Relational Database Model

3.1. Structure of RDBMS and Terminology

3.2 Database Schema and Schema Diagram

3.3 Keys: Super, Candidates, primary, foreign Composite etc

3.4. Introduction to relation Algebra

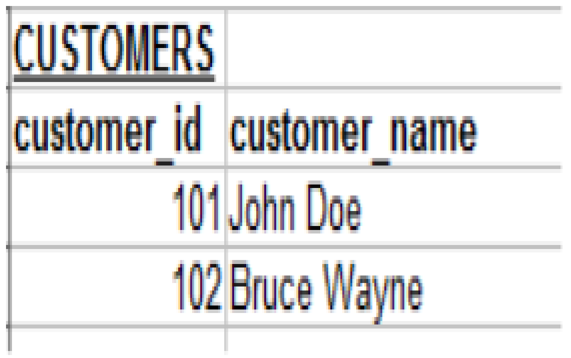
3.5**. Relational Algebra Operations: Select project, Cartesian Product, Union, Set Difference**

3.6. Natural join and outer join.

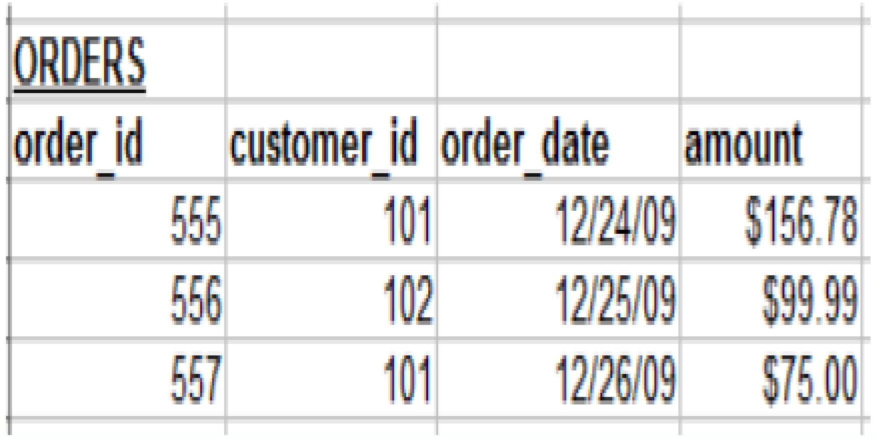
3.1. Structure of RDBMS and Terminology

- A relational database is a type of database. It uses a structure that allows us to identify and access data in relation to another piece of data in the database. Often, data in a relational database is organized into tables
- A relational database management system (RDBMS) is a program that allows you to create, update, and administer a relational database. Most relational database management systems use the SQL language to access the database
- MySQL, SQL-Server, MS-Access, Oracle are some of popular RDBMS.

Examples of RDBMS structure

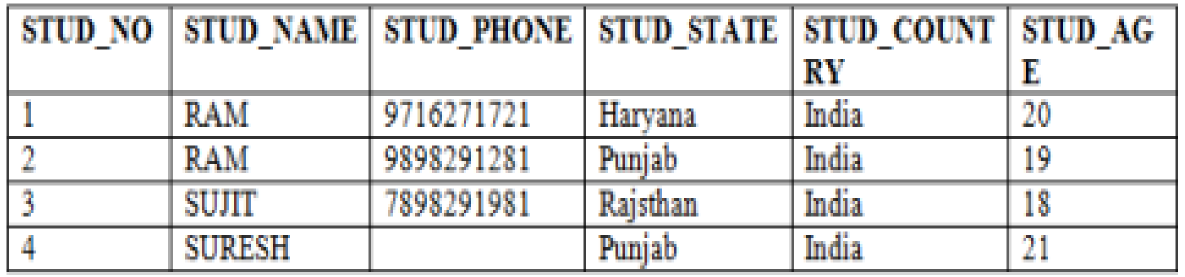


CUSTOMERS	
customer_id	customer_name
101	John Doe
102	Bruce Wayne



ORDERS			
order_id	customer_id	order_date	amount
555	101	12/24/09	$156.78
556	102	12/25/09	$99.99
557	101	12/26/09	$75.00

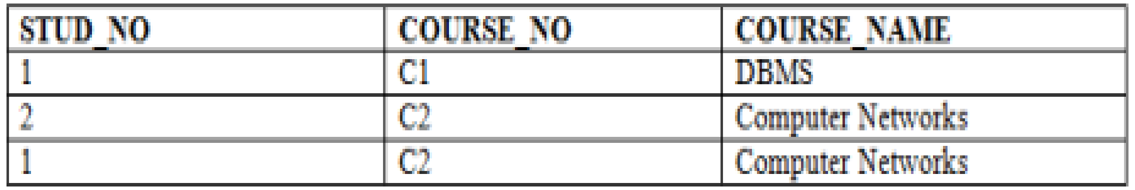
STUDENT



STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

Table 1

STUDENT_COURSE



STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computer Networks
1	C2	Computer Networks

Table 2

3.2 Database Schema and Schema Diagram

- 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.

- A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It's the database designers who design the schema to help programmers understand the database and make it useful.

A database schema can be divided broadly into two categories −

- **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.

Schema Diagram

A database schema, along with primary key and foreign key dependencies, can be depicted by schema diagrams. .

