

Database Management System

IT 261

Bachelor of Information Technology

Prepared by Dr. Khaled Eskaf

Lecture 4





Course Outlines

Week	Topic	Objective	MySQL Practical Activities
1	Introduction to Database Concepts	Understand basic database concepts.	Install MySQL, explore the interface, create a basic database.
2	Relational Data Model	Learn about the relational data model.	Create relational tables in MySQL.
3	Relational Integrity Rules	Understand relational integrity constraints.	Implement primary and foreign key constraints in MySQL.
4	Relational Algebra and Basic SQL	Explore relational algebra and basic SQL commands.	Use basic SQL queries in MySQL.
5	Advanced SQL Queries	Delve into complex SQL operations.	Execute advanced SQL commands in MySQL.
6	Introduction to Database Normalization	Begin learning database normalization principles.	Apply the first normal form (1NF) in MySQL.
7	Advanced Database Normalization	Explore advanced normalization techniques.	Normalize a database in MySQL to 2NF and 3NF.
8	Data Modeling and ER Diagrams	Understand data modeling and ER diagrams.	Create an ER diagram and implement it in MySQL.
9	Database Design for Real-World Apps	Apply database design to real-world scenarios.	Develop a database schema based on a real-world example.
10	Project and Review	Apply all learned concepts in a comprehensive project.	Complete a project in MySQL incorporating all learned skills.

Revision

Week 3 Outlines

2	Data Model	model.	Create relational tables in MySQL.
3	Relational Integrity Rules	Understand relational integrity constraints.	Implement primary and foreign key constraints in MySQL.
	Relational		

Relational integrity rules are fundamental in relational database design, ensuring data remains accurate and consistent. They include primary keys to uniquely identify records, foreign keys to maintain referential integrity (ensuring relationships between tables are preserved), and unique constraints to prevent duplicate entries. These rules help in **maintaining the reliability of data relationships across the database.**

Relational integrity rules include:

- **Primary Keys:** Uniquely identify each record in a table.
- **Foreign Keys:** Establish and enforce relationships between tables.
- **Unique Constraints:** Prevent duplicate entries in specific columns.
- **Check Constraints:** Ensure that column values meet a specified condition.
- **Not Null Constraints:** Ensure that a column cannot have a null value.

These rules ensure data accuracy and consistency within relational databases.

Table named **Students**, where each student has a unique student ID, a name, and an age. The primary key for this table would be the student ID because it uniquely identifies each record. Here's a basic example:

•**Table: Students**

•**Columns:** StudentID (Primary Key), Name, Age

•**Rows:**

- Stu_Id: 1, Stu_Name: ali ahmed, Stu_Age:20
- Stu_Id : 2, Stu_Name: hassan ahmed, Stu_Age:21

<u>Stu_Id</u>	Stu_Name	Stu_Age
101	Steve	23
102	John	24
103	Robert	28
104	Steve	29
105	Carl	29

Primary Key

Unique values

Integrity Constraints: Entity Integrity

Definition: A primary key attribute can NOT be null (i.e. MUST have data)

SALESPERSON

salesperson_id	first_name	last_name	salary	territory_id
100	John	Stewart	48,000	1
140	Allen	Beeton	52,000	2
110	Chris	Lucero	43,000	1
190	Lorenzo	Davis	55,000	OK
150	Susan	Martin	50,000	3

SALESPERSON

salesperson_id	first_name	last_name	salary	territory_id
100	John	Stewart	48,000	1
X	Allen	Beeton	52,000	2
110	Chris	Lucero	43,000	1
190	Lorenzo	Davis	55,000	2
150	Susan	Martin	50,000	3

Note: Foreign keys can only be NULL with an optional one on the parent side of the relationship

Foreign keys establish a relationship between two tables in a database, ensuring referential integrity. For example, consider two tables: **Students** and **Courses**. Each student enrolls in a course, creating a relationship between students and courses.

