UNIT II - THE RELATIONAL DATA MODEL & ALGEBRA

PRASHANT TOMAR

Relational Model

- The relational model represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.
- The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.

Relational Model Concepts

- **Attribute**

Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME etc.

- **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.

Tuple

It is nothing but a single row of a table, which contains a single record.

- **Relation Schema**

A relation schema represents the name of the relation with its attributes.

- **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.

- **Relation instance**

Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.

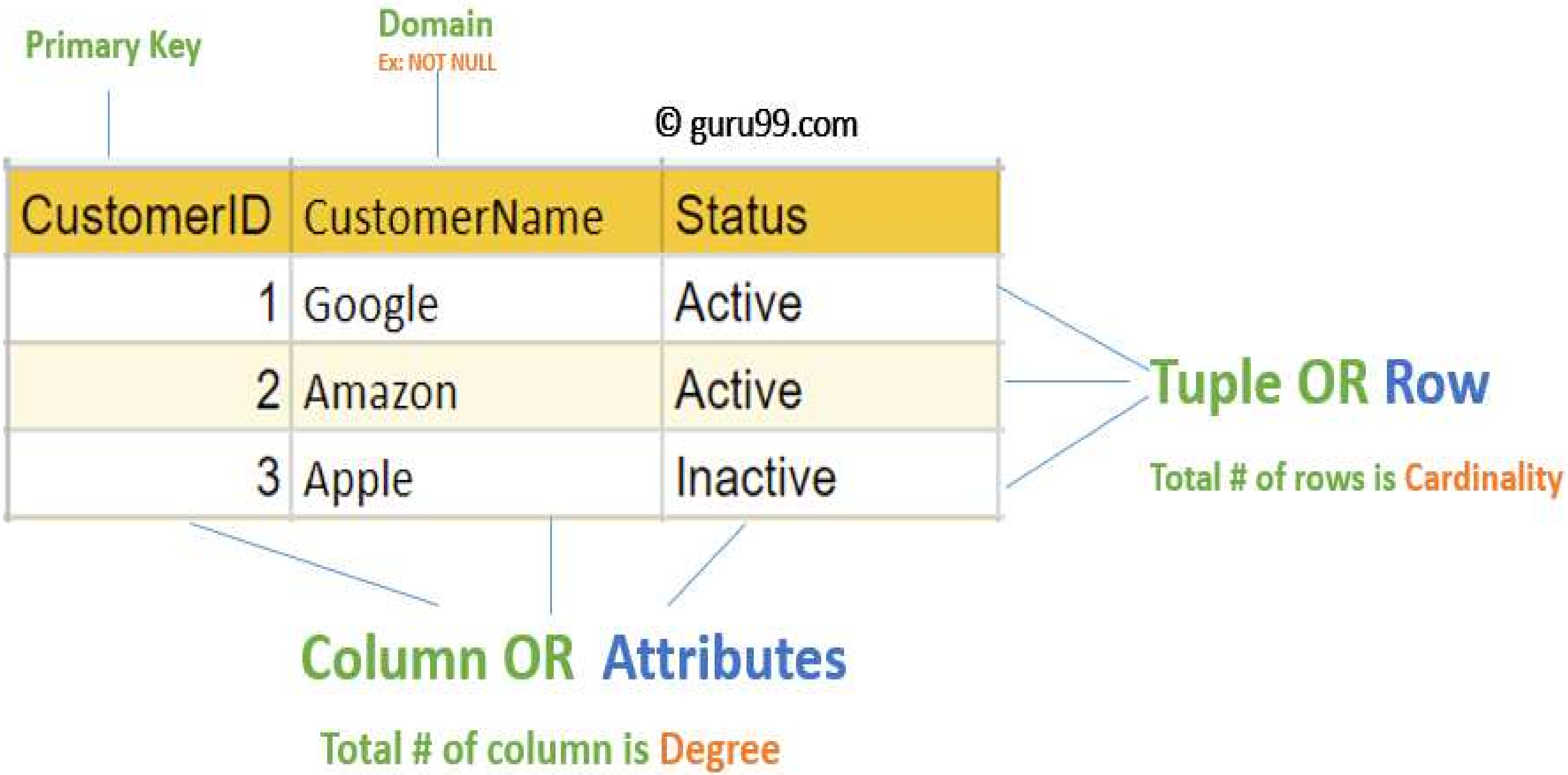
- **Relation key**

Every row has one, two or multiple attributes, which is called relation key.

