



Unit 4- Relational Data Models

Subject Code: 303105203

Prof. S.W.Thakare
Assistant Professor,
Computer science & Engineering



CHAPTER-4

Relational Data Models

Relational Data Models Topics:

Relational Data Model:

- Introduction
- Degree,
- Cardinality.

Constraints & Keys:

- Primary Key,
- Foreign Key,
- Super Key,
- Candidate Key,
- Not Null Constraint,
- Check Constraint.

Relational Data Models Topics:

Relational Algebra Operations:

- Selection,
- Projection,
- Cross-Product,
- Rename,
- Joins (Natural & Outer Join),
- SetOperators (Union, Intersection, Set Difference),
- Aggregate Functions.

Relational Data Models:

Introduction

- 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.

Relational Model Concepts:

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.

Relational Model Concepts:

- **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

Relational Model Concepts:

Table also called **Relation**

Primary Key

Domain
Ex: NOT NULL

© guru99.com

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

Tuple OR Row
Total # of rows is **Cardinality**

Column OR Attributes
Total # of column is **Degree**

Relational Model Concepts:

Basic Structure

- Formally, given sets D_1, D_2, \dots, D_n a **relation** r is a subset of

$$D_1 \times D_2 \times \dots \times D_n$$

Thus, a relation is a set of n -tuples (a_1, a_2, \dots, a_n) where each $a_i \in D_i$

- Example: If

- $customer_name = \{\text{Jones, Smith, Curry, Lindsay, ...}\}$
/* Set of all customer names */
- $customer_street = \{\text{Main, North, Park, ...}\}$ /* set of all street names */
- $customer_city = \{\text{Harrison, Rye, Pittsfield, ...}\}$ /* set of all city names */

Then $r = \{$
 (Jones, Main, Harrison),
 (Smith, North, Rye),
 (Curry, North, Rye),
 (Lindsay, Park, Pittsfield) $\}$

is a relation over

$customer_name \times customer_street \times customer_city$

Relational Model Concepts:

Attribute Types

- Each attribute of a relation has a name
- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**; that is, indivisible
 - E.g. the value of an attribute can be an account number, but cannot be a set of account numbers
- Domain is said to be atomic if all its members are atomic
- The special value *null* is a member of every domain
- The null value causes complications in the definition of many operations
 - We shall ignore the effect of null values in our main presentation and consider their effect later

Relational Model Concepts:

Relation Schema

- A_1, A_2, \dots, A_n are *attributes*
- $R = (A_1, A_2, \dots, A_n)$ is a *relation schema*

Example:

Customer_schema = (*customer_name*, *customer_street*, *customer_city*)

- $r(R)$ denotes a *relation* r on the *relation schema* R

Example:

customer (*Customer_schema*)

Relational Model Concepts:

Relation Instance

- The current values (*relation instance*) of a relation are specified by a table
- An element t of r is a *tuple*, represented by a *row* in a table

<i>customer_name</i>	<i>customer_street</i>	<i>customer_city</i>
<i>Jones</i>	<i>Main</i>	<i>Harrison</i>
<i>Smith</i>	<i>North</i>	<i>Rye</i>
<i>Curry</i>	<i>North</i>	<i>Rye</i>
<i>Lindsay</i>	<i>Park</i>	<i>Pittsfield</i>

customer

attributes
(or columns)

tuples
(or rows)

Constraints & Keys:

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

Constraints & Keys:

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.
- 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

Constraints & Keys:

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

Constraints & Keys:

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

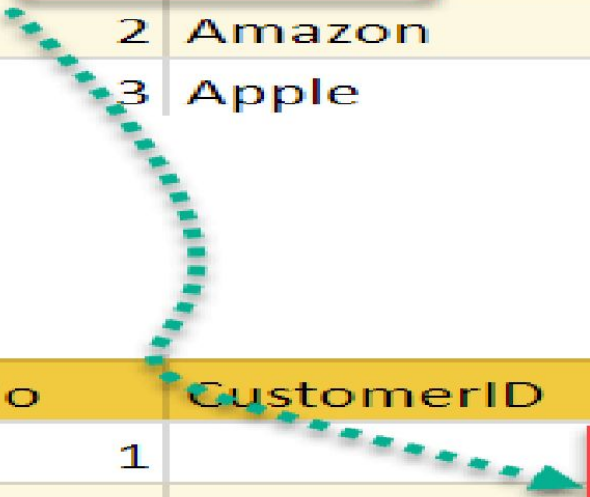
Constraints & Keys:

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

Customer

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

Billing



Keys:

- **KEYS in DBMS** 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.
- Database key is also helpful for finding unique record or row from the table.