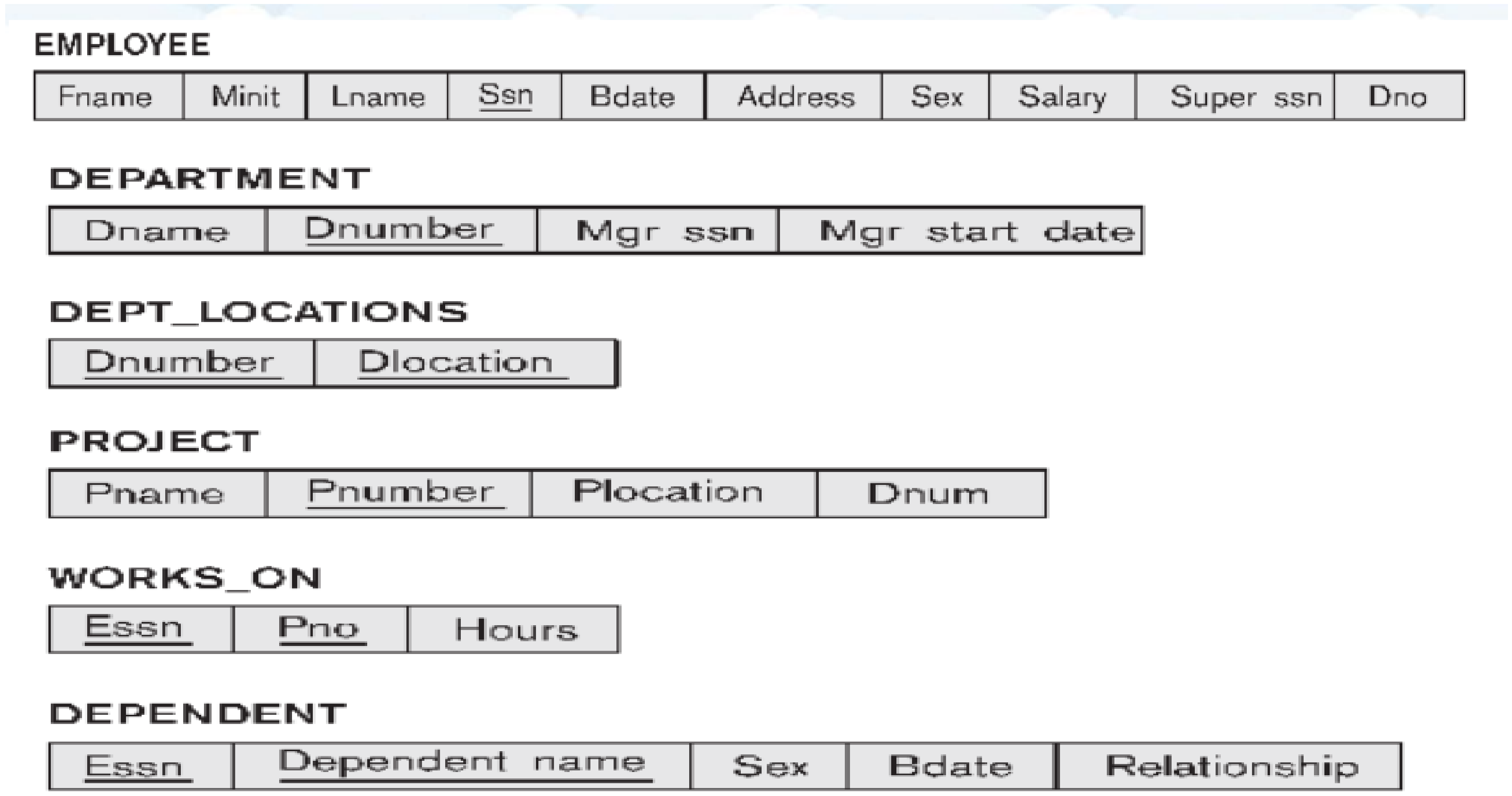
Each relation appears as a box, with the relation name at the top in blue, and the attributes listed inside the box. Primary key attributes are shown underlined.

Foreign key dependencies appear as arrows from the foreign key attributes of the referencing relation to the primary key of the referenced relation

Referential integrity constraints other than foreign key constraints are not shown explicitly in schema diagrams.

Entity relationship diagrams let us represent several kinds of constraints, including general referential integrity constraints.

Schema Diagram for Employee Database



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr ssn	Mgr start date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