•**Students Table:**

•Columns: **StudentID (Primary Key)**, StudentName

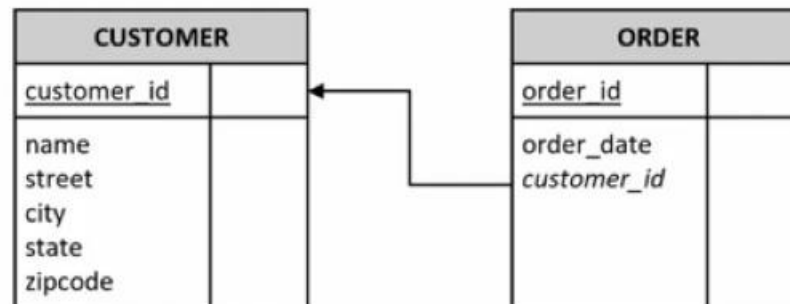
•**Courses Table:**

•Columns: CourseID (Primary Key), CourseName, **StudentID (Foreign Key)**

The **StudentID** in the **Courses** table is a foreign key that references the **StudentID** in the **Students** table. This setup ensures that each course entry is associated with a valid student, maintaining the integrity of the database.

Integrity Constraints: Referential Integrity

Definition: Any foreign key value **MUST** match a primary key value in the relation of the one side (if optional the foreign key can be null)



Unique constraints ensure that all values in a **column are different from each other**, preventing duplicate entries in a database table. For instance, in a table named **Students**, a unique constraint on the **email** column guarantees that no two students can have the same email address. Here's a simplified example of how this might be applied:

- **Table Name:** Students
- **Columns:** StudentID, Name, Email
- **Unique Constraint:** Applied to the **Email** column.

By setting a unique constraint on the **Email** column, the database ensures that every student has a distinct email address, thus maintaining the integrity and uniqueness of student contact information.

Chapter

03

This is illegal because both the Name and Email columns contain the same values for two rows.

Name	Email	Age
Amy Smith	amy.s@email.com	24
Jim Anderson	jim.a@email.com	21
Jim Anderson	jim.a@email.com	32

This is correct.

Name	Email	Age
Amy Smith	amy.s@email.com	24
Jim Anderson	jim.a@email.com	21
Jim Anderson	jim@email.com	32

Check constraints in a database ensure that all values in a column meet a specific condition. For example, if you have a table named "Employees" and you want to make sure that the age of each employee is greater than 18, you would add a check constraint on the "Age" column.

Example Table: Employees

- Columns: EmployeeID, Name, Age

```
CREATE TABLE tablename(  
    Column1 datatype,  
    Column2 datatype'  
    CHECK (column1 > 0)  
);
```

Not Null Constraints Example:

Imagine we have a table named **Students**, where each student must have a unique **student_id** and a name. To ensure every student record has a name, we apply a Not Null constraint on the **name** column.

- **Table Name:** Students

- **Columns:** student_id (INT), name (VARCHAR)

```
SELECT * FROM ConstraintDemo1
```

100 %

Results Messages

	ID	Name
1	1	Ali
2	2	NULL

A domain constraint

A domain constraint specifies the permissible values for a given attribute in a database table. For example, consider a table named "Students" with an attribute "Grade". If the "Grade" attribute only accepts values from A to F, then the domain constraint for "Grade" would be set to only these letters. This ensures data integrity by restricting the grade values entered into the database to only those that are valid within the specified domain.

Integrity Constraints: Domain Constraint

Definition: All allowable values for an attribute

Table	Attribute	Description	Domain
CUSTOMER	customer_id	Set of all possible customer IDs	integer: size 11
CUSTOMER	name	Set of all possible customer names	varchar: size 255
CUSTOMER	state	Set of all possible states	char: size 2
ORDER	order_date	Set of all possible order dates	datetime: format yyyy-mm-dd hh:mm:ss
ORDER_LINE	quantity	Set of all possible quantities	integer: size 8
PRODUCT	price	Set of all possible prices	double: size 10, 2 decimals
...	...	...	...



Do the following for the relations below:

1. Identify the attributes that would enforce entity integrity constraints.
2. Identify the referential integrity constraints that must exist.
3. Identify the domain constraints for each attribute of each relation.