- **Attribute domain**

Every attribute has some pre-defined value and scope which is known as attribute domain

Table also called Relation



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

Relational Integrity constraints

- Relational Integrity constraints is referred to conditions which must be present for a valid relation. These integrity constraints are derived from the rules in the mini-world that the database represents.
- There are many types of integrity constraints. Constraints on the Relational database management system is mostly divided into three main categories are:

1. Domain constraints
2. Key constraints
3. Referential integrity constraints

Domain Constraints

- Domain constraints can be violated if an attribute value is not appearing in the corresponding domain or it is not of the appropriate data type.
- Domain constraints specify that within each tuple, and the value of each attribute must be unique. This is specified as data types which include standard data types integers, real numbers, characters, Booleans, variable length strings, etc.

Exam: The example shown demonstrates creating a domain constraint such that CustomerName is not NULL

```
Create DOMAIN CustomerName
CHECK (value not NULL)


CREATE TABLE table_name (
  column1 datatype constraint,
  column2 datatype constraint,
  column3 datatype constraint,
 );
```

Key constraints

- An attribute that can uniquely identify a tuple in a relation is called the key of the table. The value of the attribute for different tuples in the relation has to be unique.

Example: In the given table, CustomerID is a key attribute of Customer Table. It is most likely to have a single key for one customer, CustomerID =1 is only for the CustomerName =" Google".

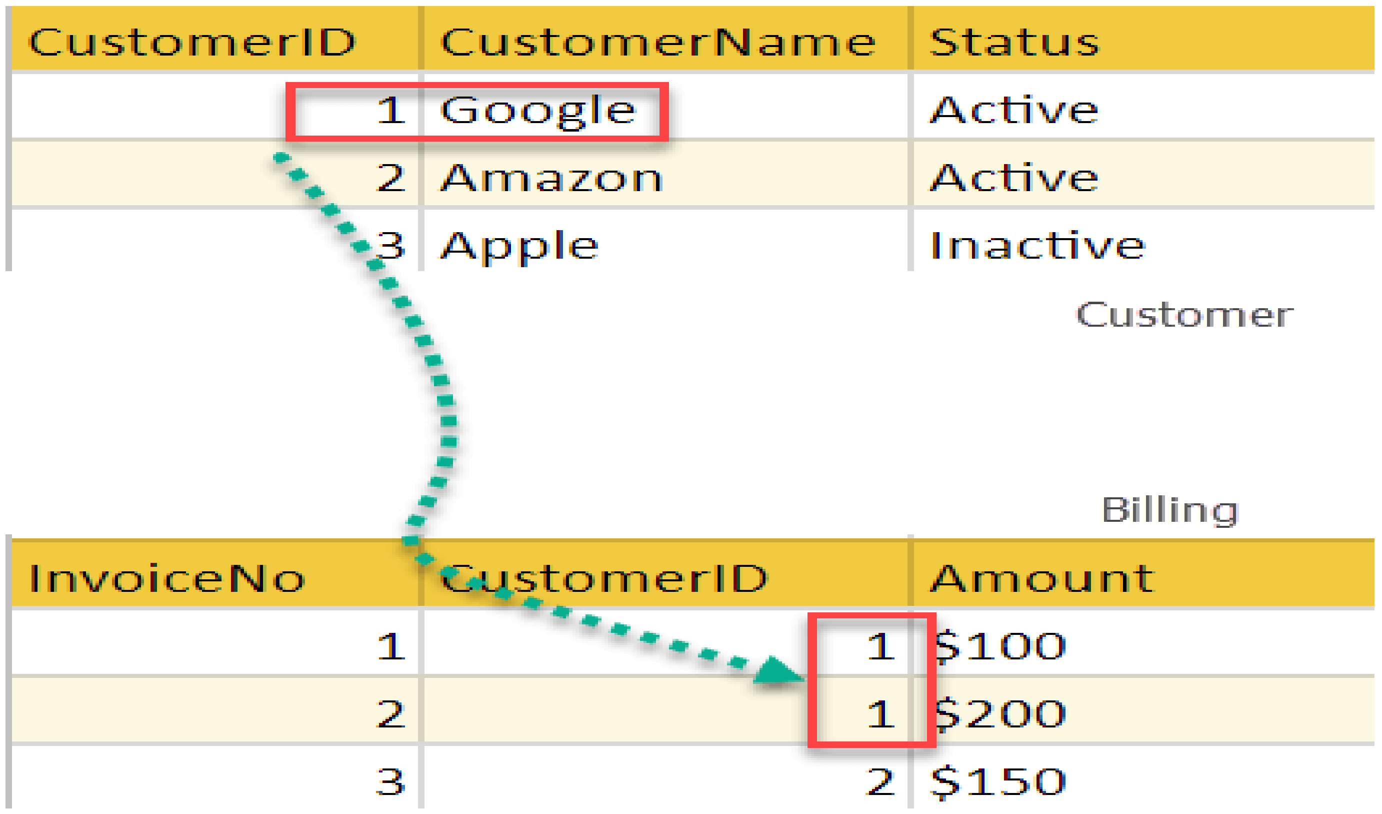
CustomerID	CustomerName	Status
1	**Google**	**Active**
2	**Amazon**	**Active**
3	**Apple**	**Inactive**

