Chapter 3

Relational Model

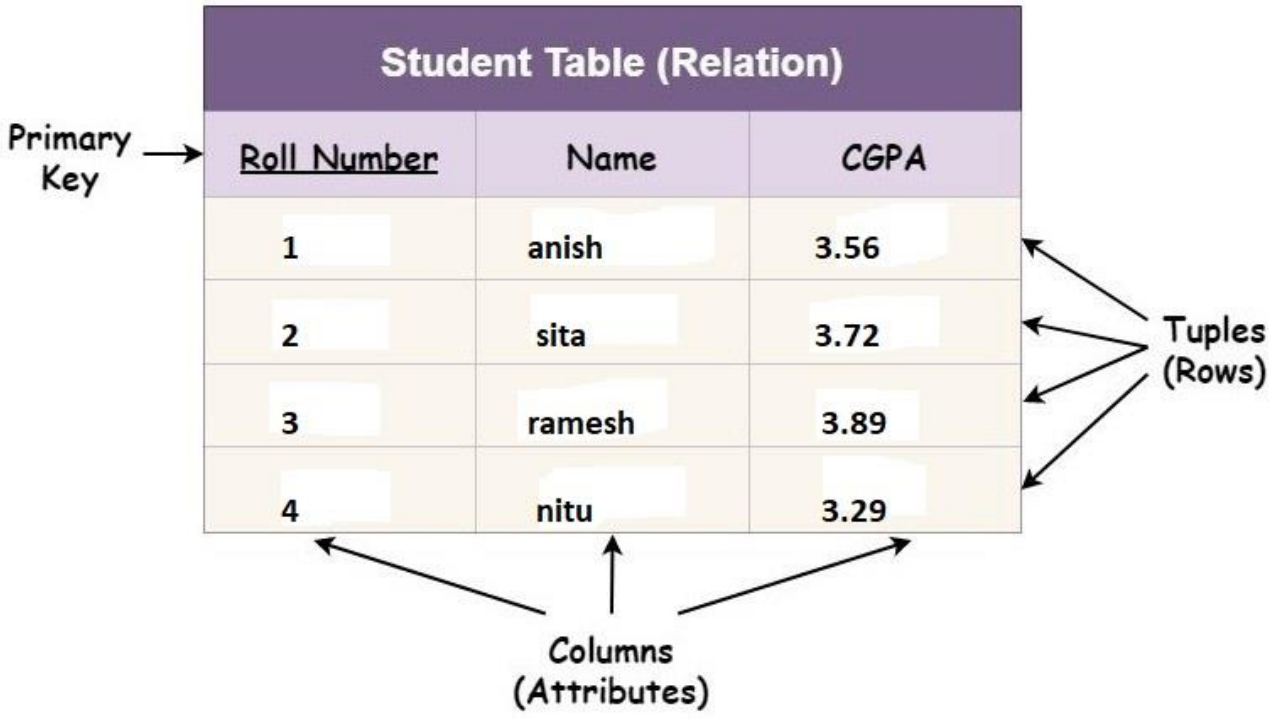
Definitions and Terminology

- Relational model is today a primary data model in which database is represented as a collection of Relation where each relation is represented by a two dimensional table.
- The relational model was first proposed by E.F. Codd in 1970.
- Because of its simplicity, the relational model is now the dominant model for commercial data processing operation.

Structure of relational model

- In this model, the data is organized into a collection of two-dimensional inter-related tables, also known as relations.
- Each relation is a collection of columns and rows.
    - ✓ column represents the attributes of an entity
    - ✓ the rows (or tuples) represents the records.

Consider a case we wish to store the name, the CGPA attained, and the roll number of all the students of a particular class. This structured data can be easily stored in a table as described below.



As we can notice from the above relation:

- Any given row of the relation indicates a student i.e. the row of the table describes a real-world entity.
- The columns of the table indicate the attributes related to the entity. In this case, the roll number, name and CGPA of student.

- ✓ Relational database is a collection of organized set of tables related to each other, and from which data can be accessed easily.
- ✓ A Relational Database management System (RDBMS) is a database management system based on the relational model .It is used to manage Relational database.
- ✓ Examples of RDBMS are Oracle, MySQL, Microsoft SQL Server, PostgreSQL etc.

As discussed earlier, a relational database is based on the relational model. This database consists of various components based on the relational model. These include:

Relation: Two-dimensional table used to store a collection of data elements.

Tuple: Each row of a table is known as record. It is also known as tuple. For example, the following row is a record that we have taken from the above table.

1	anish	3.56

Attribute: Column of the relation, depicting properties that define the relation.
Eg. Roll_Number, Name, CGPA

Domain: A domain is a set of permitted values for an attribute in table. For example, a domain of CGPA must be in the range of 0 to 4.