DEPARTMENT	
<u>department_id</u>	
name	

EMPLOYEE	
<u>employee_id</u>	
first_name	
last_name	
salary	
department_id	

CERTIFICATE	
<u>employee_id</u>	
<u>course_id</u>	
completed	

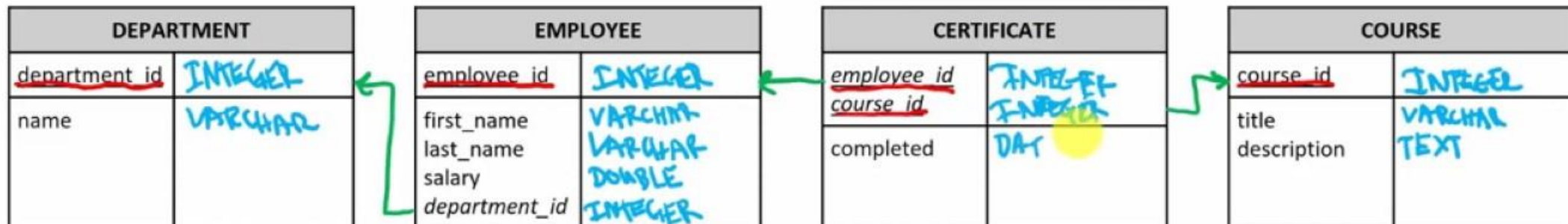
COURSE	
<u>course_id</u>	
title	
description	

Chapter

03

Do the following for the relations below:

1. Identify the attributes that would enforce entity integrity constraints.
2. Identify the referential integrity constraints that must exist.
3. Identify the domain constraints for each attribute of each relation.





To create a database in MySQL Workbench for students, courses, and teachers, follow these steps:

1.Open MySQL Workbench and connect to your MySQL server.

2.Create a New Schema: Click on the "+" icon next to "Schemas" to create a new database. Name it **school_db** and apply the changes.

3.Create Tables: Use the toolbar to create new tables for **Students**, **Courses**, and **Teachers**.

- For **Students**, add fields like **student_id**, **name**, **email**.
- For **Courses**, include **course_id**, **course_name**, **teacher_id**.
- For **Teachers**, fields might be **teacher_id**, **name**.

4.Define Primary Keys for each table (e.g., **student_id**, **course_id**, **teacher_id**) and set them to auto-increment.

5.Establish Relationships: Use foreign keys to link **Courses** to **Teachers** by **teacher_id**.

6.Insert Data: Use **INSERT INTO** statements to populate each table with sample data.

1.Create a Database (Schema):

```
CREATE DATABASE school_db; USE school_db;
```

2.Create a Table for Students:

```
CREATE TABLE Students ( student_id INT AUTO_INCREMENT, name VARCHAR(100), email VARCHAR(100), PRIMARY KEY(student_id) );
```

3. Create a Table for Courses:

```
CREATE TABLE Courses ( course_id INT AUTO_INCREMENT, course_name VARCHAR(100), teacher_id INT, PRIMARY KEY(course_id), FOREIGN KEY(teacher_id) REFERENCES Teachers(teacher_id) );
```

4.Create a Table for Teachers:

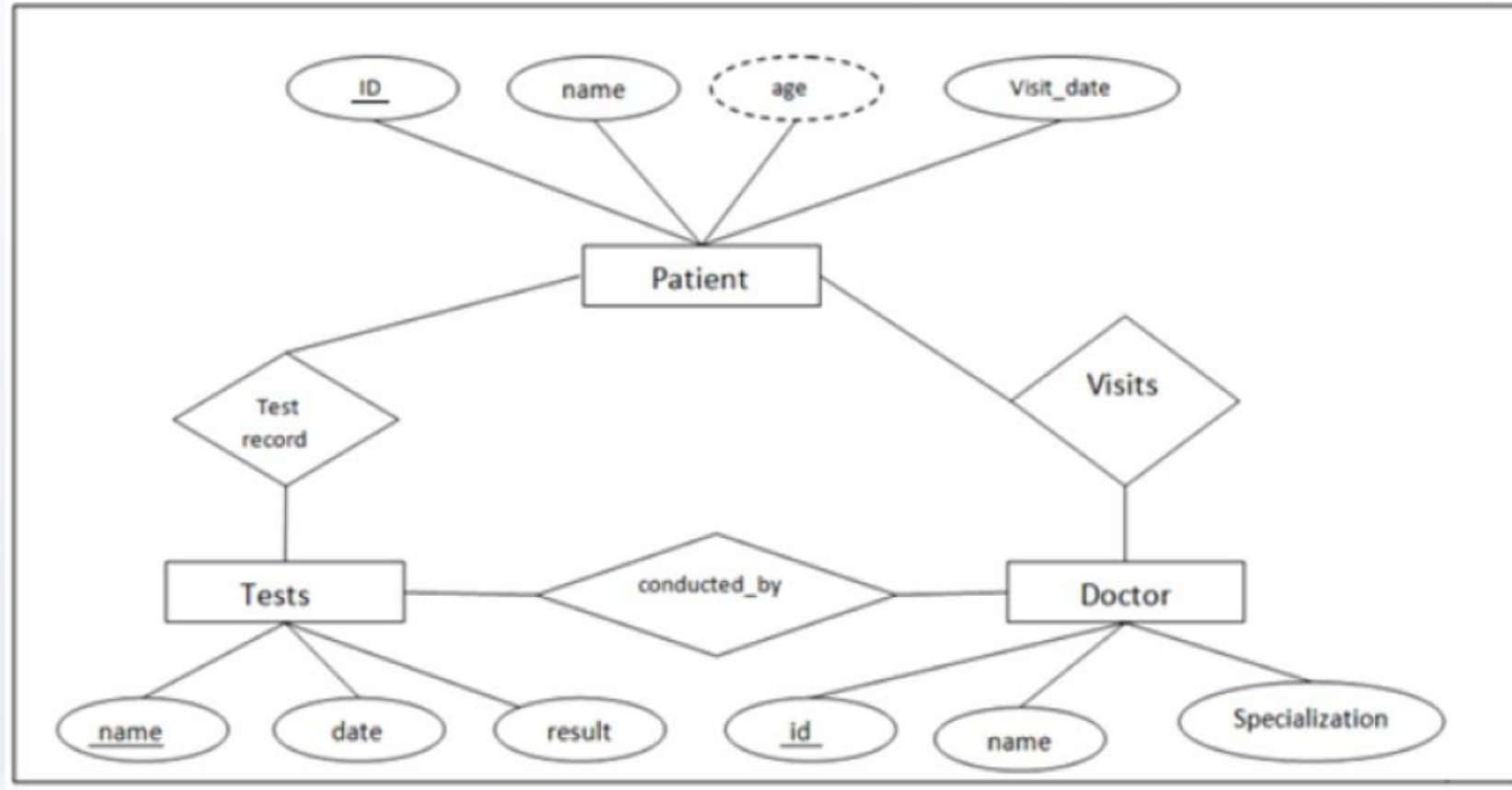
```
CREATE TABLE Teachers ( teacher_id INT AUTO_INCREMENT, name VARCHAR(100), PRIMARY KEY(teacher_id) );
```

Hospital ER Model

Hospital.

The entities are represented in rectangular boxes and are Patient, Tests and Doctor. Each of these entities have their respective attributes which are –

- **Patients** - ID(primary key), name, age, visit_date
- **Tests**- Name(primary key), date, result
- **Doctor**- ID(primary key), name, specialization