Referential integrity constraints

- Referential integrity constraints is base on the concept of **Foreign Keys**.
- A foreign key is an important attribute of a relation which should be referred to in other relationships.
- Referential integrity constraint state happens where relation refers to a key attribute of a different or same relation. However, that key element must exist in the table.

- **Example:**

In the below example, we have 2 relations, Customer and Billing. Tuple for CustomerID=1 is referenced twice in the relation Billing. So we know CustomerName=Google has billing amount $300



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

Customer

Billing

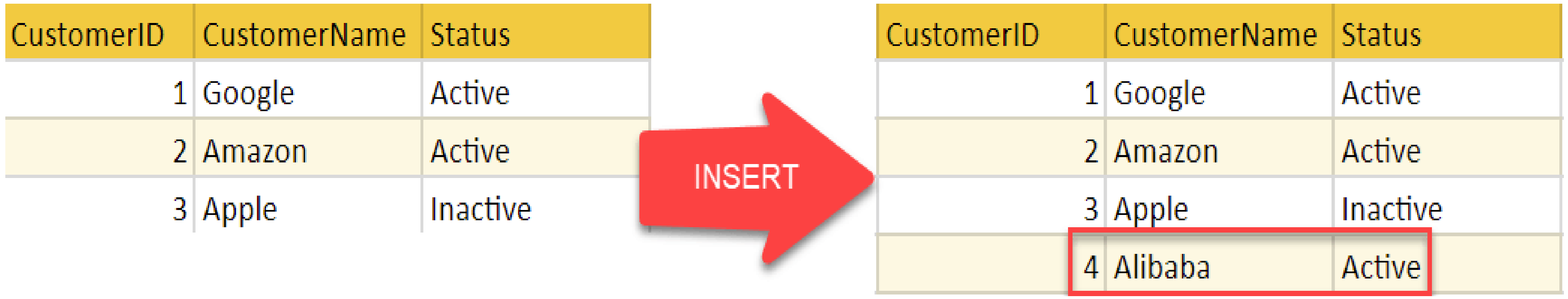
InvoiceNo	CustomerID	Amount
1	1	$100
2	1	$200
3	2	$150

Operations in Relational Model

- Four basic update 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.
  - Modify 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.

