from lalitpur branch.

RA expression:

σ amount<100000 OR B_name="lalitpur" (Loan)

Output:

Loan

| Loan_no | B_name | Amount |
|---|---|---|
| L101 | Lalitpur | 200000 |
| L102 | Bhaktapur | 50000 |

# Project Operation(∏)

- This operation shows the list of those attributes that we wish to appear in the result. Rest of the attributes are eliminated from the table.
- It is denoted by ∏.
- Notation: ∏ A1, A2, … An (r)
- **Where**
- **A1**, **A2**, **A3** is used as an attribute name of relation **r**.
- **Degree**
  - Number of attributes in <attribute list>
- **Duplicate elimination**
  - Result of PROJECT operation is a set of distinct tuples

1.Find the customer name who have account at a bank.

$\prod$ cname(Depositor)

Depositor

| Cname |
|---|
| Geeta |
| Hari |
| Ram |

Depositor

| Cname | Acc_no |
|---|---|
| Geeta | 102 |
| Hari | 100 |
| Ram | 101 |

2.Find the customer name who have loan at bank.

$\prod$ cname(Borrower)

Borrower

| Cname |
|---|
| Hari |
| Rita |
| John |

Borrower

| Cname | Loan_no |
|---|---|
| Hari | L102 |
| Rita | L100 |
| John | L101 |

3.Find the customer name and city where they live.

$\prod$ cname, c_city(Customer)

Customer

| Cname | C_city |
|---|---|
| Ram | Ktm |
| Geeta | ktm |
| Hari | Lalitpur |
| Rita | Bhaktapur |
| John | Bhaktapur |

Customer

| Cname | C_Street | C_city |
|---|---|---|
| Ram | S1 | Ktm |
| Geeta | S5 | ktm |
| Hari | S2 | Lalitpur |
| Rita | S3 | Bhaktapur |
| John | S1 | Bhaktapur |

# Project Operation with Selection

- Find the customer name who live in ktm city.

$\pi$ cname($\sigma$ c_city="ktm"(customer))

Customer

| Cname | C_Street | C_city |
| --- | --- | --- |
| Ram | S1 | Ktm |
| Geeta | S5 | ktm |
| Hari | S2 | Lalitpur |
| Rita | S3 | Bhaktapur |
| John | S1 | Bhaktapur |

Output:

Customer

| Cname |
| --- |
| Ram |
| Geeta |

- Find the customer name and customer street who live in S1 street.

$\pi$ cname, c_street($\sigma$ c_street="S1"(customer))

Output:

Customer

| Cname | C_Street |
| --- | --- |
| Ram | S1 |
| John | S1 |

# Project Operation with Selection

- Find the customer name and customer street who live in S1 street or who live in lalitpur

π cname, c_street(σ c_street="S1" OR c_city="lalitpur"(customer))

Customer

| Cname | C_Street | C_city |
|---|---|---|
| Ram | S1 | Ktm |
| Geeta | S5 | ktm |
| Hari | S2 | Lalitpur |
| Rita | S3 | Bhaktapur |
| John | S1 | Bhaktapur |

Output:

Customer

| Cname | C_Street |
|---|---|
| Ram | S1 |
| Hari | S2 |
| John | S1 |

- Find the customer name and customer street who live in S1 street and who live in lalitpur.

π cname, c_street(σ c_street="S1" AND c_city="lalitpur"(customer))

Output:

Customer

| Cname | C_Street |
|---|---|

- Find the branch name whose assets is greater than or equals to 1200000.

$\pi$ b_name($\sigma$ assets >= 1200000(Branch))

- Find the account no whose balance is between 12000 to 25000.

$\pi$ acc_no($\sigma$ balance >= 12000 AND balance<=25000(Account))

- Find the loan details of loan whose loan amount is between 10000 to 100000

$\sigma$ amount>=10000 AND amount<=100000(loan)

- Find the loan_no and amount whose loan is maintained at bhaktapur branch.

$\pi$ loan_no, amount($\sigma$ branch = ''bhaktapur''(Loan))

## Union Operation

- Suppose there are two tuples R and S. The union operation contains all the tuples that are either in R or S or both in R & S.
- It eliminates the duplicate tuples. It is denoted by ∪.

Notation: $R \cup S$

- A union operation must hold the following condition:

  -R and S must have the attribute of the same number.

  -Duplicate tuples are eliminated automatically.

  - Data type of attributes must be same

- Example:

Find the customers name who have bank transactions.