Relation Schema: A relation schema defines the structure of the relation and represents the name of the relation with its attributes. e.g. student (Roll_Number, Name, CGPA) is the relation schema for **student**. If a schema has more than 1 relation, it is called Relational Schema.

Relational Instance: It is the collection of records present in the relation at a given time. Above table shows the relation instance of **student** at a particular time. It can change whenever there is an insertion, deletion, or update in the database.

Degree: It is the total number of attributes present in the relation.eg. The **Student** relation defined above has degree 3.

Cardinality: It specifies the number of entities involved in the relation i.e., it is the total number of rows present in the relation. The **student** relation defined above has cardinality 4.

Relation Keys: It is an attribute or a group of attributes that can be used to uniquely identify an entity in a table or to determine the relationship between two tables. Relation keys can be of 6 different types:

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

Properties of Relational model

- Each relation (or table) in a database has a unique name
- An entry at the intersection of each row and column is atomic (Each relation cell contains exactly one atomic (single) value)
- Each row is unique; no two rows in a relation are identical
- Each attribute (or column) within a table has a unique name
- Tuples in a relation do not have to follow a significant order as the relation is not order-sensitive.
- Similarly, the attributes of a relation also do not have to follow certain ordering, it's up to the developer to decide the ordering of attributes.

Advantages of Relational model

- ✓ A relational database model is much simpler compared to other data models because data is stored in the form of rows and columns.
- ✓ Since there are several tables in a relational database, certain tables can be made to be confidential. These tables are protected with username and password such that only authorized users will be able to access them. The users are only allowed to work on that specific table.
- ✓ Relational database uses primary keys and foreign keys to make the tables interrelated to each other. Thus, all the data which is stored is non-repetitive. Which means that the data does not duplicate. Therefore, the data stored can be guaranteed to be accurate.
- ✓ It is flexible, so one can get the data in the form which he/she wants. He/she can extract the information very easily and information can also be manipulated by using various operators such as project, join, etc.
- ✓ The Structure of Relational database can be changed without having to change any application.

Disadvantages of relational model

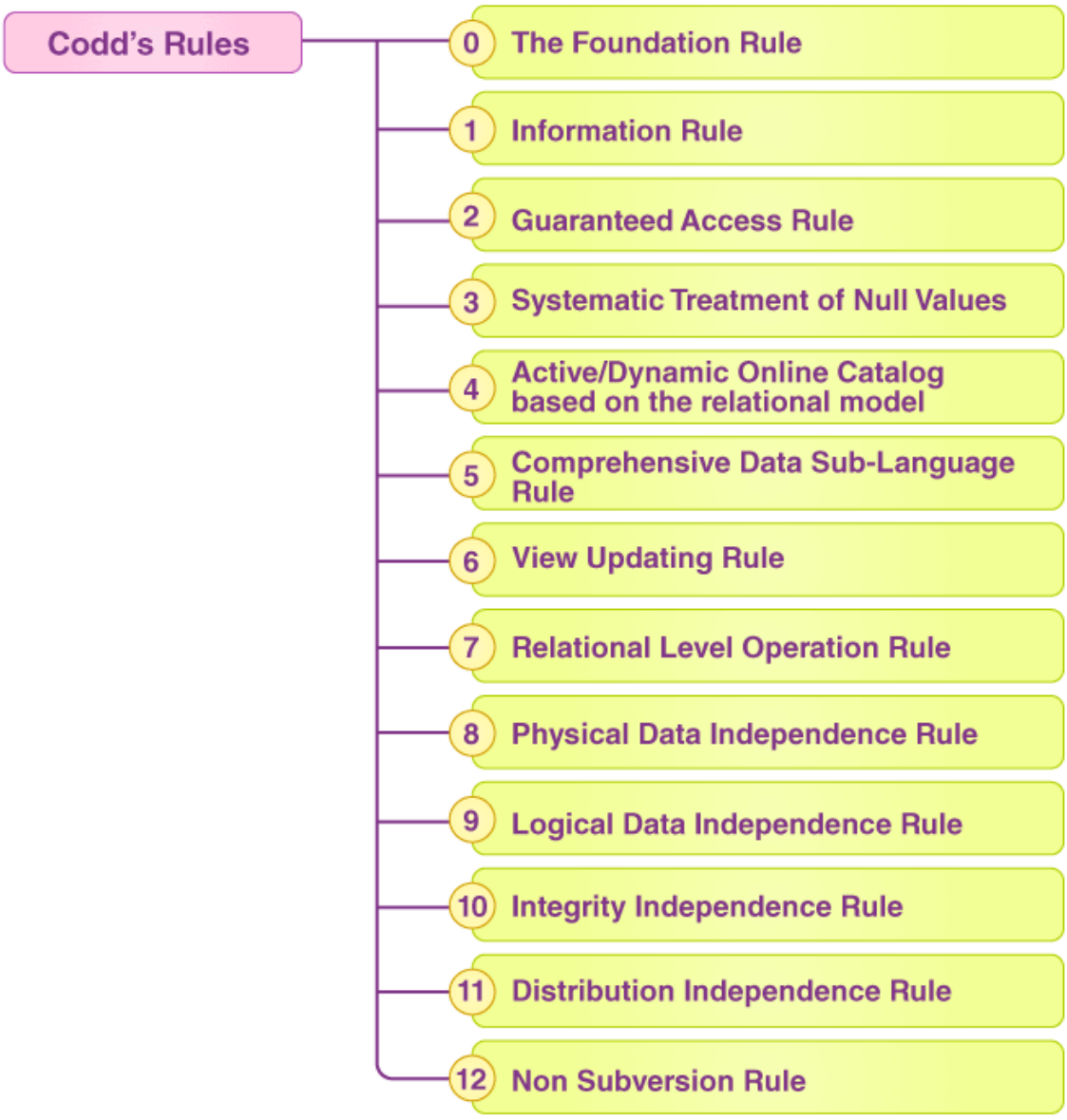
- ✓ The performance of the relational model depends upon the number of relations present in the database.
- ✓ Hence, as the number of tables increases, the requirement of physical memory increases.
- ✓ The structure becomes complex and there is a decrease in the response time for the queries.
- ✓ Because of all these factors, the cost of implementing a relational database increase.