Inset 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

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

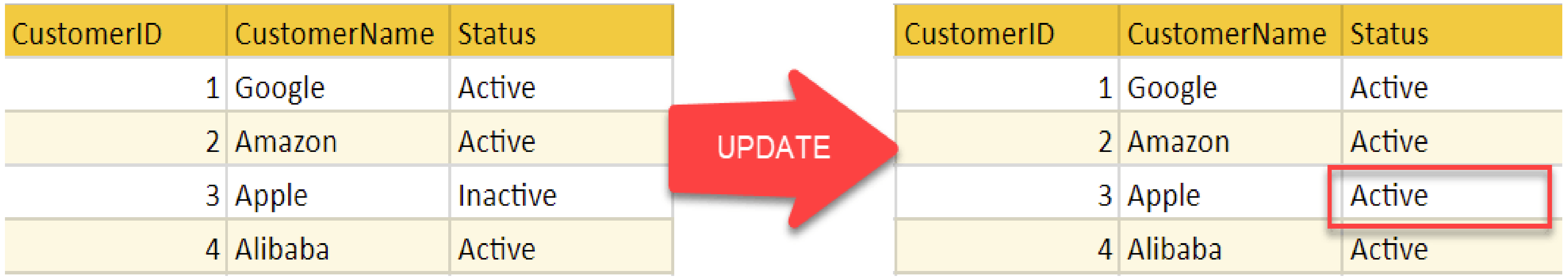
INSERT INTO *table_name*

(*column1*,*column2*,*column3*,...)

VALUES (*value1*, *value2*, *value3*, ...);

Update Operation

- You can see that in the below-given relation table CustomerName= 'Apple' is updated from Inactive to Active.



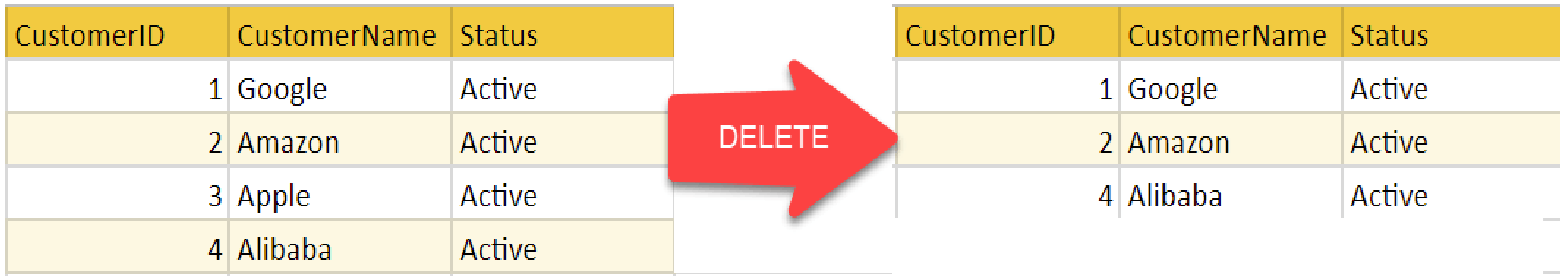
CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

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

```
UPDATE table
SET column1 = expression1,
    column2 = expression2,
    column_n = expression_n
[WHERE conditions];
```

Delete Operation

- To specify deletion, a condition on the attributes of the relation selects the tuple to be deleted.



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

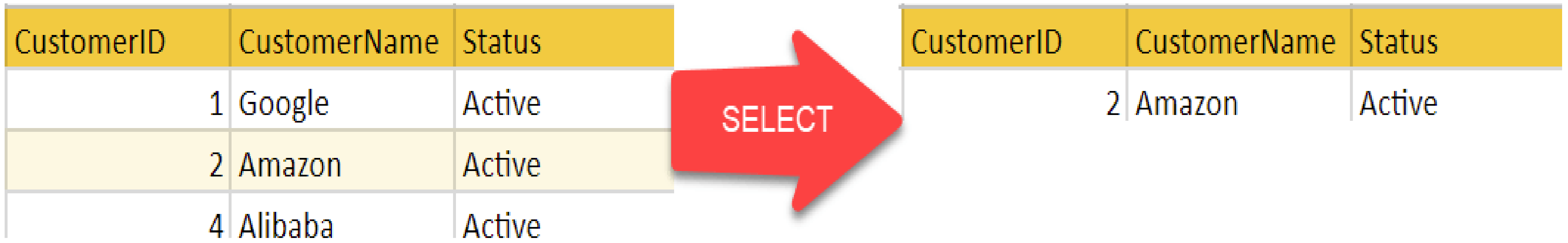
CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
4	Alibaba	Active

```
DELETE FROM table
[WHERE conditions];
```

The Delete operation could violate referential integrity if the tuple which is deleted is referenced by foreign keys from other tuples in the same database.