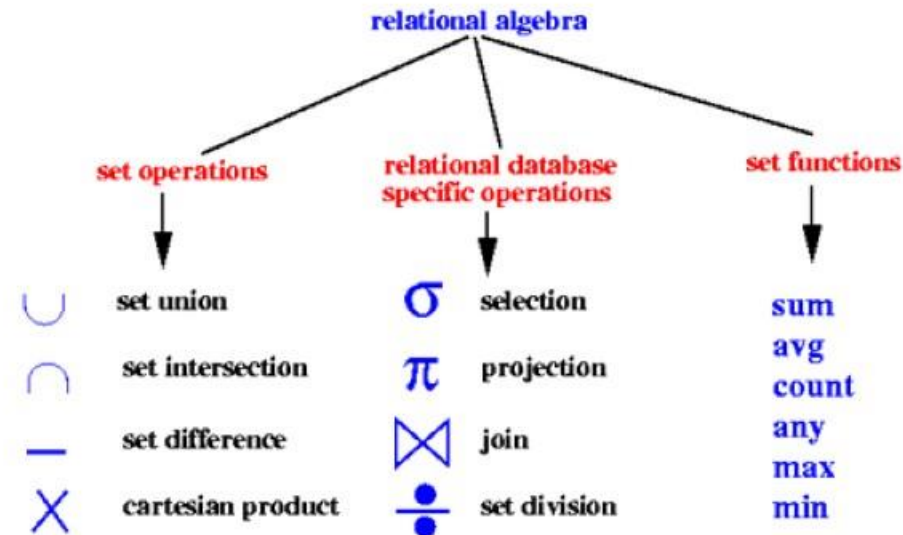
3	Rules and constraints.	constraints in MySQL.	
4	Relational Algebra and Basic SQL	Explore relational algebra and basic SQL commands.	Use basic SQL queries in MySQL.
	Advanced		Execute advanced SOL commands

Member

Member ID	Name	Date of Birth
1	Alice	03/03/1995
2	Bob	11/07/1993
3	Charlie	21/10/1997
4	Mike	16/09/1992
5	Katie	21/10/1997

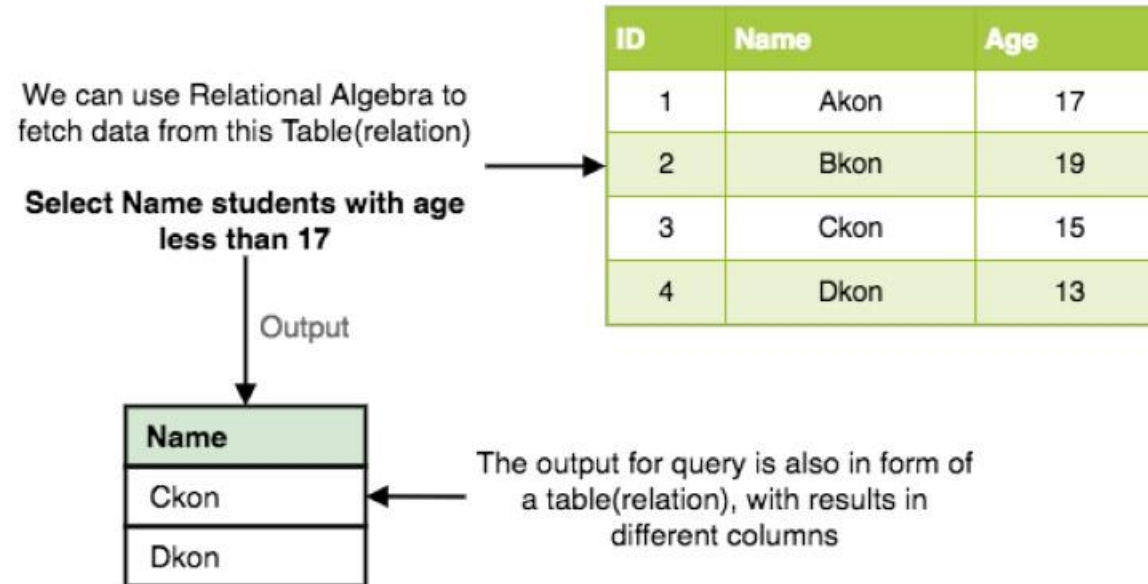
Labels: Relation (red arrow to table), Attribute (green arrow to Date of Birth), Tuple (blue arrow to row 5)

Relational Algebra Operators

 $\pi \sigma \rho$
 $\bowtie U \cap$


Every database management system must define a query language to allow users to access the data stored in the database. **Relational Algebra** is a procedural query language used to query the database tables to access data in different ways.