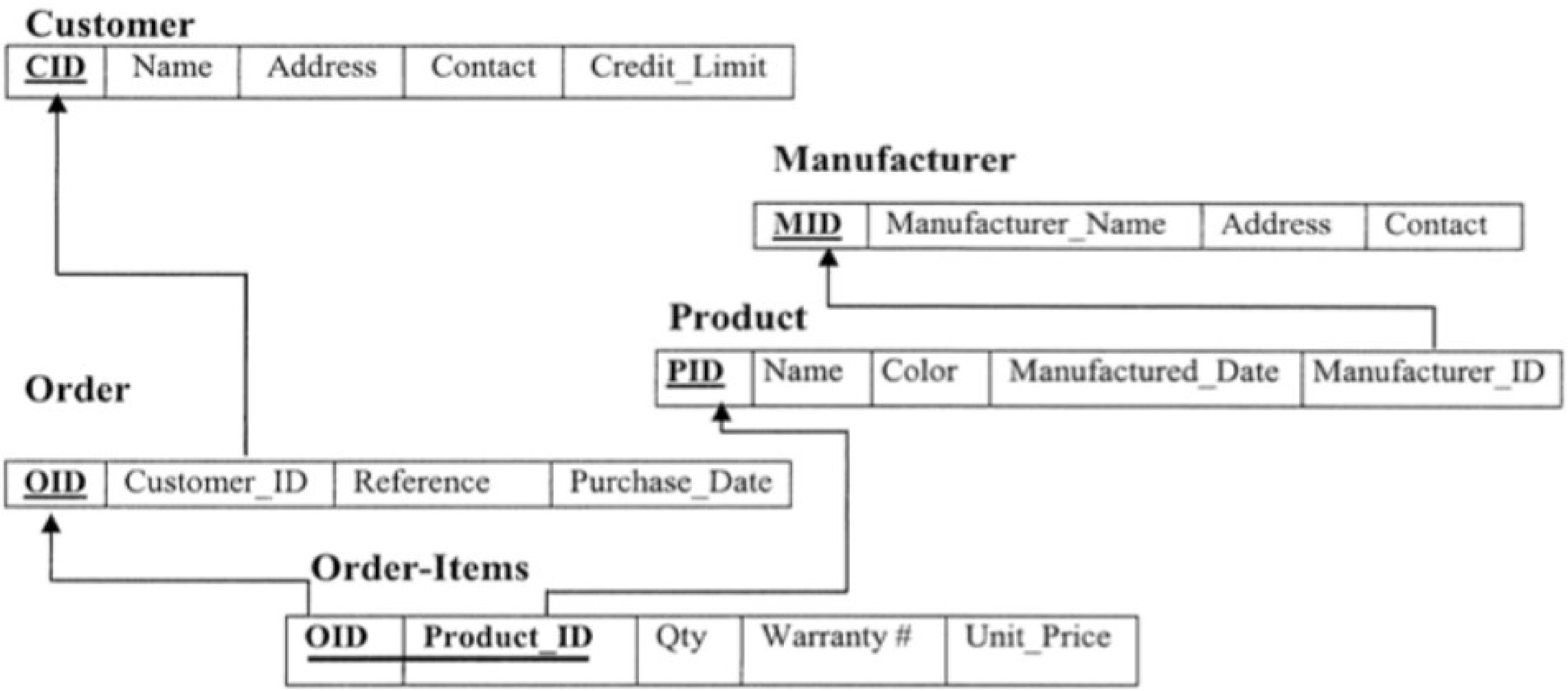
WORKS_ON

Essn	Pno	Hours

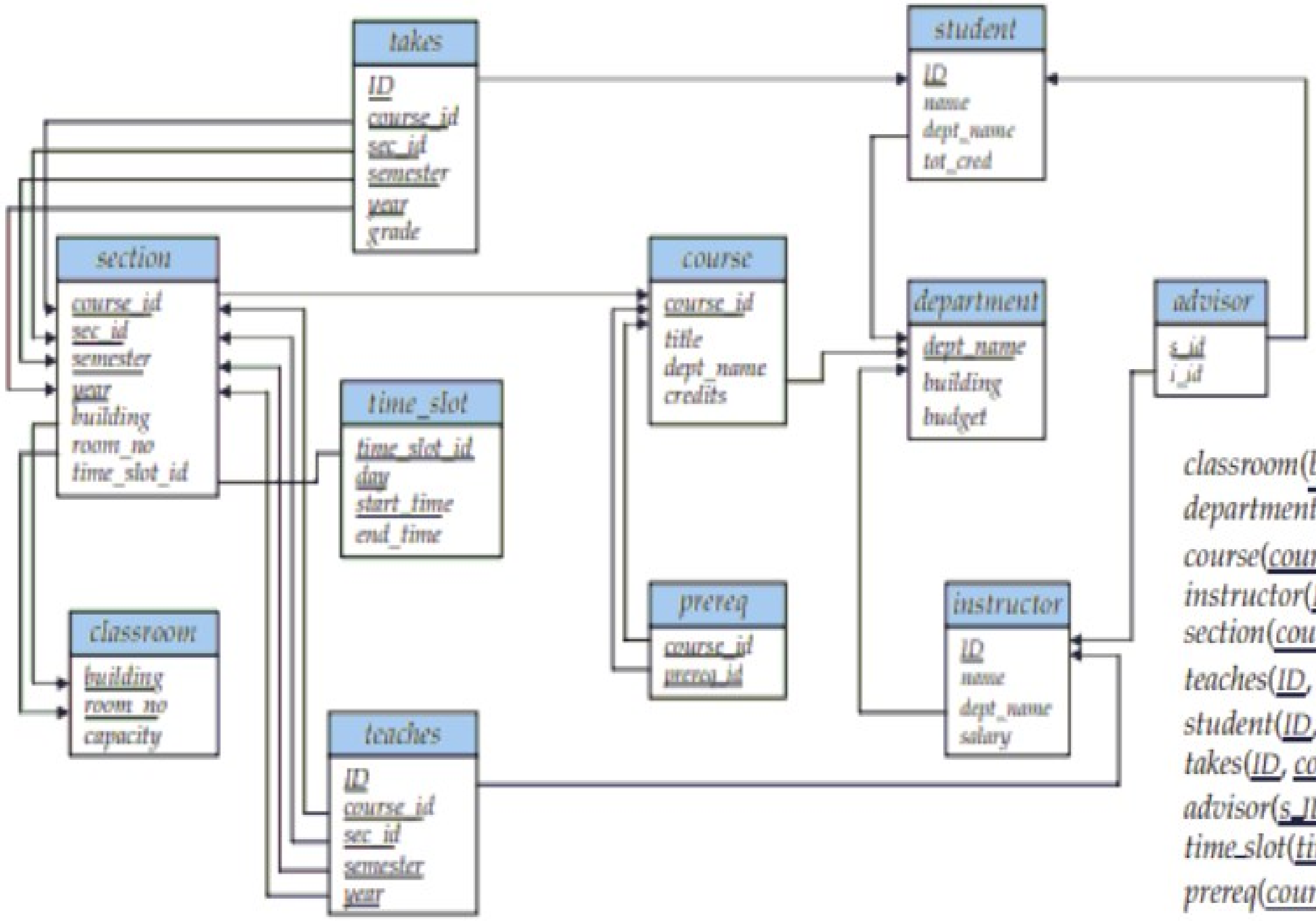
DEPENDENT

Essn	Dependent name	Sex	Bdate	Relationship

Schema Diagram for Order Database



Schema Diagram for University Database



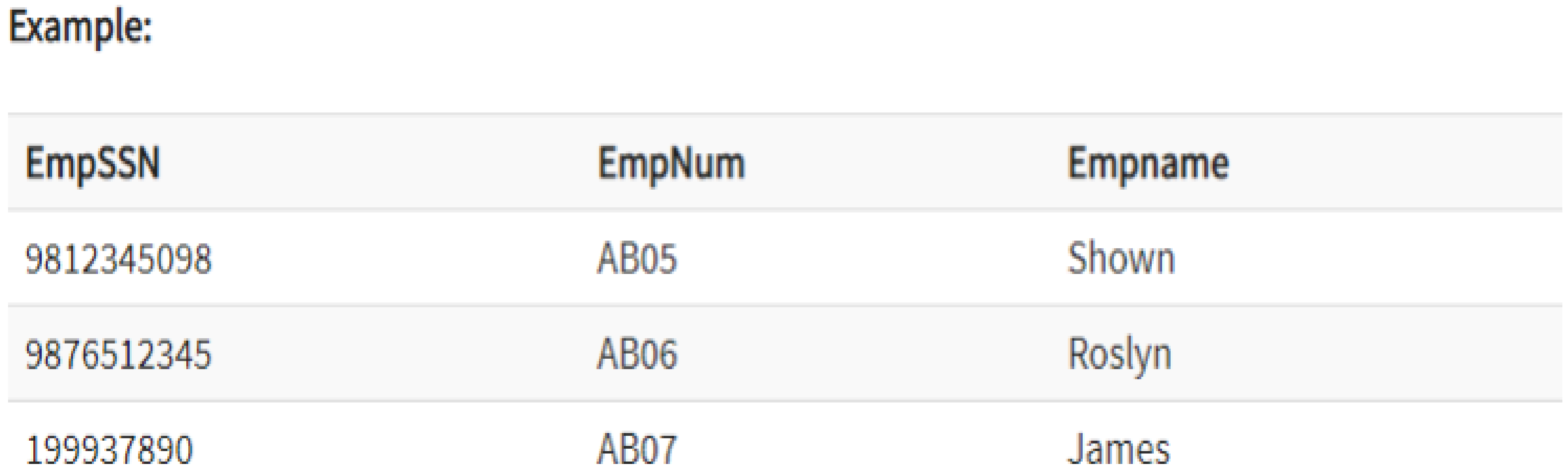
classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building, room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(ID, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)

3.3 Keys: Super, Candidates, primary, foreign Composite etc

A DBMS key is an attribute or set of an attribute 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.

1. Super key :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:

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

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

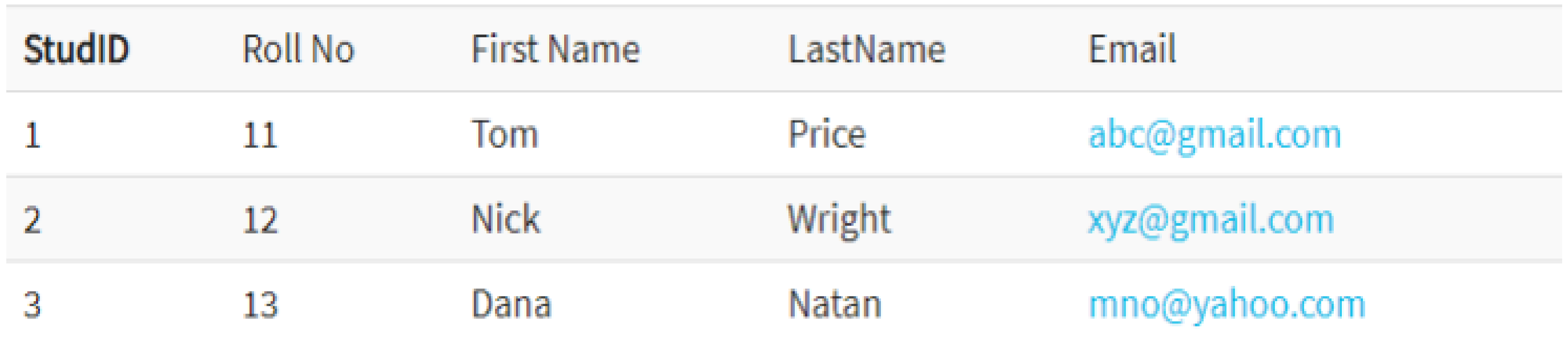
2. Primary Key

A column or group of columns in a table which helps us to uniquely identifies every row in that table is called a primary key.