Difference between DBMS and RDBMS

DBMS	RDBMS
DBMS applications store data as file.	RDBMS applications store data in a tabular form.
Data elements need to access individually.	Multiple data elements can be accessed at the same time.
No relationship between data.	Data is stored in the form of tables which are related to each other.
Normalization is not present in DBMS.	Normalization is present in RDBMS
DBMS does not support distributed database.	RDBMS supports distributed database.
DBMS is meant to be for small organization and deal with small data.	RDBMS is designed to handle large amount of data.
The software and hardware requirements are low.	The software and hardware requirements are higher.
The data in a DBMS is subject to low security levels with regards to data manipulation.	It features multiple layers of security while handling data.
Examples: Window Registry, Forxpro, dbase III plus etc.	Examples: MySQL, PostgreSQL, SQL Server, Oracle, Microsoft Access etc.

E.F. Codd's 12 Rules for RDBMS

- ✓ E.F Codd was a Computer Scientist who invented the Relational model for Database management. Based on relational model, the Relational database was created.
- ✓ Codd proposed 13 rules popularly known as Codd's 12 rules to test DBMS's concept against his relational model.
- ✓ Codd's rule actualy define what quality a DBMS requires in order to become a Relational Database Management System(RDBMS).


Codd's Rules
0 The Foundation Rule
1 Information Rule
2 Guaranteed Access Rule
3 Systematic Treatment of Null Values
4 Active/Dynamic Online Catalog based on the relational model
5 Comprehensive Data Sub-Language Rule
6 View Updating Rule
7 Relational Level Operation Rule
8 Physical Data Independence Rule
9 Logical Data Independence Rule
10 Integrity Independence Rule
11 Distribution Independence Rule
12 Non Subversion Rule

Rule 0: The Foundation Rule

The database must be in relational form. So that the system can manage the database through its relational capabilities.

Rule 1: The Information Rule

A database contains various information, and this information must be stored in each cell of a table in the form of rows and columns.

Rule 2: The Guaranteed Access Rule

Every single or precise data (atomic value) may be accessed logically from a relational database using the combination of primary key value, table name, and column name.

Rule 3: The Systematic Treatment of Null Values

This rule defines the systematic treatment of Null values in database records. The null value has various meanings in the database, like missing the data, no value in a cell, inappropriate information, unknown data and the primary key should not be null.

Rule 4: The Dynamic/Active Online Catalog on the basis of the Relational Model

Database dictionary(catalog) is the structure description of the complete Database and it must be stored online. The Catalog must be governed by same rules as rest of the database. The same query language should be used on catalog as used to query database.

Rule 5: The Comprehensive Data SubLanguage Rule

The relational database supports a variety of languages, and in order to access the database, the language has to be linear, explicit, or a well-defined syntax, character strings. It must support the following operations: view definition, integrity constraints, data manipulation, data definition, as well as limit transaction management. It is considered a DB violation if the DB permits access to the data and information without the use of any language.