Keys:

Types of Keys in DBMS (Database Management System)

There are mainly Eight different types of Keys in DBMS and each key has it's different functionality:

1. Super Key
2. Primary Key
3. Candidate Key
4. Alternate Key
5. Foreign Key
6. Compound Key
7. Composite Key
8. Surrogate Key

Keys:

A **superkey** is a group of single or multiple keys which identifies rows in a table. A Super key may have additional attributes that are not needed for unique identification.

Example:

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

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

Keys:

PRIMARY KEY in DBMS 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.

Example:

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

Keys:

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

Keys:

CANDIDATE KEY in SQL 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.

Properties of Candidate key:

- It must contain unique values
- Candidate key in SQL may have multiple attributes
- Must not contain null values

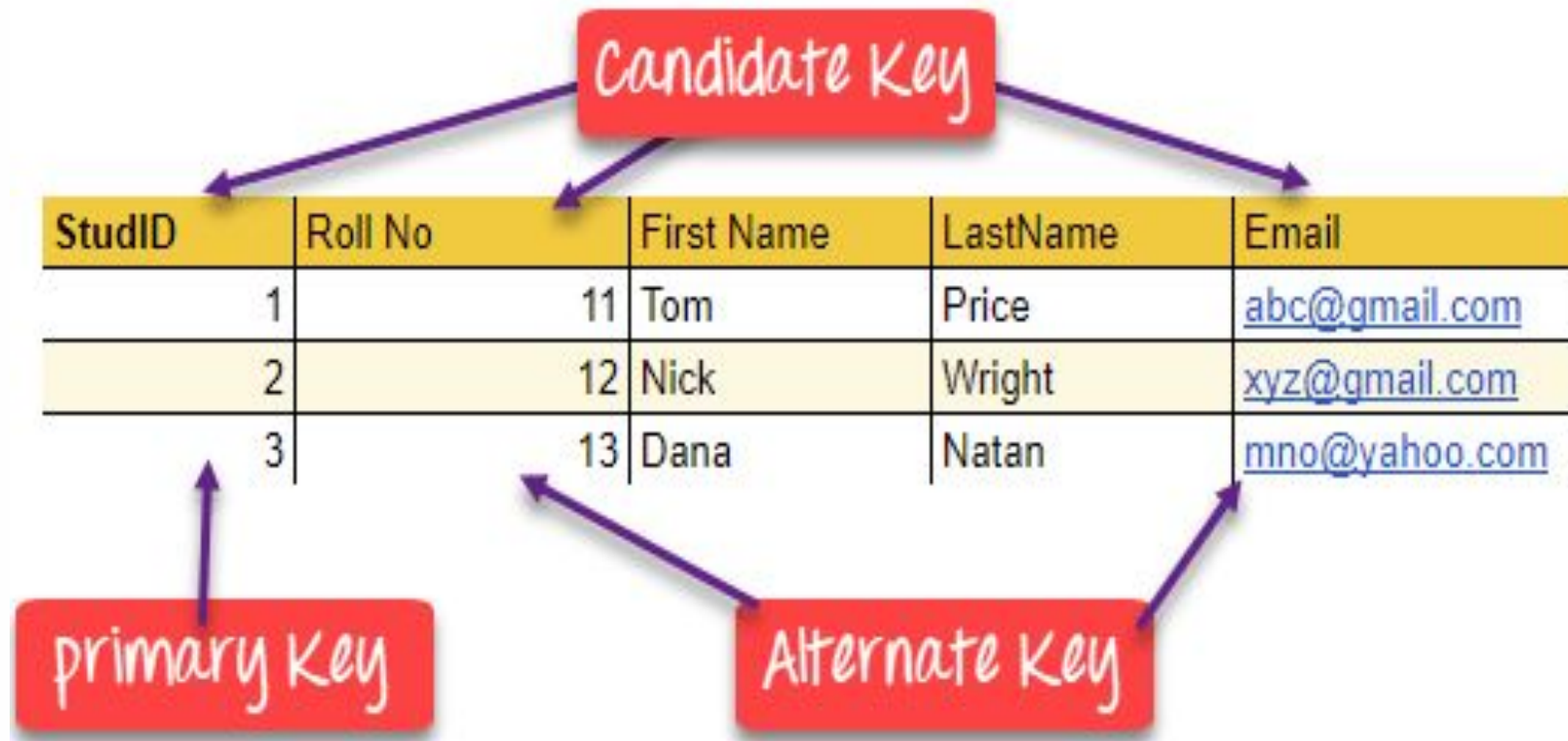
Keys:

- It should contain minimum fields to ensure uniqueness
- Uniquely identify each record in a table

Candidate key Example: In the given table Stud ID, Roll No, and email are candidate keys which help us to uniquely identify the student record in the table.