This DBMS can't be a duplicate. The same value can't appear more than once in the table. 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.



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

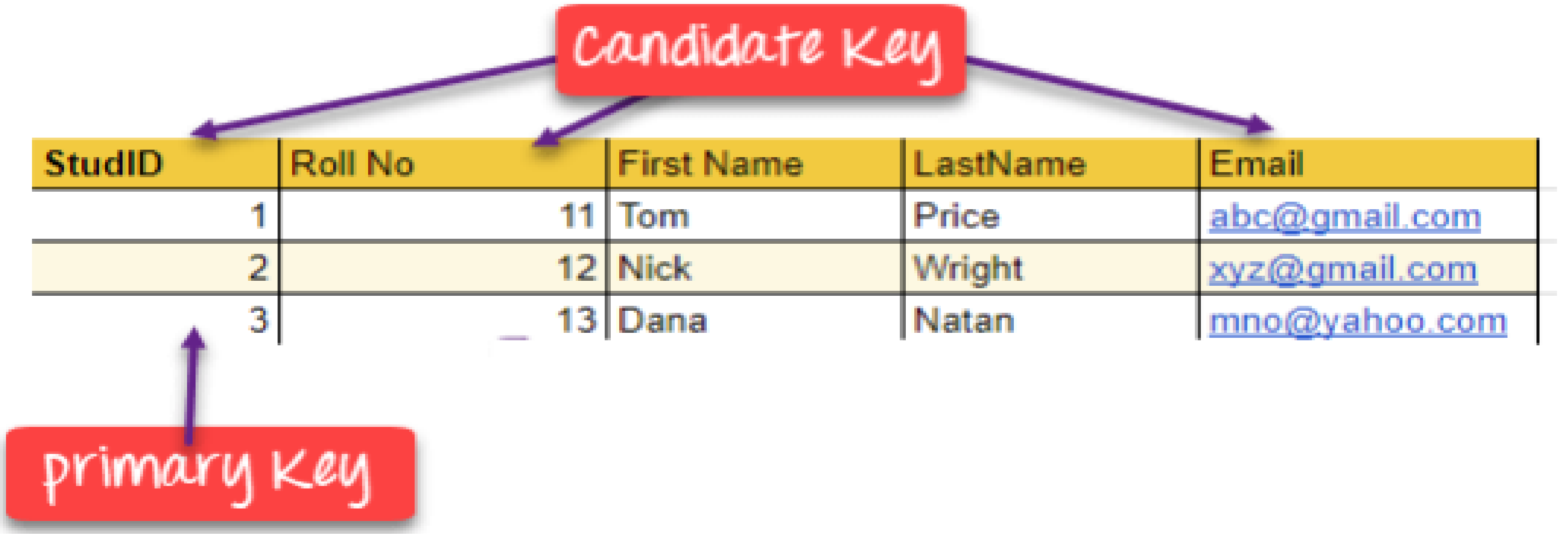
Here, StudId is a Primary Key

3. Candidate Key

- A super key with no repeated attribute is called candidate key.
- The Primary key should be selected from the candidate keys. Every table must have at least a single candidate key. Properties of Candidate key:
- It must contain unique values
- Candidate key may have multiple attributes
- Must not contain null values
- It should contain minimum fields to ensure uniqueness
- Uniquely identify each record in a table.

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.

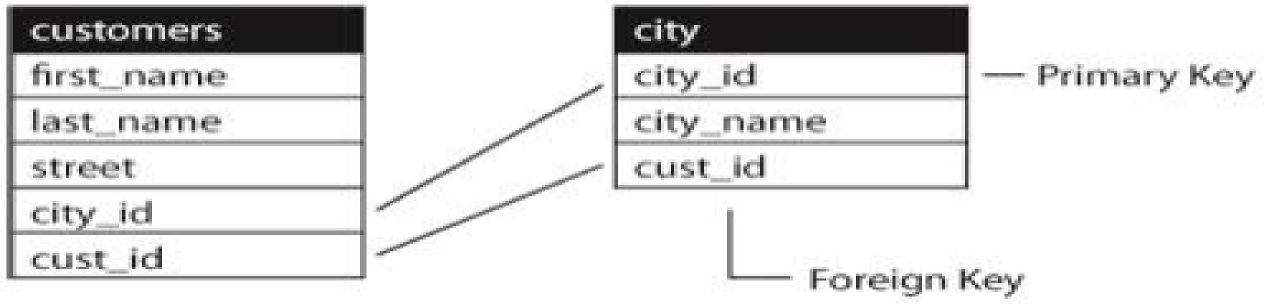
StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com