Rule 6: The View Updating Rule

A view table can theoretically be updated, and DB systems must update them in practice.

Rule 7: The Relational Level Operation (or High-Level Insert, Delete, and Update) Rule

A database system should follow high-level relational operations such as insert, update, and delete in each level or a single row. It also supports union, intersection and minus operation in the database system.

Rule 8: The Physical Data Independence Rule

The working of a database system should be independent of the physical storage of its data. If a file is modified (renamed or moved to another location), it should not interfere with the working of the system.

Rule 9: The Logical Data Independence Rule

It indicates that any modifications made at the logical level (or the table structures) should not have an impact on the user's experience (application). For example, if a table is split into two separate tables or into two table joins in order to produce a single table, the application at the user view should not be affected.

Rule 10: The Integrity Independence Rule

A database must maintain integrity independence when inserting data into table's cells using the SQL query language. All entered values should not be changed or rely on any external factor or application to maintain integrity. It is also helpful in making the database-independent for each front-end application

Rule 11: The Distribution Independence Rule

This rule denotes that a database must function properly even if it's stored in multiple locations and used by various end-users. Let's say a person uses an application to access the database. In such a case, they must not be aware that another user is using the same data, and thus, the data they always obtain is only available on one site. The database can be accessed by end-users, and each user's access data must be independent in order for them to run SQL queries.

Rule 12: The Non-Subversion Rule

The non-subversion rule defines RDBMS as a SQL language to store and manipulate the data in the database. If a system has a low-level or separate language other than SQL to access the database system, it should not subvert or bypass integrity to transform data**.**

Relational Algebra

- ✓ The relational algebra is a procedural query language.
- ✓ It consist set of operation that takes one or more relations as inputs and produce a new relation as output.
- ✓ The fundamental operations in relational algebra are selection, projection, union, set difference, Cartesian product, and rename.
- ✓ Set intersection, natural join, division and assignments other operations of relational algebra which can be define in terms of fundamental operations.

1. Fundamental Operations

- ✓ The fundamental operations **selection, projection and rename** on one relation so they called unary operations.
- ✓ Others operations **union, set difference and Cartesian product** operates on pairs of relations and so called binary operations.

1.1 Selection Operation

- ✓ The Select Operation selects tuples that satisfy a given predicate.
- ✓ Select is denoted by a lowercase Greek letter sigma (σ), with the predicate appearing as a subscript.
- ✓ The relation is specifying within parentheses after σ. That is, general structure of selection is $\sigma_p(r)$ where p is selection predicate.
- ✓ Formally, selection operation define as
  $\sigma_p(r) = \{t | t \in r \text{ and } p(t)\}$
  where p is formula in propositional calculus consisting terms connected by connectives: ^(and), ∨ (or), ¬ (not).
- ✓ Each term is in the format <attribute>op<attribute>or <constant>
  where op is one of the comparison operators: =,≠ ,<,≤,>,≥

Let us consider the following employee relation

emp_id	name	department	salary
1	ramesh	civil	68000
2	krishna	computer	75000
3	rita	software	65000
4	sita	mechanical	32000
5	juna	mechanical	85000
6	nikita	computer	60000
7	anish	civil	55000

Find all the employees working on computer department

$\sigma_{department="computer"}(employee)$

output

emp_id	name	department	Salary
2	krishna	computer	75000
6	nikita	computer	60000

Find all the employees whose salary is greater than 70000

$\sigma_{salary>70000}(employee)$

output

emp_id	name	department	salary
2	krishna	computer	75000
5	juna	mechanical	85000

Find all the employees with name sita and department is mechanical

$\sigma_{name= "sita" \wedge department="mechanical"}(employee)$

emp_id	name	department	salary
4	sita	mechanical	32000

Find all the employees whose department is either civil or computer

$\sigma_{department= "civil" \vee department="computer"}(employee)$

emp_id	name	department	salary
1	ramesh	civil	68000
2	krishna	computer	75000
6	nikita	computer	60000
7	anish	civil	55000

Find all the employees whose department is either civil or computer and name is not nikita

$\sigma_{((department="civil" \vee department="computer") \wedge name \neq nikita)}(employee)$

emp_id	name	department	salary
1	ramesh	civil	68000
2	krishna	computer	75000
7	anish	civil	55000

1.2 Projection operation

- ✓ The projection operation retrieves tuples for specified attributes of relation.
- ✓ It eliminates duplicate tuples in relation.
- ✓ The projection is denoted by uppercase Greek letter pi (∏).
- ✓ We need to specify attributes that we wish to appear in the result as a subscript to ∏.
- ✓ The general structure of projection is

$\prod_{A1,A2,\ldots,Ak}(r)$
where A1, A2, . .Ak are attributes of relation r.