Select Operation

- To specify selection, all and condition on the attributes of the relation selects the tuple.



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

CustomerID	CustomerName	Status
2	Amazon	Active

```
SELECT expressions
FROM tables
[WHERE conditions];
```

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.

Functional Dependencies

Functional Dependency

- Functional Dependency is when one attribute determines another attribute in a DBMS system. Functional Dependency plays a vital role to find the difference between good and bad database design.
- A *functional dependency* (FD) is a relationship between two attributes, typically between the PK and other non-key attributes within a table.
- In other words, functional dependency is a constraint that describes the relationship between attributes in a relation

Example

Employee number	Employee Name	Salary	City
1	ANKIT	38000	GREATER NOIDA
2	VAIBHAV	50000	NOIDA
3	AMIT	25000	DELHI

- In this example, if we know the value of Employee number, we can obtain Employee Name, city, salary, etc.
- By this, we can say that the city, Employee Name, and salary are functionally depended on Employee number.
- A functional dependency is denoted by an arrow $\rightarrow$
- The functional dependency of **X** on **Y** is represented by **X** $\rightarrow$**Y**

Key Terms	Description
Axiom	Axioms is a set of inference rules used to infer all the functional dependencies on a relational database.
Decomposition	It is a rule that suggests if you have a table that appears to contain two entities which are determined by the same primary key then you should consider breaking them up into two different tables.
Dependent	It is displayed on the right side of the functional dependency diagram.
Determinant	It is displayed on the left side of the functional dependency Diagram.
Union	It suggests that if two tables are separate, and the PK is the same, you should consider putting them. together

Rules of Functional Dependencies

- **Reflexive rule**

    If **X** is a set of attributes and **Y** is subset of **X**, then **X** holds a value of **Y**.

- **Augmentation rule**

    When **x -> y** holds, and **c** is attribute set, then **ac -> bc** also holds. That is adding attributes which do not change the basic dependencies.

- **Transitivity rule**

    This rule is very much similar to the transitive rule in algebra if **x -> y** holds and **y -> z** holds, then **x -> z** also holds. **X -> y** is called as functionally that determines y.

Types of Functional Dependencies

- **Multivalued dependency**
- **Trivial functional dependency**
- **Non-trivial functional dependency**
- **Transitive dependency**

Multivalued dependency in DBMS

- Multivalued dependency occurs in the situation where there are multiple independent multivalued attributes in a single table. A multivalued dependency is a complete constraint between two sets of attributes in a relation. It requires that certain tuples be present in a relation.

Car_model	Maf_year	Color
H001	2017	Metallic
H001	2017	Green
H005	2018	Metallic
H005	2018	Blue
H010	2015	Metallic
H033	2012	Gray

- In previous example, **maf_year** and **color** are independent of each other but dependent on **car_model**. In previous example, these two columns are said to be **multivalue** dependent on **car_model**.
- This dependence can be represented like this:

car_model -> maf_year

car_model-> colour

Trivial Functional dependency

- The Trivial dependency is a set of attributes which are called a trivial if the set of attributes are included in that attribute.
- So, X -> Y is a trivial functional dependency if Y is a subset of X.

Emp_id	Emp_name
AS555	Harry
AS811	George
AS999	Kevin

- Consider this table with two columns Emp_id and Emp_name.
- {Emp_id, Emp_name} -> Emp_id is a trivial functional dependency as Emp_id is a subset of {Emp_id,Emp_name}.

Non trivial functional dependency

- Functional dependency which also known as a nontrivial dependency occurs when A->B holds true where B is not a subset of A. In a relationship, if attribute B is not a subset of attribute A, then it is considered as a non-trivial dependency.

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Apple	Tim Cook	57

- (Company} -> {CEO} (if we know the Company, we knows the CEO name)
- But CEO is not a subset of Company, and hence it's non-trivial functional dependency.

Transitive dependency

- A transitive is a type of functional dependency which happens when t is indirectly formed by two functional dependencies.