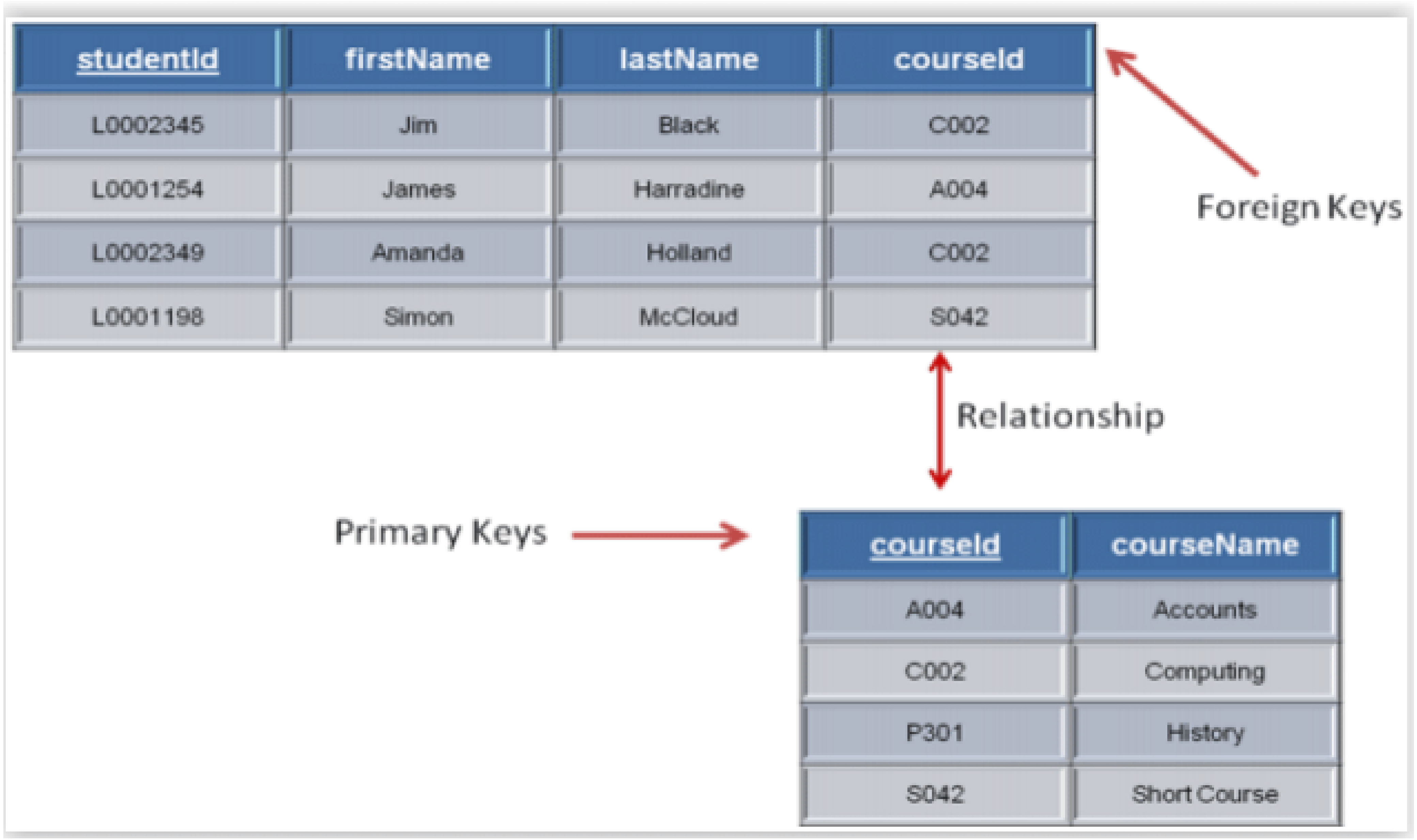


StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com

4. Foreign key

- A foreign key is a column which is added to create a relationship with another table.
- Foreign keys help us to maintain data integrity and also allows navigation between two different instances of an entity.
- Every relationship in the model needs to be supported by a foreign key.



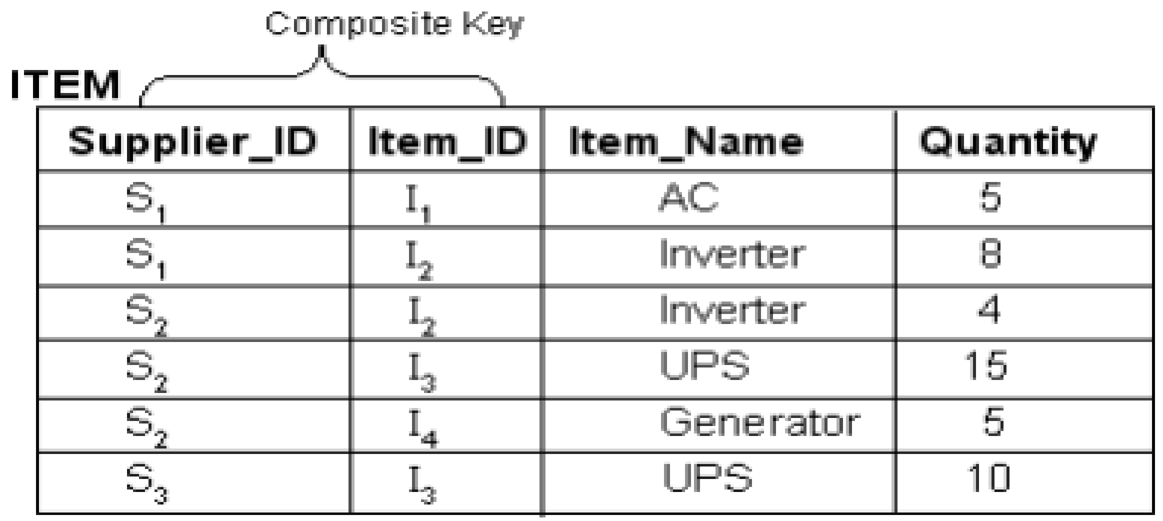


studentId	firstName	lastName	courseId
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042

courseId	courseName
A004	Accounts
C002	Computing
P301	History
S042	Short Course

5. Composite key

A key which has multiple attributes to uniquely identify rows in a table is called a composite key.


Composite Key

ITEM

Supplier_ID	Item_ID	Item_Name	Quantity
S_1	I_1	AC	5
S_1	I_2	Inverter	8
S_2	I_2	Inverter	4
S_2	I_3	UPS	15
S_2	I_4	Generator	5
S_3	I_3	UPS	10

6. Alternate key

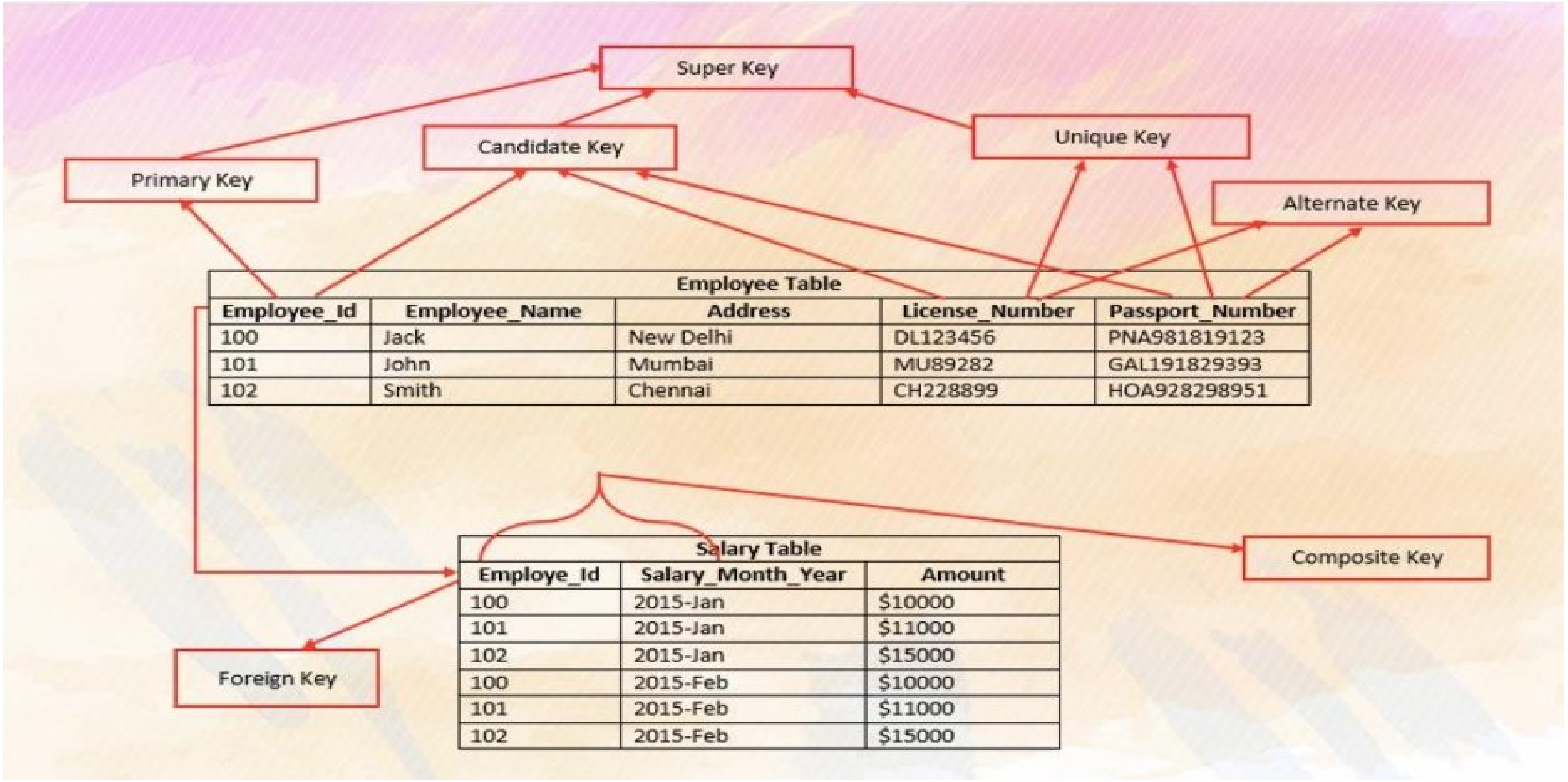
All the keys which are not primary key are called an alternate key. It is a key which is currently not the primary key.