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

Keys:



Keys:

FOREIGN KEY is a column that creates a relationship between two tables.

- 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.

Example:

DeptCode	DeptName
001	Science
002	English
005	Computer

Teacher ID	Fname	Lname
B002	David	Warner
B017	Sara	Joseph
B009	Mike	Brunton



Keys:

- In this key in dbms example, we have two table, teach and department in a school.
- However, there is no way to see which search work in which department.
- In this table, adding the foreign key in Deptcode to the Teacher name, we can create a relationship between the two tables.

Teacher ID	DeptCode	Fname	Lname
B002	002	David	Warner
B017	002	Sara	Joseph
B009	001	Mike	Brunton

Not Null Constraints:

- The SQL NOT NULL forces particular values or records should not to hold a null value.
- In SQL, using constraints we can apply limits on the type of data that can be stored in the particular column of the table.
- Constraints are typically placed specified along with CREATE statement. By default, a column can hold null values.

Query:

```
CREATE TABLE Emp(  
    EmpID INT PRIMARY KEY,  
    Name VARCHAR(50),  
    Country VARCHAR(50),  
    Age int(2),  
    Salary int(10)
```

Not Null Constraints:

- Output:

Emp

EmpID	Name	Country	Age	Salary
empty				

- If you don't want to have a null column or a null value you need to define constraints like NOT NULL.
- **NOT NULL** constraints make sure that a column does not hold null values, or in other words, NOT NULL constraints make sure that you cannot insert a new record or update a record without entering a value to the specified column(i.e., NOT NULL column).
- It prevents for acceptance of NULL values. It can be applied to column-level constraints.

Not Null Constraints:

NOT NULL on CREATE a Table

- In SQL, we can add NOT NULL constraints while creating a table.
- For example, the “EMPID” will not accept NULL values when the EMPLOYEES table is created because NOT NULL constraints are used with these columns.
- Query:
- `CREATE TABLE Emp(`
- `EmpID INT NOT NULL PRIMARY KEY,`
- `Name VARCHAR (50),`
- `Country VARCHAR(50),`
- `Age int(2),`
- `Salary int(10));`

Output:

Not Null Constraints:

NOT NULL on CREATE a Table

Field	Type	Null	Key	Default	Extra
EmpID	int	NO	PRI	NULL	
Name	varchar(50)		YES		NULL
Country	varchar(50)		YES		NULL
Age	int	YES		NULL	
Salary	int	YES		NULL	

NOT NULL on ALTER Table

- We can also add a NOT NULL constraint in the existing table using the ALTER statement.
- For example, if the EMPLOYEES table has already been created then add NOT NULL constraints to the “Name” column using ALTER statements in SQL as follows:

Query:

```
ALTER TABLE Emp modify Name Varchar(50) NOT NULL;
```


Check Constraints:

CHECK Constraint

- The CHECK constraint is used to limit the value range that can be placed in a column.
- If you define a CHECK constraint on a column it will allow only certain values for this column.
- If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

CHECK on CREATE TABLE

- The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created.
- The CHECK constraint ensures that the age of a person must be 18, or older:

Check Constraints:

MySQL:

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    CHECK (Age>=18)  
);
```

Check Constraints:

CHECK on ALTER TABLE

- To create a CHECK constraint on the "Age" column when the table is already created, use the following SQL:
- MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons  
ADD CHECK (Age>=18);
```

- To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:
- MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons  
ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND  
City='Sandnes');
```

Check Constraints:

DROP a CHECK Constraint

To drop a **CHECK** constraint, use the following SQL:

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons  
DROP CONSTRAINT CHK_PersonAge;
```

MySQL:

```
ALTER TABLE Persons  
DROP CHECK CHK_PersonAge;
```

Relational Algebra Operations:

- 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:

- We use the term **relation instance** to refer to a specific instance of a relation, i.e., containing a specific set of rows.
- Relational Algebra is procedural query language, which takes Relation as input and generate relation as output.
- Relational algebra mainly provides theoretical foundation for relational databases and SQL.