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Apple	Tim Cook	57

- **{Company} -> {CEO}** (if we know the compay, we know its CEO's name)
- **{CEO } -> {Age}** If we know the **CEO**, we know the **Age**
- Therefore according to the rule of rule of transitive dependency:
- **{ Company} -> {Age}** should hold, that makes sense because if we know the company name, we can know his age.
- **Note:** You need to remember that transitive dependency can only occur in a relation of three or more attributes.

Advantages of Functional Dependency

- Functional Dependency avoids data redundancy. Therefore same data do not repeat at multiple locations in that database
- It helps you to maintain the quality of data in the database
- It helps you to defined meanings and constraints of databases
- It helps you to identify bad designs
- It helps you to find the facts regarding the database design

Relational Algebra

- Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.
- Relational algebra is a widely used procedural query language. It collects instances of relations as input and gives occurrences of relations as output. It uses various operation to perform this action.
- Relational algebra operations are performed recursively on a relation. The output of these operations is a new relation, which might be formed from one or more input relations.

Basic Relational Algebra Operations

- **Unary Relational Operations**
  - SELECT (symbol: **σ**)
  - PROJECT (symbol: **π**)
  - RENAME (symbol: **ρ**)
- **Relational Algebra Operations From Set Theory**
  - UNION (**υ**)
  - INTERSECTION (∩)
  - DIFFERENCE (**-**)
  - CARTESIAN PRODUCT (**x**)
- **Binary Relational Operations**
  - JOIN
  - DIVISION

SELECT (σ)

- The SELECT operation is used for selecting a subset of the tuples according to a given selection condition. **Sigma(σ)** Symbol denotes it. It is used as an expression to choose tuples which meet the selection condition. Select operation selects tuples that satisfy a given predicate.

$$\boldsymbol{\sigma_p(r)}$$

- **σ** **-** is the predicate
- **r** **-** stands for relation which is the name of the table
- **p** **-** is prepositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like =, ≠, ≥, <, >, ≤.

- **For example: LOAN Relation**

BRANCH_NAME	LOAN_NO	AMOUNT
Downtown	L-17	1000
Redwood	L-23	2000
Perryride	L-15	1500
Downtown	L-14	1500
Perryride	L-16	1300

- $\sigma_{\text{BRANCH\_NAME}} = \text{"perryride"}\ (\text{LOAN})$

- **OUTPUT**

BRANCH_NAME	LOAN_NO	AMOUNT
Perryride	L-15	1500
Perryride	L-16	1300

σsubject = "database"(Books)

- **Output** − Selects tuples from books where subject is 'database'.

σsubject = "database" and price = "450"(Books)

- **Output** − Selects tuples from books where subject is 'database' and 'price' is 450.

σsubject = "database" and price = "450" or year > "2010"(Books)

Output − Selects tuples from books where subject is 'database' and 'price' is 450 or

those books published after 2010.

Projection(π)

- The projection eliminates all attributes of the input relation but those mentioned in the projection list. The projection method defines a relation that contains a vertical subset of Relation.
- This helps to extract the values of specified attributes to eliminates duplicate values. (**pi**) The symbol used to choose attributes from a relation. This operation helps you to keep specific columns from a relation and discards the other columns.

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

- Here, the projection of CustomerName and status will give

  $\Pi_{\text{CustomerName, Status}}$ (Customers)

CustomerName	Status
Google	**Active**
Amazon	**Active**
Apple	**Inactive**
Alibaba	**Active**

Union operation (∪)

- UNION is symbolized by ∪ symbol. It includes all tuples that are in tables A or in B. It also eliminates duplicate tuples. So, set A UNION set B would be expressed as:

  The result <- A ∪ B

For a union operation to be valid, the following conditions must hold –

- R and S must be the same number of attributes.
- Attribute domains need to be compatible.
- Duplicate tuples should be automatically removed.