Example:

Find name and their salary from employee relation

$\prod_{name,salary}$(employee)

output

name	salary
ramesh	68000
krishna	75000
rita	65000
sita	32000
juna	85000
nikita	60000
anish	55000

Composition of relational operations

Relational algebra operations can be composed together into relational-algebra expression. This required for complicated query.

Example:

Find name and salary of employees of civil department

$\prod_{name,\ salary}(\sigma_{department="civil"}$ (employee))

output

name	salary
ramesh	68000
anish	55000

Find name and department of employees whose salary is less than or equals to 60000

$\Pi_{name,\ department}(\sigma_{salary\ \leq\ 60000}(employee))$

output

name	department
sita	mechanical
nikita	computer
anish	civil

1.3 Union operation

Suppose r and s are two relations, then union operation contains all tuples that appear in r, s, or both.

The union of two relations r and s are defines as

$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$

For r U s to be valid, it must hold

- ✓ r,s must have same arity (same number of attributes)
- ✓ The attribute domain must be compatible (e.g. domain of ith column of r must deals with same type of domain of ith column of s)
- ✓ Duplicate rows are eliminated by this operations

Let us consider two relations

depositer

customer_name	account_no
ram	A-101
sita	A-105
hari	A-205
nishant	A-405
gita	A-505

borrower

customer_name	loan_no
aditya	L-224
nishant	L-185
mahesh	L-150
gita	L-174
ronish	L-145

Example:

Find name of all customers who have either account or loan

$\Pi_{customer\_name}$(depositor) U $\Pi_{customer\_name}$(borrower**)**

Output

customer_name
ram
sita
hari
nishant
gita
Aditya
Mahesh
ronish

1.4 Set difference Operation

- ✓ The set difference allows us to find tuples that are in one relation but not in another relation. The expression r-s produces a relation containing those tuples in r but not in s.
- ✓ Formally, let r and s are two relations then their difference r-s define as

  r-s=={t | t∈r and t∉s}

- ✓ The set difference must be taken between compatible relations. For r-s to be valid, it must hold

  - r and s must have the same arity (same number of attributes)
  - attribute domains of r and s must be compatible

Example:

Find name of all customer of the bank who have account but not loan

$\Pi_{customer\_name}$(depositor) - $\Pi_{customer\_name}$(borrower)

customer_name
ram
sita
hari

1.5 Cartesian product

- ✓ It allows us to combine information from any two relations.
- ✓ The Cartesian product operation denoted by cross (X).
- ✓ Cartesian product of two relations r and s, denoted by r X s returns a relation instance whose schema contains all the fields of r (in same order as they appear in r(followed all field of s (in the same order as they appear in s).
- ✓ The result of r×s contains one tuples <r,s> (concatenation of tuples of r and s) for each pair tuples t∈r, q∈s.

Formally, r X s={<t,q>|t∈r and q∈s}

Note:

If r and s are two relation having n and m number of attributes and p and q number of records respectively, then the Cartesian product of these two relation denoted by r X s results a new relation having (n+m) numberof attributes and (p*q) number of records

Example:
Relation Employee

emp_id	name
1	anish
2	sita
3	nitesh

Relation Department

dept_id	dept_name
1	computer
2	civil
3	electrical

Employee X Department

emp_id	name	dept_id	dept_name
1	anish	1	computer
1	anish	2	civil
1	anish	3	electrical
2	nitesh	1	computer
2	nitesh	2	civil
2	nitesh	3	electrical
3	sita	1	computer
3	sita	2	civil
3	sita	3	electrical

Example 2:

Relation borrower

customer_name	loan_number
X	L01
Y	L02

Relation loan

loan_number	branch_name	amount
L01	B1	5000
L02	B2	6000

Query: Find all customer who taken loan from branch "B1".

$\prod_{customer\_name}(\sigma_{borrower.loan\_number=loan.loan\_number}(\sigma_{branch\_name="B1"}(borrower \times loan)))$

Process:

borrower×loan

customer_name	borrower.loan_number	loan.loan_number	branch_name	amount
X	L01	L01	B1	5000
X	L01	L02	B2	6000
Y	L02	L01	B1	5000
Y	L02	L02	B2	6000

$\sigma_{branch\_name="B1"}(borrower \times loan)$

customer_name	borrower.loan_number	loan.loan_number	branch_name	amount
X	L01	L01	B1	5000
Y	L02	L01	B1	5000

$\sigma_{borrower.loan\_number=loan.loan\_number}(\sigma_{branch\_name="B1"}(borrower \times loan))$

customer_name	borrower.loan_number	loan.loan_number	branch_name	amount
X	L01	L01	B1	5000