However, A table may have single or multiple choices for the primary key.

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

Summary



Employee Table				
Employee_Id	Employee_Name	Address	License_Number	Passport_Number
100	Jack	New Delhi	DL123456	PNA981819123
101	John	Mumbai	MU89282	GAL191829393
102	Smith	Chennai	CH228899	HOA928298951

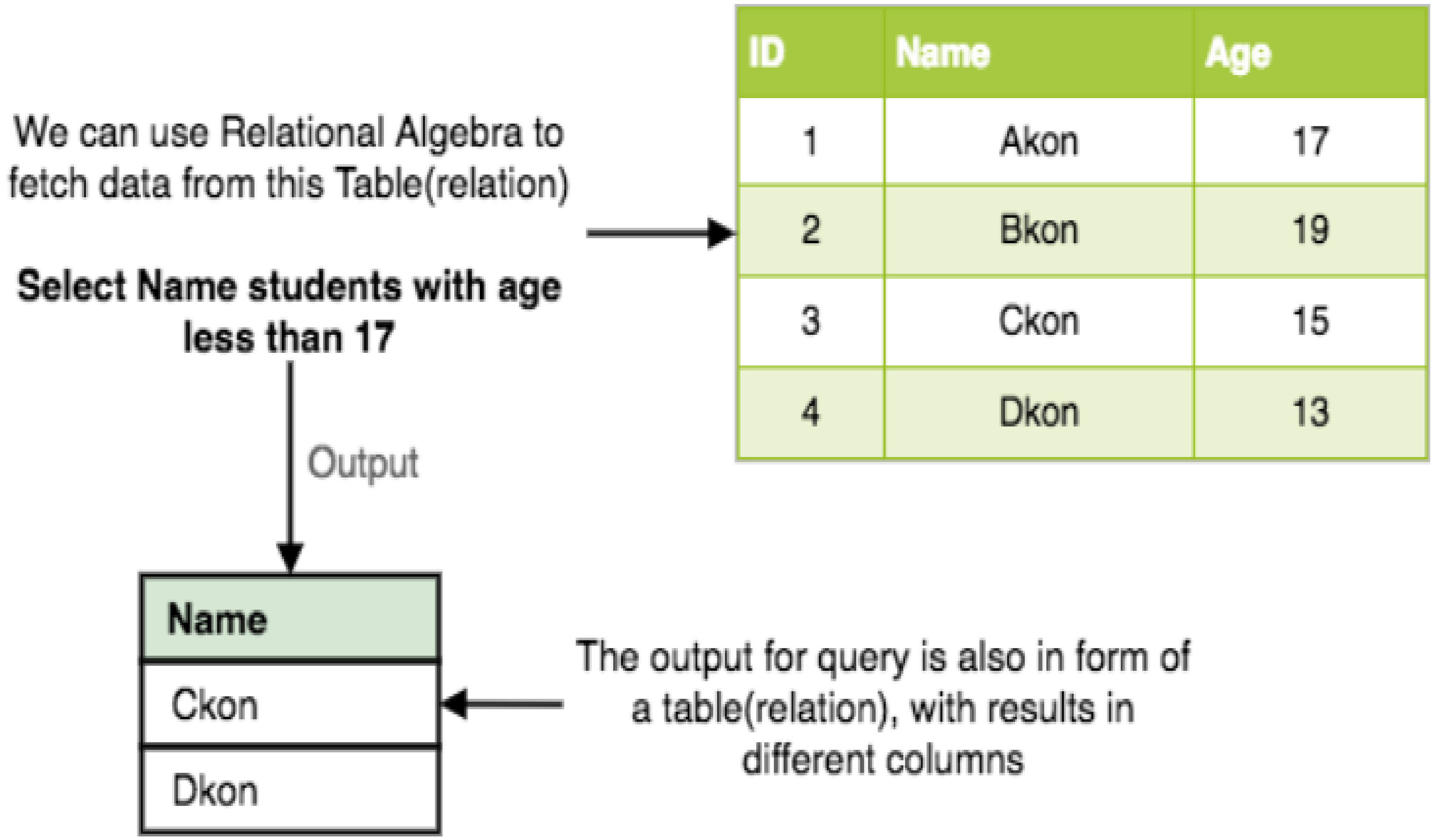
Salary Table		
Employe_Id	Salary_Month_Year	Amount
100	2015-Jan	$10000
101	2015-Jan	$11000
102	2015-Jan	$15000
100	2015-Feb	$10000
101	2015-Feb	$11000
102	2015-Feb	$15000

3.4. Introduction to relation Algebra

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)



We can use Relational Algebra to fetch data from this Table(relation)

ID	Name	Age
1	Akon	17
2	Bkon	19
3	Ckon	15
4	Dkon	13

Select Name students with age less than 17

Name
Ckon
Dkon

The output for query is also in form of a table(relation), with results in different columns

- The primary operations that we can perform using relational algebra are: Select, Project, Union, Set Difference, Cartesian Product and Join.

a) Select Operation (σ)

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

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.