Table A		Table B	
Column 1	Column 2	Column 1	Column 2
1	1	1	1
1	2	1	3

A ∪ **B** gives:

Table A ∪ B	
Column 1	Column 2
1	1
1	2
1	3

Set Difference (-)

- (**-**) Symbol denotes it. The result of A - B, is a relation which includes all tuples that are in A but not in B.
  - The attribute name of A has to match with the attribute name in B.
  - The two-operand relations A and B should be either compatible or Union compatible.
  - It should be defined relation consisting of the tuples that are in relation A, but not in B.
- Example: **A – B**

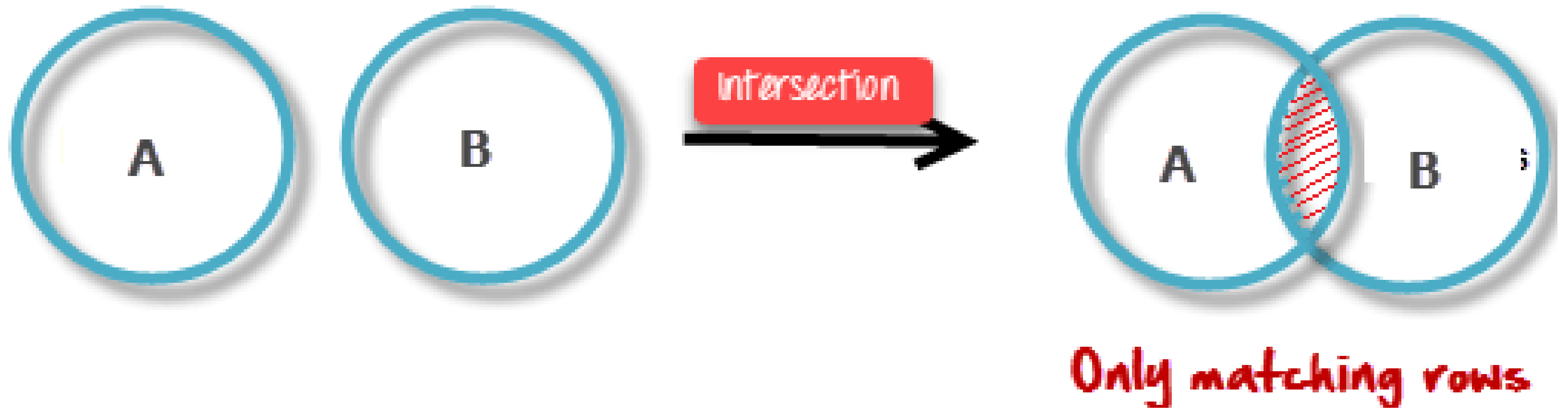
Table A – B	
Column 1	**Column 2**
1	2

Intersection

- An intersection is defined by the symbol ∩

  $A \cap B$

- Defines a relation consisting of a set of all tuple that are in both A and B. However, A and B must be union-compatible.



- Example:

A ∩ B

Table A ∩ B	
Column 1	**Column 2**
1	2

Cartesian product(X)

- This type of operation is helpful to merge columns from two relations. Generally, a Cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations.

- **Example – Cartesian product**

$$\sigma_{\text{column 2}} =_{\text{'1'}} (A \text{ X } B)$$

- **Output** – The above example shows all rows from relation A and B whose column 2 has value 1

$\sigma_{\text{column 2} = '1'}(A \times B)$	
Column 1	**Column 2**
1	1
1	1

Join Operations

- Join operation is essentially a cartesian product followed by a selection criterion.
- Join operation denoted by ⋈.
- JOIN operation also allows joining variously related tuples from different relations.

Types of JOIN:

Various forms of join operation are:

- **Inner Joins:**
  - Theta join
  - EQUI join
  - Natural join

- Outer join:
    - Left Outer Join
    - Right Outer Join
    - Full Outer Join

Inner Join:

In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded. Let's study various types of Inner Joins:

Theta Join

- The general case of JOIN operation is called a Theta join. It is denoted by symbol **θ**
- Example $\mathbf{A \bowtie_{\theta} B}$
- Theta join can use any conditions in the selection criteria. For example:

$$\mathbf{A \bowtie_{A.column\ 2 > B.column\ 2} (B)}$$

$\mathbf{A \bowtie_{A.column\ 2 > B.column\ 2} (B)}$	
Column 1	**Column 2**
1	2

EQUI join

- When a theta join uses only equivalence condition, it becomes a equi join. For example:

$$\mathbf{A} \bowtie_{\mathbf{A.column\ 2\ =\ B.column\ 2}} \mathbf{(B)}$$

$\mathbf{A} \bowtie_{\mathbf{A.column\ 2\ =\ B.column\ 2}} \mathbf{(B)}$	
Column 1	**Column 2**
1	1

- EQUI join is the most difficult operations to implement efficiently in an RDBMS and one reason why RDBMS have essential performance problems.