$\pi$ c_name(depositor) U $\pi$ c_name(borrower)

Output:

| Cname |
|---|
| Geeta |
| Hari |
| Ram |
| Rita |
| John |

Borrower

| Cname | Loan_no |
|---|---|
| Hari | L102 |
| Rita | L100 |
| John | L101 |

Depositor

| Cname | Acc_no |
|---|---|
| Geeta | 102 |
| Hari | 100 |
| Ram | 101 |

# Intersection Operation

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in both R & S.
- It is denoted by intersection ∩.

Notation: $R \cap S$

- **Example:**

Find the customers name who have both account and loan at bank.

$\pi$ c_name(depositor) $\cap$ $\pi$ c_name(borrower)

Output:

| Cname |
| --- |
| Hari |

Borrower

| Cname | Loan_no |
| --- | --- |
| Hari | L102 |
| Rita | L100 |
| John | L101 |

Depositor

| Cname | Acc_no |
| --- | --- |
| Geeta | 102 |
| Hari | 100 |
| Ram | 101 |

# Set Difference

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in R but not in S.
- It is denoted by intersection minus (-).

Notation: R - S

Example

Depositor

| Cname | Acc_no |
|---|---|
| Geeta | 102 |
| Hari | 100 |
| Ram | 101 |

Find the Customers who have account but not loan at the bank.

π c_name(depositor) - π c_name(borrower)

Output:

| Cname |
|---|
| Geeta |
| Ram |

Borrower

| Cname | Loan_no |
|---|---|
| Hari | L102 |
| Rita | L100 |
| John | L101 |

# Cartesian product

- The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product.
- It is denoted by X.
- Notation: R X S

Example:

Depositor X Borrower

Borrower

| Cname | Loan_no |
|---|---|
| Hari | L102 |
| Rita | L100 |
| John | L101 |

| Cname | Acc_no | Cname | Loan No |
|---|---|---|---|
| Geeta | 102 | Hari | L102 |
| Geeta | 102 | Rita | L100 |
| Geeta | 102 | John | L101 |
| Hari | 100 | Hari | L102 |
| Hari | 100 | Rita | L100 |
| Hari | 100 | John | L101 |
| Ram | 101 | Hari | L102 |
| Ram | 101 | Rita | L100 |
| Ram | 101 | John | L101 |

Depositor

| Cname | Acc_no |
|---|---|
| Geeta | 102 |
| Hari | 100 |
| Ram | 101 |

- Find the Customers who have both account and loan at the bank.

$\pi_{depositor.cname}(\sigma_{Depositor.cname=Borrower.cname}(Depositor \times Borrower))$

| Cname | Acc_no | Cname | Loan No |
|---|---|---|---|
| Geeta | 102 | Hari | L102 |
| Geeta | 102 | Rita | L100 |
| Geeta | 102 | John | L101 |
| Hari | 100 | Hari | L102 |
| Hari | 100 | Rita | L100 |
| Hari | 100 | John | L101 |
| Ram | 101 | Hari | L102 |
| Ram | 101 | Rita | L100 |
| Ram | 101 | John | L101 |

Borrower

| Cname | Loan_no |
|---|---|
| Hari | L102 |
| Rita | L100 |
| John | L101 |

Depositor

| Cname | Acc_no |
|---|---|
| Geeta | 102 |
| Hari | 100 |
| Ram | 101 |

Output:

| Cname |
|---|
| Hari |

# Rename Operation ($\rho$)

- The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** $\rho$.
- **Notation** – $\rho_{x}$ (E)
- Where the result of expression **E** is saved with name of **x**.

1. Renaming relation:

   $\rho_{cust1}$ (Customer)

2. Renaming Attributes:

   $\rho_{cname,Cust\_street,cust\_city}$ (Customer)

Customer

| Cname | C_Street | C_city |
|---|---|---|
| Ram | S1 | Ktm |
| Geeta | S5 | ktm |
| Hari | S2 | Lalitpur |
| Rita | S3 | Bhaktapur |
| John | S1 | Bhaktapur |

# Assignment Operation (← )

- The assignment operation assign the right side result to the left side relation. The evaluation of an assignment does not result in any relation being displayed to the user. The result of the expression to the right of the ← is assigned to the relation variable on the left of the ← .