Relational Algebra Operations:

Basic Relational Algebra Operations

- **Unary Relational Operations**
- SELECT (symbol: σ)
- PROJECT (symbol: π)
- RENAME (symbol: ρ)
- **Relational Algebra Operations From Set Theory**
- UNION (\cup)
- INTERSECTION (\cap)
- DIFFERENCE ($-$)
- CARTESIAN PRODUCT (\times)
- **Binary Relational Operations**
- JOIN
- DIVISION

Relational Algebra Operations:

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.

$\sigma p(r)$

- σ - is the predicate
- r - stands for relation which is the name of the table
- p - is propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like $=, \neq, \geq, <, >, \leq$.

Relational Algebra Operations:

SELECT (σ)

- 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

Relational Algebra Operations:

$\sigma_{\text{subject} = \text{"database"}}(\text{Books})$

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

$\sigma_{\text{subject} = \text{"database"} \text{ and } \text{price} = \text{"450"}}(\text{Books})$

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

$\sigma_{\text{subject} = \text{"database"} \text{ and } \text{price} = \text{"450"} \text{ or } \text{year} > \text{"2010"}}(\text{Books})$

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

those books published after 2010.

Relational Algebra Operations:

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. (**π**)
- 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.



Relational Algebra Operations:

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

- Here, the projection of CustomerName and status will give

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

CustomerName	Status
Google	Active
Amazon	Active
Apple	Inactive
Alibaba	Active

Relational Algebra Operations:

- **CARTESIAN PRODUCT** on two relations that is on two sets of tuples, it will take every tuple one by one from the left set(relation) and will pair it up with all the tuples in the right set(relation).
- So, the CROSS PRODUCT of two relation $A(R_1, R_2, R_3, \dots, R_p)$ with degree p , and $B(S_1, S_2, S_3, \dots, S_n)$ with degree n , is a relation $C(R_1, R_2, R_3, \dots, R_p, S_1, S_2, S_3, \dots, S_n)$ with degree $p + n$ attributes.
- CROSS PRODUCT is a binary set operation means, at a time we can apply the operation on two relations.
- But the two relations on which we are performing the operations do not have the same type of tuples, which means Union compatibility (or Type compatibility) of the two relations is not necessary.

Relational Algebra Operations:

Notation:

$A \times S$

where A and S are the relations,

the symbol ' \times ' is used to denote the CROSS PRODUCT operator.

Example:

Consider two relations STUDENT(SNO, FNAME, LNAME) and DETAIL(ROLLNO, AGE) below:

On applying CROSS PRODUCT on STUDENT and DETAIL:

STUDENT \times DETAILS

SNO	FNAME	LNAME
1	Albert	Singh
2	Nora	Fatehi

ROLLNO	AGE
5	18
9	21

SNO	FNAME	LNAME	ROLLNO	AGE
1	Albert	Singh	5	18
1	Albert	Singh	9	21
2	Nora	Fatehi	5	18
2	Nora	Fatehi	9	21

Relational Algebra Operations:

RENAME OPERATION

- The rename operation is denoted by ρ .
- The rename operation allows to rename a relation. The syntax of rename operation is- $\rho\langle\text{New_Relation_Name}\rangle (\text{Existing_Relation_name})$

For example,

$\rho\langle\text{Employee_Under_Project}\rangle (\text{Employee})$

- Renames the relation Employee with Employee_Under_Project.
- The Rename operator performs operation on only relation. So, it is a unary operator

Relational Algebra Operations:

The Rename Operation

- The SQL allows renaming relations and attributes using the **as** clause:
old-name as new-name
- Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.
 - **select distinct** *T.name*
from *instructor as T, instructor as S*
where *T.salary > S.salary and S.dept_name = 'Comp. Sci.'*
- Keyword **as** is optional and may be omitted
instructor as T \equiv *instructor T*

Relational Algebra Operations:

Join Operations

- Join operation is essentially a cartesian product followed by a selection criterion.
- Join operation denoted by \bowtie .
- 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

Relational Algebra Operations:

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:

Relational Algebra Operations:

NATURAL JOIN (\bowtie)

- 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

- $C \bowtie D$

C \bowtie D		
Num	Square	Cube
2	4	8
3	9	18

Relational Algebra Operations:

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 join, 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.