$\prod_{customer\_name}(\sigma_{borrower.loan\_number=loan.loan\_number}(\sigma_{branch\_name="B1"}(borrower \times loan)))$

customer_name
X

1.6 Rename operation

- ✓ The result of relational-algebra expression does not have a name to refer it. It is better to give name to result relation.
- ✓ The rename operator is denoted by lower case Greek letter rho (ρ).
- ✓ Rename operation in relation-algebra expressed as $\rho_x(E)$ where E is a relational algebra expression and x is name for result relation. It returns the result of expression E under the name x.
- ✓ Since a relation r is itself a relational-algebra expression thus, the rename operation can also apply to rename the relation r (i.e. to get same relation under a new name).
- ✓ Rename operation can also used to rename attributes of relation. Assume a relational algebra expression E has arity n. Then expression $\rho_{x(A1,A2,...An)}(E)$ returns the result of expression E under the name x and it renames attributes to A1,A2,..,An.

Let us consider the relation

Customer(customer_id,customer_name,customer_city)

Example1:

To find all the customer_name from this relation algebra expression can be written as

$\prod_{customer\_name}(customer)$

This result of the expression can be renamed as

$\rho_{cname}(\prod_{customer\_name}(customer))$

Example 2:

Rename the relation named customer

$\rho_{newcustomer}(customer)$

Here relation named customer can be renamed as newcustomer.

Example 3:

Relation named can also be renamed as well their attributes also

$\rho_{c(cid,cname,ccity)}$ (Customer)

Her relation name customer is renamed as c and their attributes customer_id,customer_name,customer_city are renamed as cid,cname and ccity respectively.

2. Additional relational operations

2.1 Set intersection operation

- ✓ Suppose r and s are two relations, then set intersection operation contains all tuples that are in both r and s.
- ✓ Let r and s are two relation having same arity and attributes of r and s are compatible then their intersection r∩ s define as r∩ s={t|t∈r and t∈s}

Example

Find all customer who have both loan and account

$\Pi_{customer\_name}$(borrower) ∩ $\Pi_{customer\_name}$ (depositor**)**

customer_name
nishant
gita

2.2 Division

- ✓ The division operator takes two relations and builds another relation consisting of values of an attribute of one relation that matches all the values in another relation
- ✓ Division operator r÷s or r/s can be applied if and only if, Attributes of s is proper subset of Attributes of r.
- ✓ The relation returned by division operator will have attributes = (All attributes of r – All Attributes of s)

Example 1:

K	X	Y
1	A	2
1	B	4
2	A	2
3	B	4
4	B	4
3	A	2

Relation r

X	Y
A	2
B	4

Relation s

r÷ s:

K
1
3

Example 2:

Consider the following relation

subject

subject_name	course_name
DBMS	CMP
C++	ELX
C++	CMP
OS	CMP

course

course_name
CMP
ELX

Find the name of subject taught in all course

subject÷ course

subject_name
C++

2.3 Assignment Operation

- ✓ The assignment operation provides convenient way to express complex query.
- ✓ The assignment operation denoted by ←, works like assignment in programming language.
- ✓ The evaluation of an assignment does not result any relation being displayed to the user. But the result of the expression to the right of the ← is assigned to the relation variable.
- ✓ This relation variable may used in subsequent expressions.
- ✓ With the assignment expression, a query can be written as a sequential program consisting a series of assignments followed by an expression whose value is displayed as the result of the query.

Example: Find all customer who taken loan from bank as well as has bank account.

temp1←$\prod_{customer\_name}$(borrower)

temp2←$\prod_{customer\_name}$(depositor)

result← temp1 ∩ temp2

2.4 Join Operation

- ✓ A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied.
- ✓ It is denoted by ⋈.
- ✓ Join operation is essentially a Cartesian product followed by a selection criterion.
- ✓ In its simplest form, the join operation is just the cross product of two relations which produce large result size but using this operation, one record from relation **R** and on record from Relation **S** can be combined together to form the result if the combination satisfies the join condition.
- ✓ The join condition can be =, , ≤, ≥, ≠

Various forms of join operation are:

1. Equi Join
2. Natural Join
3. Outer Join
    i. Left Outer Join
    ii. Right Outer Join
    iii. Full Outer Join

Before discussing various joins let us consider the following relations

Student

stu_id	sname	address
1	Anish	Kathmandu
2	Rita	Lalitpur
3	Krishna	Bhaktapur
4	Nitu	Butwal

Course

stu_id	cname	fee
1	Java	25000
3	Python	22000
4	PHP	18000
5	MERN stack	30000

1. Equi join

An equijoin is an operation that combines two relations based on the equality of values in specified attributes.