Notation: Temp ← Result

Example:

- Find the customer name and their loan am

Loan

| Loan_no | B_name | Amount |
|---|---|---|
| L100 | Bhaktapur | 100000 |
| L101 | Lalitpur | 200000 |
| L102 | Bhaktapur | 50000 |

$R1 \leftarrow \sigma_{borrower.Loan\_no=loan.Loan\_no}(Loan \times Borrower)$

Borrower

| Cname | Loan_no |
|---|---|
| Hari | L102 |
| Rita | L100 |
| John | L101 |

$\pi_{Cname, Amount}(R1)$

## Division Operator(÷):

- The division operator takes two relation and builds another relation consisting of values of an attribute of one relation that match all the values in the other relation.
- The divide operation is the opposite of the product operation. It is suited to queries that include the phrase "for all".

Example:

A÷B1

| Sno |
|---|
| S1 |
| S2 |
| S3 |
| S4 |

A÷B2

| Sno |
|---|
| S1 |
| S4 |

A÷B3

| Sno |
|---|
| S1 |

Relation A

| Sno | Pno |
|---|---|
| S1 | P1 |
| S1 | P2 |
| S1 | P3 |
| S1 | P4 |
| S2 | P1 |
| S2 | P2 |
| S3 | P2 |
| S4 | P2 |
| S4 | P4 |

Relation B1

| Pno |
|---|
| P2 |

Relation B2

| Pno |
|---|
| P2 |
| P4 |

Relation B3

| Pno |
|---|
| P1 |
| P2 |
| P3 |

## Join Operation (⋈):

- We understand the benefits of taking a Cartesian product of two relations, which gives us all the possible tuples that are paired together. But it might not be feasible for us in certain cases to take a Cartesian product where we encounter huge relations with thousands of tuples having a considerable large number of attributes.
- **Join** is a combination of a Cartesian product followed by a selection process. A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.

Notation: $R1 \bowtie R2$

Types:

- Theta(θ) Join
- Inner Join
- Natural Join
- Outer Join

- **Theta(θ) Join:**
- Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol θ.
- Notation: R1 ⋈θ R2
- R1 and R2 are relations having attributes (A1, A2, .., An) and (B1, B2,.. ,Bn) such that the attributes don't have anything in common, that is R1 ∩ R2 = Φ.

Example:

Depositor ⋈depositor.cname=Customer.cname Customer

Depositor

| Cname | Acc_no |
|---|---|
| Geeta | 102 |
| Hari | 100 |
| Ram | 101 |

| Cname | Acc_no | Cname | C_Street | C_city |
|---|---|---|---|---|
| Geeta | 102 | Geeta | S5 | ktm |
| Hari | 100 | Hari | S2 | Lalitpur |
| Ram | 101 | Ram | S1 | Ktm |

Customer

| Cname | C_Street | C_city |
|---|---|---|
| Ram | S1 | Ktm |
| Geeta | S5 | ktm |
| Hari | S2 | Lalitour |
| Rita | S3 | Bhaktapur |
| John | S1 | Bhaktapur |

- Find the customers name, accno  and city who live in Ktm.

$\pi$ borrower.cname, acc_no,c_city ($\sigma$ customer.ccity="ktm" (Depositor $\bowtie$depositor.cname=Customer.cname Customer))

| Cname | Acc_no | C_city |
|---|---|---|
| Geeta | 102 | ktm |
| Ram | 101 | Ktm |

- **Inner Join or Equijoin**: When Theta join uses only **equality** comparison operator, it is said to be equijoin. The above example corresponds to equijoin.

$R1 \bowtie_{R1.B>R2.D} R2$

| A | B | C | D |
|---|---|---|---|
| A1 | 500 | C1 | 300 |
| A2 | 800 | C1 | 300 |

R1

| A | B |
|---|---|
| A1 | 500 |
| A2 | 800 |
| A3 | 200 |

R2

| C | D |
|---|---|
| C1 | 300 |
| C2 | 900 |
| C3 | 1500 |

- Theta Join

$$R1 \bowtie_{R1.B>R2.D} R2$$

R1

| A | B |
|---|---|
| A1 | 500 |
| A2 | 800 |
| A3 | 200 |

R2

| C | D |
|---|---|
| C1 | 300 |
| C2 | 900 |
| C3 | 1500 |

| A | B | C | D |
|---|---|---|---|
| A1 | 500 | C1 | 300 |
| A2 | 800 | C1 | 300 |

- **Natural Join (⋈):**
- Natural join does not use any comparison operator. It does not concatenate the way a Cartesian product does. We can perform a Natural Join only if there is at least one common attribute that exists between two relations. In addition, the attributes must have the same name and domain.
- Natural join acts on those matching attributes where the values of attributes in both the relations are same.