σage > 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 operators, to specify two conditions, for example, **σage > 17 and gender = 'Male' (Student)**

This will return tuples(rows) from table Student with information of male students, of age more than 17.

b) Project Operation (∏)

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.

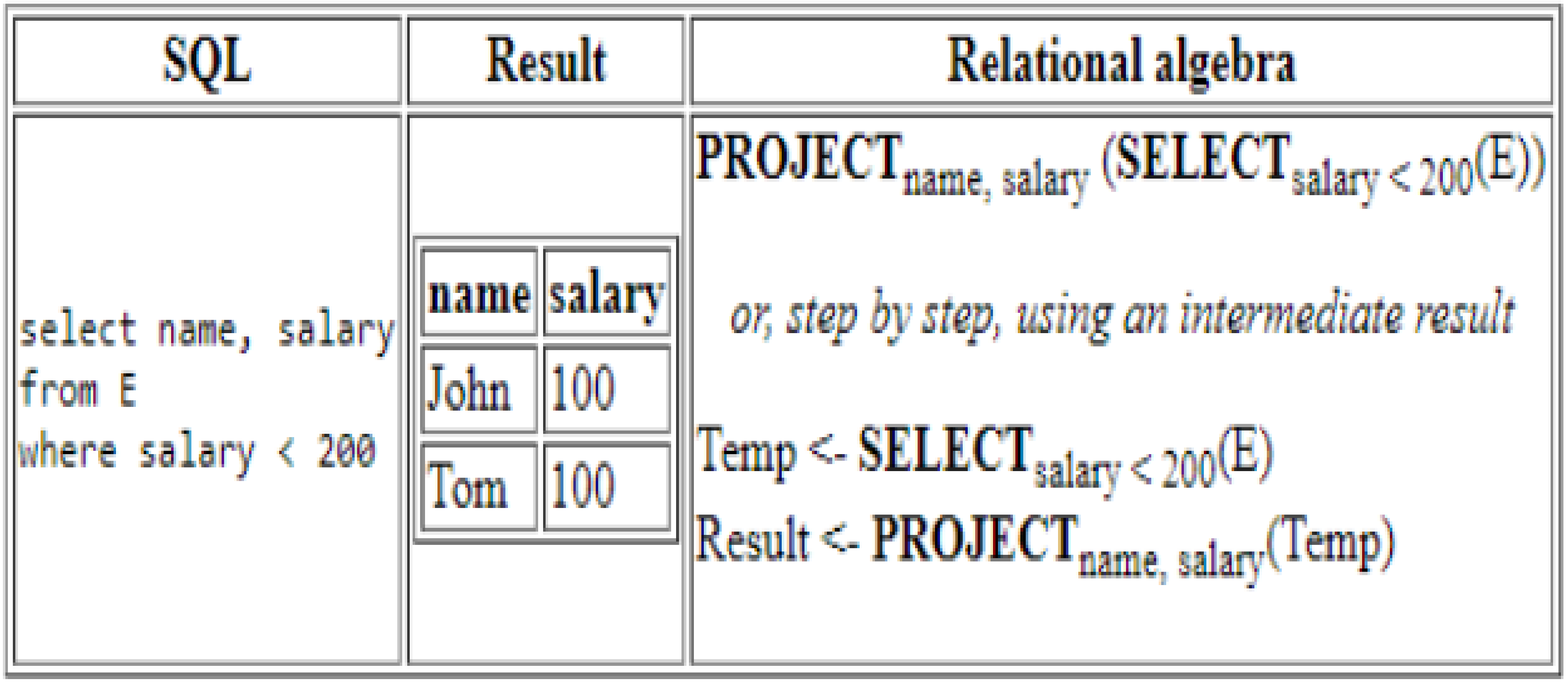
It will only project or show the columns or attributes asked for, and will also remove duplicate data from the columns.

For example,

∏Name, Age(Student)

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

Relational algebra expressions



SQL	Result	Relational algebra

`select name, salary`
`from E`
`where salary < 200`
	name | **salary**
	John | 100
	Tom | 100
		$\textbf{PROJECT}_{name,\ salary}\ (\textbf{SELECT}_{salary\ <\ 200}(E))$
		
		or, step by step, using an intermediate result
		
		$Temp \leftarrow \textbf{SELECT}_{salary\ <\ 200}(E)$
		$Result \leftarrow \textbf{PROJECT}_{name,\ salary}(Temp)$

c) Union Operation (∪)

This operation is used to fetch data from two relations(tables).

For this operation to work, the relations(tables) specified should have same number of attributes(columns) and same attribute domain.

Also the duplicate rows are automatically eliminated from the result.

Syntax: A ∪ 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,

$$\prod_{Student}(RegularClass) \cup \prod_{Student}(ExtraClass)$$

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

d) 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:

$\prod_{Student}(RegularClass) - \prod_{Student}(ExtraClass)$

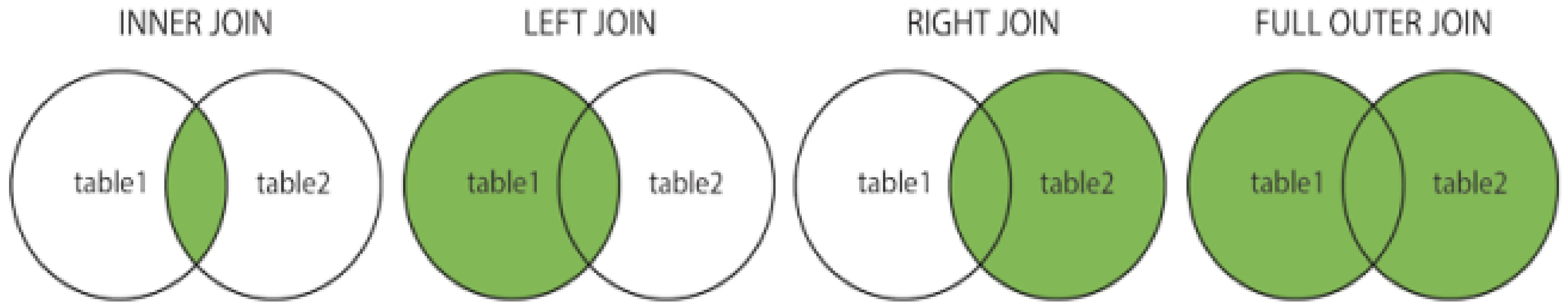
- **Cartesian Product (X)**

- This is used to combine data from two different relations(tables) into one and fetch data from the combined relation. Syntax: A X 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:
- σtime = 'morning' (RegularClass X ExtraClass)
- For the above query to work, both RegularClass and ExtraClass should have the attribute time

f) Join

Here are the different types of the JOINs in SQL:

- (INNER) JOIN ($\bowtie$) : Returns records that have matching values in both tables.
- LEFT (OUTER) JOIN (⟕): Return all records from the left table, and the matched records from the right table
- RIGHT (OUTER) JOIN (⟖): Return all records from the right table, and the matched records from the left table
- FULL (OUTER) JOIN (⟗): Return all records when there is a match in either left or right table