It is denoted by the symbol ⋈ and is defined as follows:

R ⋈ $_{<condition>}$ S

Where R and S are the relations to be joined, and <condition> represents the equality condition on the common attribute(**S**).

student ⋈ $_{Student.stu\_id = Course.stu\_id}$ course

stu_id	sname	addreess	cname	fee
1	Anish	Kathmandu	Java	25000
3	Krishna	Bhaktapur	Python	22000
4	Nitu	Butwal	PHP	18000

The resulting relation includes only the tuples that have matching "stu_id" values in both the "Student" and "Course" relations.

2. Natural Join

- ✓ Natural join automatically matches and combines tuples from two relations based on the common attribute(s) with the same name.
- ✓ In example given below, the common attribute is "stu_id".
- ✓ Natural join does not use any comparison operator
- ✓ The name and type of the attribute must be same.
- ✓ It eliminates duplicate attributes in the result.
- ✓ It is denoted by ⋈

Student ⋈ course

stu_id	sname	addreess	cname	fee
1	Anish	Kathmandu	Java	25000
3	Krishna	Bhaktapur	Python	22000
4	Nitu	Butwal	PHP	18000

The resulting relation includes only the tuples that have matching attribute values with the same name ("stu_id" in this case).

3. Outer join

- ✓ It is an extension of natural join to deal with missing values of relation.
- ✓ It is used to retrieve all records from relation, even for those tuples with no matching value in the other relation based on the join condition.
- ✓ In such cases, it returns NULL as the value for the missing attributes.

It is further classified as:

i. Left Outer Join
ii. Right Outer Join
iii. Full Outer Join

i) Left outer join (⟕)

- ✓ Left outer join returns all tuples from the left relation and the matching tuples from the right relation.
- ✓ If there is no match in the right relation, NULL values are used for the attributes of the right relation in the resulting relation

Student ⟕ course

stu_id	sname	address	cname	fee
1	Anish	Kathmandu	Java	25000
2	Rita	Lalitpur	NULL	NULL
3	Krishna	Bhaktapur	python	22000
4	Nitu	Butwal	PHP	18000

The resulting relation includes all tuples from the left relation ("Student") and the matching tuples from the right relation ("Course"). The NULL values in the "cname" and "fee" columns indicate that there is no match for the corresponding tuples in the "Course" relation.

ii) Right Outer Join(⟖)

- ✓ Right outer join returns all tuples from the right relation and the matching tuples from the left relation.
- ✓ If there is no match in the left relation, NULL values are used for the attributes of the left relation in the resulting relation.

Student ⟖ Course

stu_id	sname	address	cname	fee
1	Anish	Kathmandu	Java	25000
3	Krishna	Bhaktapur	Python	22000
4	Nitu	Butwal	PHP	18000
5	NULL	NULL	MERN stack	30000

The resulting relation includes all tuples from the right relation ("Course") and the matching tuples from the left relation ("Student"). The NULL values in the "sname" and "address" columns indicate that there is no match for the corresponding tuples in the "Student" relation.

iii) Full outer join(⟗)

- ✓ Full outer join returns all tuples from both relations.
- ✓ If there is no match for a tuple in either relation, NULL values are used for the attributes of the relation that does not have a match.

Student ⟗ Course

stu_id	sname	address	cname	fee
1	Anish	Kathmandu	Java	25000
2	Rita	Lalitpur	NULL	NULL
3	Krishna	Bhaktapur	python	22000
4	Nitu	Butwal	PHP	18000
5	NULL	NULL	MERN stack	30000

The resulting relation includes all tuples from both the left relation ("Student") and the right relation ("Course"). The NULL values indicate that there is no match for the corresponding tuples in either relation.

3. Extended Relational- Algebra Operations

3.1 Generalized Projection

Generalized projection operation allows arithmetic and string functions in the projection list.

The generalized projection has the form

$\prod_{F1,F2,..Fn}(E)$

where E is any relational algebra expression. Each F1,F2, . .Fn are arithmetic expression involving constants and attributes in the schema of E.

Example 1:

Suppose a relation

credit_info(customer_name, credit_limit,credit_balance)

Find how much more each person can spend.

$\prod_{customer\_name,\ credit\_limit - credit\_balance}(credit\_info)$

Example 2:

Suppose relation

employee(employee_id,ename,salary)

Find employee and their corresponding bonus, assume that bonus for each employee is 10% of his/her salary.

$\prod_{ename,salary*1.10}(employee)$

3.2 Aggregation

- ✓ The aggregate operation permits the use of aggregate functions such as min ,average etc. on set of values.
- ✓ Aggregation function takes a collection of values and returns a single value as a result.