Example: Depositor ⋈ Borrower

Depositor

| Cname | Acc_no |
|---|---|
| Geeta | 102 |
| Hari | 100 |
| Ram | 101 |

Borrower

| Cname | Loan_no |
|---|---|
| Hari | L102 |
| Rita | L100 |
| John | L101 |

| Cname | Acc_no | Loan_no |
|---|---|---|
| Hari | 100 | L102 |

- Find the customers name who have both loan and account at the bank.

$\Pi$ cname(Depositor ⋈ Borrower)

| Cname |
| --- |
| Hari |

- **Outer Joins:**
- Theta Join, Equijoin, and Natural Join are called inner joins. An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation. Therefore, we need to use outer joins to include all the tuples from the participating relations in the resulting relation. There are three kinds of outer joins − left outer join, right outer join, and full outer join.

## Left outer join:

- Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In the left outer join, tuples in R have no matching tuples in S.
- It is denoted by ⟕.

Example:

EMPLOYEE ⟕ FACT_WORKERS

EMPLOYEE

| EMP_NAME | STREET | CITY |
|---|---|---|
| Ram | Civil line | Mumbai |
| Shyam | Park street | Kolkata |
| Ravi | M.G. Street | Delhi |
| Hari | Nehru nagar | Hyderabad |

FACT_WORKERS

| EMP_NAME | BRANCH | SALARY |
|---|---|---|
| Ram | Infosys | 10000 |
| Shyam | Wipro | 20000 |
| Kuber | HCL | 30000 |
| Hari | TCS | 50000 |

| EMP_NAME | STREET | CITY | BRANCH | SALARY |
|---|---|---|---|---|
| Ram | Civil line | Mumbai | Infosys | 10000 |
| Shyam | Park street | Kolkata | Wipro | 20000 |
| Hari | Nehru street | Hyderabad | TCS | 50000 |
| Ravi | M.G. Street | Delhi | NULL | NULL |

- **Right outer join:**
- Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In right outer join, tuples in S have no matching tuples in R.
- It is denoted by ⟖.

EMPLOYEE ⟖ FACT_WORKERS

EMPLOYEE

| EMP_NAME | STREET | CITY |
|---|---|---|
| Ram | Civil line | Mumbai |
| Shyam | Park street | Kolkata |
| Ravi | M.G. Street | Delhi |
| Hari | Nehru nagar | Hyderabad |

T_WORKERS

| IP_NAME | BRANCH | SALARY |
|---|---|---|
| n | Infosys | 10000 |
| am | Wipro | 20000 |
| Kuber | HCL | 30000 |
| Hari | TCS | 50000 |

| EMP_NAME | BRANCH | SALARY | STREET | CITY |
|---|---|---|---|---|
| Ram | Infosys | 10000 | Civil line | Mumbai |
| Shyam | Wipro | 20000 | Park street | Kolkata |
| Hari | TCS | 50000 | Nehru street | Hyderabad |
| Kuber | HCL | 30000 | NULL | NULL |

- **Full outer join:**
- Full outer join is like a left or right join except that it contains all rows from both tables.
- In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tu

attribute name.
- It is denoted by ⟗.

EMPLOYEE ⟗ FACT_WORKERS

| EMP_NAME | STREET | CITY | BRANCH | SALARY |
|---|---|---|---|---|
| Ram | Civil line | Mumbai | Infosys | 10000 |
| Shyam | Park street | Kolkata | Wipro | 20000 |
| Hari | Nehru street | Hyderabad | TCS | 50000 |
| Ravi | M.G. Street | Delhi | NULL | NULL |
| Kuber | NULL | NULL | HCL | 30000 |

EMPLOYEE

| EMP_NAME | STREET | CITY |
|---|---|---|
| Ram | Civil line | Mumbai |
| Shyam | Park street | Kolkata |
| Ravi | M.G. Street | Delhi |
| ari | Nehru nagar | Hyderabad |

_WORKERS

| P_NAME | BRANCH | SALARY |
|---|---|---|
| | Infosys | 10000 |
| m | Wipro | 20000 |
| r | HCL | 30000 |
| Hari | TCS | 50000 |

# Extended Operations

- Relational algebra operations have been extended in various ways
  - More generalized
  - More useful
- Three major extensions:
  - Generalized projection
  - Aggregate functions
  - Additional join operations*( Discussed in the previous lectures)*
- All of these appear in SQL standards

# Generalized Projection Operation

- Would like to include computed results into relations

  e.g. "Retrieve all credit accounts, computing the current 'available credit' for each account."