NATURAL JOIN (⋈)

- Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same. Example: Consider the following two tables.

C	
Num	**Square**
2	4
3	9

D	
Num	**Cube**
2	8
3	9

- C ⋈ D

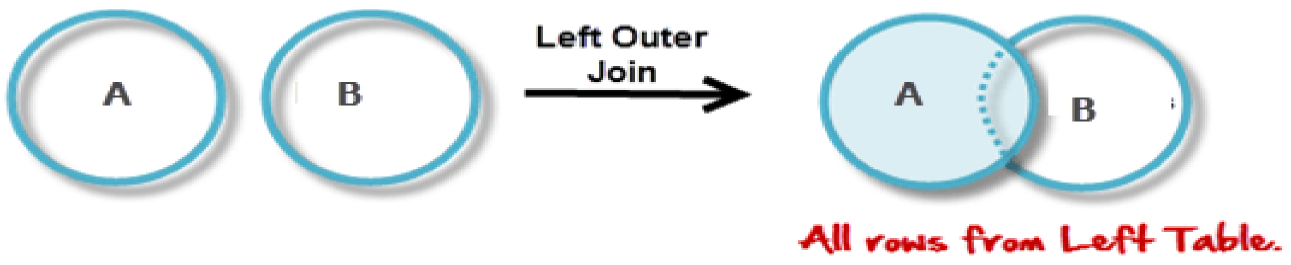
C ⋈ D		
Num	**Square**	**Cube**
2	4	8
3	9	18

OUTER JOIN

- In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.

Left Outer Join (A B)

In the left outer ⟕n, operation allows keeping all tuple in the left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.



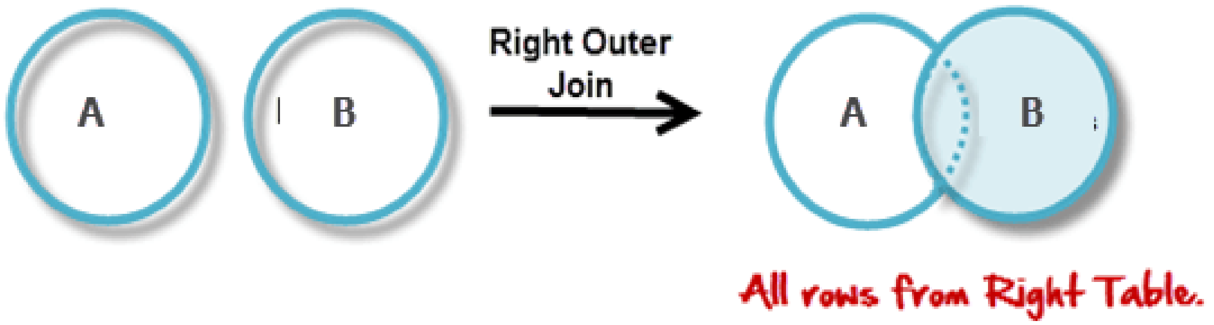
Left Table (Courses)	
A	**B**
100	Database
101	Mechanics
102	Electronics

Right Table (HoD)	
C	**D**
100	Alex
102	Maya
104	Mira

Courses ⟕ HoD			
A	**B**	**C**	**D**
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya

Right Outer Join (A ⟖ B)

- In the right outer join, operation allows keeping all tuple in the right relation. However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.



- **A ⟖ B**

Left Table (Courses)	
A	**B**
100	Database
101	Mechanics
102	Electronics

Right Table (HoD)	
C	**D**
100	Alex
102	Maya
104	Mira

Courses ⟖ HoD			
A	**B**	**C**	**D**
100	Database	100	Alex
102	Electronics	102	Maya
---	---	104	Mira

Full Outer Join (A ⟗ B)

- In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.
- **A ⟗ B**

Courses ⟗ HoD			
A	**B**	**C**	**D**
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya
---	---	104	Mira