Relational Algebra Operations:

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

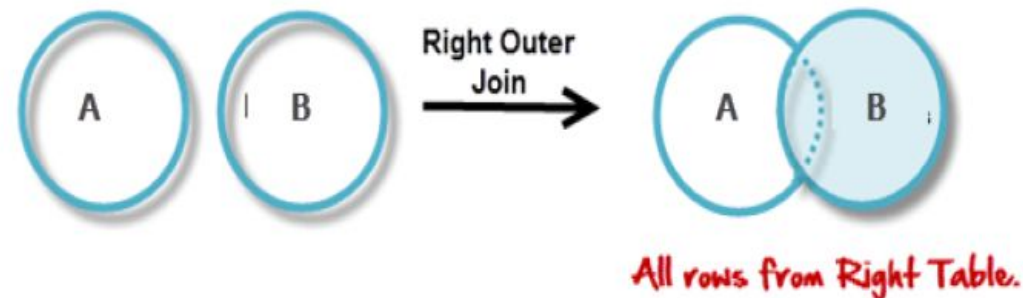
Right Table (HoD)	
C	D
100	Alex
102	Maya
104	Mira

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

Relational Algebra Operations:

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.



Relational Algebra Operations:

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

Relational Algebra Operations:

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.

• $A \bowtie B$

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

Relational Algebra Operations:

Set Operators

Union operation (\cup)

- UNION is symbolized by \cup 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 $\leftarrow A \cup B$

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

Relational Algebra Operations:

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

A \cup B gives:

Table A \cup B	
Column 1	Column 2
1	1
1	2
1	3

Relational Algebra Operations:

Set Operators

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: **$A - B$**

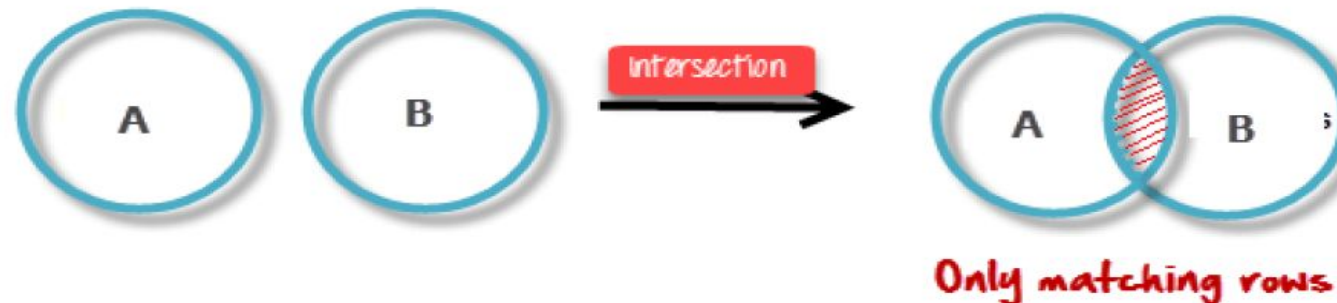
Table A – B	
Column 1	Column 2
1	2

Relational Algebra Operations:

Set Operators

Intersection

- An intersection is defined by the symbol \cap
 $A \cap B$
- 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.





Relational Algebra Operations:

Aggregate Functions

Aggregate Functions

- These functions operate on the multiset of values of a column of a relation, and return a value

avg: average value

min: minimum value

max: maximum value

sum: sum of values

count: number of values

Relational Algebra Operations:

Aggregate Functions

Aggregate Functions (Cont.)

- Find the average salary of instructors in the Computer Science department
 - **select avg (salary)**
from instructor
where dept_name= 'Comp. Sci.';
- Find the total number of instructors who teach a course in the Spring 2010 semester
 - **select count (distinct ID)**
from teaches
where semester = 'Spring' and year = 2010;
- Find the number of tuples in the *course* relation
 - **select count (*)**
from course;

Aggregate Functions

- Find the average salary of instructors in each department

- **select dept_name, avg (salary) as avg_salary**
from instructor
group by dept_name;

<i>dept_name</i>	<i>avg_salary</i>
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Relational Algebra Operations:

Aggregate Functions

Aggregate Functions – Having Clause

- Find the names and average salaries of all departments whose average salary is greater than 42000

```
select dept_name, avg (salary)
from instructor
group by dept_name
having avg (salary) > 42000;
```

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

Relational Algebra Operations:

Aggregate Functions

Null Values and Aggregates

- Total all salaries

```
select sum (salary )  
from instructor
```

- Above statement ignores null amounts
- Result is *null* if there is no non-null amount
- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes
- What if collection has only null values?
 - count returns 0
 - all other aggregates return null

× DIGITAL LEARNING CONTENT



Parul[®] University



www.paruluniversity.ac.in