¤ Available credit = credit limit – current balance

*Project operation is generalized to include computed results*

- Can specify functions on attributes, as well as attributes themselves
- Can also assign names to computed values
- ¤ (Renaming attributes is also allowed, even though this is also provided by the rename operat

Written as: $\Pi_{F_1, F_2, \ldots, F_n}(E)$

- $F_i$ are arithmetic expressions
- $E$ is an expression that produces a relation
- Can also name values: $F_i$ **as** *name*

# Generalized Projection Example

- "Compute available credit for every credit account."

$\pi_{\text{cred\_id, (limit – balance) as available\_credit}}(\text{credit\_acct})$

| cred_id | limit | balance |
|---|---|---|
| C-273 | 2500 | 150 |
| C-291 | 750 | 600 |
| C-304 | 15000 | 3500 |
| C-313 | 300 | 25 |

*credit_acct*

⇒

| cred_id | available_credit |
|---|---|
| C-273 | 2350 |
| C-291 | 150 |
| C-304 | 11500 |
| C-313 | 275 |

# Aggregate Functions

- Very useful to apply a function to a collection of values to generate a single result
- Most common aggregate functions:

| | |
|---|---|
| **sum** | sums the values in the collection |
| **avg** | computes average of values in the collection |
| **count** | counts number of elements in the collection |
| **min** | returns minimum value in the collection |
| **max** | returns maximum value in the collection |

- Aggregate functions work on <u>multisets</u>, not sets

    A value can appear in the input multiple times

# Aggregate Function Examples

"Find the total amount owed to the credit company."

$\mathcal{G}_{\mathbf{sum}(balance)}(credit\_acct)$

| |
|---|
| 4275 |

| cred_id | limit | balance |
|---|---|---|
| C-273 | 2500 | 150 |
| C-291 | 750 | 600 |
| C-304 | 15000 | 3500 |
| C-313 | 300 | 25 |

*credit_acct*

"Find the maximum available credit of any account."

$\mathcal{G}_{\mathbf{max}(available\_credit)}(\Pi_{(limit - balance)\ \mathbf{as}\ available\_credit}(credit\_acct))$

| |
|---|
| 11500 |

# Grouping and Aggregation

- Sometimes need to compute aggregates on a *per-item* basis
- Back to the puzzle database:

  *puzzle_list*(*puzzle_name*)

  *completed*(*person_name*, *puzzle_name*)

| puzzle_name |
|---|
| altekruse |
| soma cube |
| puzzle box |

*puzzle_list*

- Examples:
  - How many puzzles has *each person* completed?
  - How many people have completed each puzzle?

| person_name | puzzle_name |
|---|---|
| Alex | altekruse |
| Alex | soma cube |
| Bob | puzzle box |
| Carl | altekruse |
| Bob | soma cube |
| Carl | puzzle box |
| Alex | puzzle box |
| Carl | soma cube |

*completed*

| puzzle_name |
|---|
| altekruse |
| soma cube |
| puzzle box |

*puzzle_list*

| person_name | puzzle_name |
|---|---|
| Alex | altekruse |
| Alex | soma cube |
| Bob | puzzle box |
| Carl | altekruse |
| Bob | soma cube |
| Carl | puzzle box |
| Alex | puzzle box |
| Carl | soma cube |

*completed*

"How many puzzles has each person completed?"

$_{person\_name}\mathcal{G}_{\mathbf{count}(puzzle\_name)}(completed)$

- First, input relation *completed* is grouped by unique values of *person_name*
- Then, **count**(*puzzle_name*) is applied separately to each group

$$_{person\_name}\mathcal{G}_{\mathbf{count}(puzzle\_name)}(completed)$$

Input relation is
grouped by *person_name*

| person_name | puzzle_name |
|---|---|
| Alex | altekruse |
| Alex | soma cube |
| Alex | puzzle box |
| Bob | puzzle box |
| Bob | soma cube |
| Carl | altekruse |
| Carl | puzzle box |
| Carl | soma cube |

Aggregate function is
applied to each group

| person_name | |
|---|---|
| Alex | 3 |
| Bob | 2 |
| Carl | 3 |

# Distinct Values

"How many puzzles has each person completed?"

- Each puzzle appears multiple times now.

| person_name | puzzle_name | seconds |
|---|---|---|
| Alex | altekruse | 350 |
| Alex | soma cube | 45 |
| Bob | puzzle box | 240 |
| Carl | altekruse | 285 |
| Bob | puzzle box | 215 |
| Alex | altekruse | 290 |

*completed_times*

- Need to count distinct occurrences of each puzzle's name

$$_{person\_name}\mathcal{G}_{\textbf{count-distinct}(puzzle\_name)}(completed\_times)$$