In relational algebra, input is a relation (table from which data has to be accessed) and output is also a relation (a temporary table holding the data asked for by the user).



Studying relational algebra operators in database management systems (DBMS) is fundamental for several reasons:

•**1- Building Blocks of SQL:**

- SQL is the language we use to talk to databases.
- Relational algebra is like the grammar rules behind SQL.
- Knowing these rules helps us understand how SQL works and why it works that way.

•**2- Making Queries Faster:**

- Relational algebra shows us different ways to ask the same question to a database.
- Some ways are faster than others.
- Learning relational algebra helps us figure out the best way to ask questions so the database can give us answers quickly.

•**3- Clear and Precise Query Descriptions:**

- Relational algebra gives us a clear method to describe what we want from a database.
- This clarity helps avoid mistakes and misunderstandings in what we are asking the database to do.

What is Relational Algebra?

Relational Algebra is a formal system for manipulating relations (tables) in a relational database. It consists of a set of operations that take one or two relations as input and produce a new relation as output. These operations allow you to query and manipulate data stored in a database in a precise and mathematical way.

Member ID	Name	Date of Birth
1	Alice	03/03/1995
2	Bob	11/07/1993
3	Charlie	21/10/1997
4	Mike	16/09/1992
5	Katie	21/10/1997

Relational Algebra works on the whole table at once, so we do not have to use loops etc to iterate over all the rows(tuples) of data one by one. All we have to do is specify the table name from which we need the data, and in a single line of command, relational algebra will traverse the entire given table to fetch data for you.

The primary operations that we can perform using relational algebra are:

- 1.Select
- 2.Project
- 3.Union
- 4.Set Different
- 5.Cartesian product
- 6.Rename

Select Operation (σ) Sigma :

This is used to fetch rows(tuples) from table(relation) which satisfies a given condition.

Syntax: $\sigma_p(r)$

Where, σ represents the Select Predicate, r is the name of relation(table name in which you want to look for data), and p is the propositional logic, where we specify the conditions that must be satisfied by the data. In propositional logic, one can use **unary** and **binary** operators like $=$, $<$, $>$ etc, to specify the conditions.

Let's take an example of the Student table we specified above in the Introduction of relational algebra, and fetch data for **students** with **age** more than 17.

$\sigma_{age > 17}$ (Student)

This will fetch the tuples(rows) from table **Student**, for which **age** will be greater than 17.

You can also use, **and**, **or** etc operators, to specify two conditions, for example,

$\sigma_{age > 17 \text{ and } gender = 'Male'}$ (Student)

This will return tuples(rows) from table **Student** with information of male students, of age more than 17.(Consider the Student table has an attribute **Gender** too.)

A	B	C
1	2	4
2	2	3
3	2	3
4	3	4

Example:

For the above relation, $\sigma(c>3)R$ will select the tuples which have c more than 3.

Note: The selection operator only selects the required tuples but does not display them.
For display, the data projection operator is used.

A	B	C
1	2	4
4	3	4

Project Operation (Π) PI:

Project operation is used to project only a certain set of attributes of a relation. In simple words, If you want to see only the **names** all of the students in the **Student** table, then you can use Project Operation.

It will only project or show the columns or attributes asked for, and will also remove duplicate data from the columns.

Syntax: $\Pi_{A_1, A_2 \dots}(r)$

where A1, A2 etc are attribute names(column names).

For example,

$\Pi_{\text{Name, Age}}(\text{Student})$

Above statement will show us only the **Name** and **Age** columns for all the rows of data in **Student** table.

Projection(π):

It is used to project required column data from a relation.

Example: Consider Table 1. Suppose we want columns B and C from Relation R.

$\pi(B,C)R$ will show following columns.

Note: By Default, projection removes duplicate data.

B	C
2	4
2	3
3	4

A	B	C
1	2	4
2	2	3
3	2	3
4	3	4

Union Operation (U):

This operation is used to fetch data from two relations (tables) or temporary relation (result of another operation). For this operation to work, the relations (tables) specified should have same number of attributes (columns) and same attribute domain. Also the duplicate tuples are automatically eliminated from the result.