INNER JOIN
table1
table2
LEFT JOIN
table1
table2
RIGHT JOIN
table1
table2
FULL OUTER JOIN
table1
table2

Relational algebra exercise

Consider the following relational database schema consisting of the four relation schemas:

passenger (pid, pname, pgender, pcity)

agency (aid, aname, acity)

flight (fid, fdate, time, src, dest)

booking (pid, aid, fid, fdate)

1. **Get the complete details of all flights to pokhara**

$$\sigma_{\textit{destination} = \text{“pokhara ”}}(\text{flight})$$

2. **Get the details about all flights from pokhara to bharatpur.**

$$\sigma_{\textit{src} = \text{“pokhara”} \wedge \textit{dest} = \text{“bharatpur”}}(\text{flight})$$

3. **Find only the flight numbers for passenger with pid 123 for flights to pokhara before 25/12/2079.**

$$\Pi_{\textit{fid}}(\sigma_{\textit{pid} = 123}(\text{booking}) \bowtie \sigma_{\textit{dest} = \text{“pokhara”} \wedge \textit{fdate} < 25/12/2079}(\text{flight}))$$

4. **Find the passenger names for passengers who have bookings on at least one flight.**

$$\Pi_{pname}(\text{passenger} \bowtie \text{booking})$$

5. **Find the passenger names for those who do not have any bookings in any flights.**

$$\Pi_{pname}((\Pi_{pid}(\text{passenger}) - \Pi_{pid}(\text{booking})) \bowtie \text{passenger})$$

6. **Find the agency names for agencies that located in the same city as passenger with passenger id 123.**

$$\Pi_{aname}(\text{agency} \bowtie_{acity = pcity}(\sigma_{pid = 123}(\text{passenger})))$$

6. **Get the details of flights that are scheduled on both dates 01/12/2079 and 02/12/2070 at 16:00 hours.**

$$(\sigma_{\mathit{fdate = 01/12/2079\ \wedge\ time = 16:00}}(\text{flight})) \cap (\sigma_{\mathit{fdate = 02/12/2079\wedge\ time = 16:00}}(\text{flight}))$$

7. **Get the details of flights that are scheduled on either of the dates 01/12/2079 or 02/12/2079 or both at 16:00 hours.**

$$(\sigma_{\mathit{fdate = 01/12/2079\ \wedge\ time = 16:00}}(\text{flight})) \cup (\sigma_{\mathit{fdate = 02/12/2079\wedge\ time = 16:00}}(\text{flight}))$$

8. **Find the agency names for agencies who do not have any bookings for passenger with id 123.**

$$\Pi_{\mathit{aname}}(\text{agency} \bowtie (\Pi_{\text{aid}}(\text{agency}) - \Pi_{\text{aid}}(\sigma_{\mathit{pid = 123}}(\text{booking}))))$$

Another example.

Suppliers(sID, sName, address)

Parts(pID, pName, colour)

Catalog(sID, pID, price)

1. Find the names of all red parts.

Answer: $\Pi_{pName}(\sigma_{colour=\text{“red”}} Parts)$

2. Find all prices for parts that are red or green. (A part may have different prices from different manufacturers.)

Answer: $\mathbf{\Pi_{price}((\sigma_{colour=\text{“red”} \vee colour=\text{“green”}} Parts) \bowtie Catalog)}$

3. Find the sIDs of all suppliers who supply a part that is red or green.

Answer: $\Pi_{sID}((\sigma_{colour=\text{“red”} \vee colour=\text{“green”}} Parts) \bowtie Catalog)$

4. Find the names of all suppliers who supply a part that is red or green.

Answer: $\Pi_{sName}((\Pi_{sID}((\sigma_{colour=\text{“red”} \vee colour=\text{“green”}} Parts) \bowtie Catalog)) \bowtie Suppliers)$

Clients(CID, name, phone)

Staff(SID, name)

Appointments(CID, date, time, service, SID)

1. Find the appointment time and client name of all appointments for staff member Giuliano on Feb14. (Assume that you can compare a date value to “Feb 14” using “=”). At each step, use projection to pare down to only the attributes you need.

Answer:

Πname,time((ΠCID,SID,timeσdate=“Feb14”Appointments) ⋈ (ΠSIDσname=“Giuliano”Staff) ⋈ Clients)

Another example.

Person (name, age, gender)

name is a key

Frequents (name, pizzeria)

(name, pizzeria) is a key

Eats (name, pizza)

(name, pizza) is a key

Serves (pizzeria, pizza, price)

(pizzeria, pizza) is a key

Question

1. Find all pizzerias frequented by at least one person under the age of 18.
2. Find the names of all females who eat both mushroom and pepperoni pizza.
3. Find all pizzerias that serve at least one pizza that Amy eats for less than $10.00
4. Find all pizzerias that are frequented by only females or only males
5. Find the names of all people who frequent only pizzerias serving at least one pizza they eat

a. $\pi_{\text{pizzeria}}\left(\sigma_{\text{age}<18}(\text{Person}) \bowtie \text{Frequents}\right)$

b. $\pi_{\text{name}}\left(\sigma_{\text{gender}=\text{'female'} \wedge (\text{pizza}=\text{'mushroom'} \vee \text{pizza}=\text{'pepperoni'})}(\text{Person} \bowtie \text{Eats})\right)$

c. $\pi_{\text{name}}\left(\sigma_{\text{gender}=\text{'female'} \wedge \text{pizza}=\text{'mushroom'}}(\text{Person} \bowtie \text{Eats})\right) \cap$
$\pi_{\text{name}}\left(\sigma_{\text{gender}=\text{'female'} \wedge \text{pizza}=\text{'pepperoni'}}(\text{Person} \bowtie \text{Eats})\right)$

d. $\pi_{\text{pizzeria}}\left(\sigma_{\text{name}=\text{'Amy'}}(\text{Eats}) \bowtie \sigma_{\text{price}<10}(\text{Serves})\right)$

e.
$$\begin{pmatrix} \pi_{\text{pizzeria}}(\sigma_{\text{gender}=\text{'female'}}(\text{Person}) \bowtie \text{Frequents}) - \\ \pi_{\text{pizzeria}}(\sigma_{\text{gender}=\text{'male'}}(\text{Person}) \bowtie \text{Frequents}) \end{pmatrix} \cup \begin{pmatrix} \pi_{\text{pizzeria}}(\sigma_{\text{gender}=\text{'male'}}(\text{Person}) \bowtie \text{Frequents}) - \\ \pi_{\text{pizzeria}}(\sigma_{\text{gender}=\text{'female'}}(\text{Person}) \bowtie \text{Frequents}) \end{pmatrix}$$

g. $\pi_{\text{name}}(\text{Person}) - \pi_{\text{name}}\left(\text{Frequents} - \pi_{\text{name},\text{pizzeria}}(\text{Eats} \bowtie \text{Serves})\right)$

The end

Any query ?