Syntax: $A \cup B$

where A and B are relations.

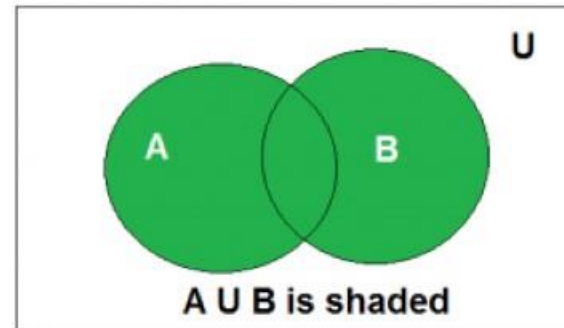
For example, if we have two tables **RegularClass** and **ExtraClass**, both have a column **student** to save name of student, then,

$\Pi_{\text{Student}}(\text{RegularClass}) \cup \Pi_{\text{Student}}(\text{ExtraClass})$

Above operation will give us name of **Students** who are attending both regular classes and extra classes, eliminating repetition.

Union

Union of the sets A and B, denoted by $A \cup B$, is the set of distinct elements that belong to set A or set B, or both.



Venn diagram of $A \cup B$

Above is the Venn Diagram of $A \cup B$.

Example: Find the union of $A = \{2, 3, 4\}$ and $B = \{3, 4, 5\}$; Solution : $A \cup B = \{2, 3, 4, 5\}$.

Union(U): Union operation in relational algebra is the same as union operation in [set theory](#).

Example:

Student_Name	Roll_Number
Ram	01
Mohan	02
Vivek	13
Geeta	17

GERMAN	
Student_Name	Roll_Number
Vivek	13
Geeta	17
Shyam	21
Rohan	25

Consider the following table of Students having different optional subjects in their course.

$\pi(\text{Student_Name})\text{FRENCH} \cup \pi(\text{Student_Name})\text{GERMAN}$

Consider the following table of Students having different optional subjects in their course.

Student_Name
Ram
Mohan
Vivek
Geeta
Shyam
Rohan

FRENCH

Student_Name	Roll_Number
Ram	01
Mohan	02
Vivek	13
Geeta	17

GERMAN

Student_Name	Roll_Number
Vivek	13
Geeta	17
Shyam	21
Rohan	25

$\pi(\text{Student_Name})\text{FRENCH} \cup \pi(\text{Student_Name})\text{GERMAN}$

Note: The only constraint in the union of two relations is that both relations must have the same **set of Attributes**.

Set Difference (-) :

This operation is used to find data present in one relation and not present in the second relation. This operation is also applicable on two relations, just like Union operation.

Syntax: $A - B$

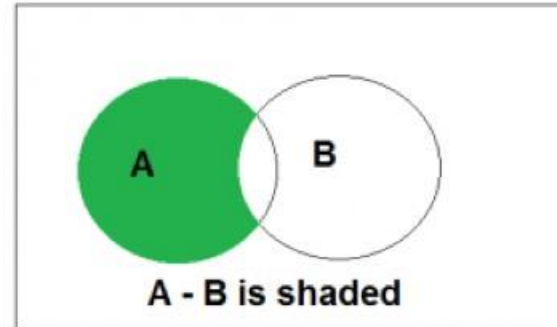
where A and B are relations.

For example, if we want to find name of students who attend the regular class but not the extra class, then, we can use the below operation:

$\Pi_{\text{Student}}(\text{RegularClass}) - \Pi_{\text{Student}}(\text{ExtraClass})$

Set Difference

The difference between sets is denoted by ' $A - B$ ', which is the set containing elements that are in A but not in B. i.e., all elements of A except the element of B.



Example: If $A = \{1, 2, 3, 4, 5\}$ and $B = \{2, 4, 6, 8\}$, find $A - B$ Solution: $A - B = \{1, 3, 5\}$

Set Difference(-): Set Difference in relational algebra is the same set difference operation as in set theory.

Example: From the above table of FRENCH and GERMAN, Set Difference is used as follows

$\pi(\text{Student_Name})\text{FRENCH} - \pi(\text{Student_Name})\text{GERMAN}$

Student_Name
Ram
Mohan

FRENCH	
Student_Name	Roll_Number
Ram	01
Mohan	02
Vivek	13
Geeta	17

GERMAN	
Student_Name	Roll_Number
Vivek	13
Geeta	17
Shyam	21
Rohan	25

Note: The only constraint in the Set Difference between two relations is that both relations must have the same set of Attributes.

Set Intersection($\cap$): Set Intersection in relational algebra is the same set intersection operation in set theory.

Example: From the above table of FRENCH and GERMAN, the Set Intersection is used as follows

$\pi(\text{Student_Name})\text{FRENCH} \cap \pi(\text{Student_Name})\text{GERMAN}$

Student_Name
Vivek
Geeta

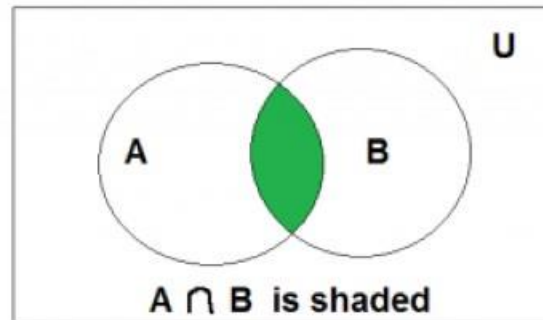
FRENCH	
Student_Name	Roll_Number
Ram	01
Mohan	02
Vivek	13
Geeta	17

GERMAN	
Student_Name	Roll_Number
Vivek	13
Geeta	17
Shyam	21
Rohan	25

Note: The only constraint in the Set Difference between two relations is that both relations must have the same set of Attributes.

Intersection

The intersection of the sets A and B , denoted by $A \cap B$, is the set of elements that belong to both A and B i.e. set of the common elements in A and B .



Example: Find the intersection of $A = \{2, 3, 4\}$ and $B = \{3, 4, 5\}$ Solution : $A \cap B = \{3, 4\}$.

Rename Operation (ρ) rho :

This operation is used to rename the output relation for any query operation which returns result like Select, Project etc. Or to simply rename a relation(table)

Syntax: $\rho(\text{RelationNew}, \text{RelationOld})$

Rename(ρ): Rename is a unary operation used for renaming attributes of a relation.

$\rho(a/b)R$ will rename the attribute 'b' of the relation by 'a'.

Cartesian Product (X) - Cross Product(X):

This is used to combine data from two different relations(tables) into one and fetch data from the combined relation.

Syntax: $A \times B$

For example, if we want to find the information for Regular Class and Extra Class which are conducted during morning, then, we can use the following operation:

$\sigma_{\text{time} = \text{'morning'}} (\text{RegularClass} \times \text{ExtraClass})$

For the above query to work, both **RegularClass** and **ExtraClass** should have the attribute **time**.

Cross Product(X): Cross-product between two relations. Let's say A and B, so the cross product between A X B will result in all the attributes of A followed by each attribute of B. Each record of A will pair with every record of B.

Example:

A

Name	Age	Sex
Ram	14	M
Sona	15	F
Kim	20	M

B

ID	Course
1	DS
2	DBMS

A

Name	Age	Sex
Ram	14	M
Sona	15	F
Kim	20	M

B

ID	Course
1	DS
2	DBMS

A X B

Name	Age	Sex	ID	Course
Ram	14	M	1	DS
Ram	14	M	2	DBMS
Sona	15	F	1	DS
Sona	15	F	2	DBMS
Kim	20	M	1	DS
Kim	20	M	2	DBMS

Note: If A has 'n' tuples and B has 'm' tuples then A X B will have 'n*m' tuples.

Apart from these common operations Relational Algebra is also used for **Join** operations like,

- Natural Join
- Conditional Join



مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY



Thank You