Some aggregate functions are

avg: average value
min: minimum value
max: maximum value
sum: sum of values
count: number of values

- ✓ Aggregate operation in relational algebra denoted by the symbol *g* (i.e. $\mathcal{G}$ is the letter G in calligraphic font)

$$_{G1, G2, \ldots, Gn}\mathcal{G}_{F1(A1), F2(A2), \ldots, Fn(An)}(E)$$

- *E* is any relational-algebra expression
- $G_1, G_2 \ldots, G_n$ is a list of attributes on which to group (can be empty)
- Each F_i is an aggregate function
- Each A_i is an attribute name

Let us consider the following relation named Employee

emp_id	name	department	salary
1	Ramesh	Civil	55000
2	Prizma	Computer	65000
3	Riya	IT	52000
4	Narayan	Civil	25000
5	Nimesh	computer	5000

Find the average salary of employee

$$\mathcal{G}_{avg(salary)}(\text{Employee})$$

Find the minimum salary of employee

$$\mathcal{G}_{min(salary)}(\text{Employee})$$

Find the name of employee who have highest salary

$$\Pi_{name}(\sigma_{salary=\mathcal{G}_{max(salary)}(\text{Employee})})$$

Find the total salary paid by employee in each department

$$_{department}\mathcal{G}_{sum(salary)}(\text{Employee})$$

department	salary
Civil	80000
Computer	70000
IT	52000

Modification of database

Insertion, deletion and updating operations are responsible for database modification

Deletion

- ✓ A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.
- ✓ Can delete only whole tuples; cannot delete values on only particular attributes
- ✓ A deletion is expressed in relational algebra by:

$$r \leftarrow r - E$$

where *r* is a relation and *E* is a relational algebra query.

Let us consider the following relation

Employee(employee_id,name,department,salary)

Example 1: Delete information of all employee from civil department

Employee ← Employee- $\sigma_{\text{department="civil"}}$ (Employee)

Example 2:

Delete all records of employee with salary in the range 25000 to 60000

$Employee \leftarrow Employee - \sigma_{(salary \geq 25000) \wedge (salary \leq 60000)}$ (Employee)

Insertion

To insert data into relation we can either specify a tuple to be inserted or write a query whose result is a set of tuples to be inserted.

In relational-algebra, an insertion is express by $r \leftarrow r \cup E$
where r is a relation and E is a relational algebra expression

Example:

Insert the information of employee whose employee id is 10, named "rikesh" working in civil department and having salary 50000

Employee ← Employee $\cup$ {(10,"rikesh", "civil",50000)}

Updating

Updating allow to change a value in a tuple without changing all values in tuple. In relational algebra, updating express by
$r \leftarrow \prod_{F1,F2,\ldots,Fn}(r)$

where each F_i is either

- ✓ the *i*th attribute of *r*, if the *i*th attribute is not updated, or,
- ✓ expression involving only constant and attributes of r, if the attribute is to be updated. It gives the new value for the attribute.

Example:

Increase salary of all employees by 5 %

$Employee \leftarrow \prod_{employee\_id,name,deparment,salary*1.05}(Employee)$

Increase the salary of employee by 20% if salary is less than 50000 and increase salary of remaining employees by 10%

$Employee \leftarrow \prod_{employee\_id,\ name,,department,\ salary\ *\ 1.2}(\sigma_{salary<50000}(Employee)) \cup \prod_{employee\_id,name,department,salary*1.1}(\sigma_{salary \geq 50000}(Employee))$

Increase salary of employees of civil department by 15%

$Employee \leftarrow \prod_{employee\_id,name,department,salary*1.15}(\sigma_{department= \text{“civil}}(Employee))$

$U\ (Employee - \sigma_{department= \text{“civil}}(Employee))$

Update an employee so that ram now shifted to computer department

$Employee \leftarrow \prod_{employee\_id,name,\text{”computer”},salary}(\sigma_{name= \text{“ram”}}(Employee))\ U$

$(Employee - \sigma_{name=\text{“ram”}}(Employee))$

Note:

update operation will be in another form as well

$r \leftarrow \prod_{F1,F2,F3,\ldots Fn}(\sigma_p(r)) \cup (r - \sigma_p(r))$

Database schema

Database schema is a logical design of a database and the database instance is a snapshot of the data in the database at a given instant in time.

The concept of a **relation** corresponds to the programming language notation of a variable While the concept of a **relation schema** corresponds to the programming language notation of type definition. In general a relation schema consists of list of attributes and their corresponding domains.

The concept of a **relation instance** corresponds to the programming –language notation of a value of a variable. The value of a given variable may change with time; similarly the contents of a relation instance may change with time as the relation is updated. In contrast, the schema of a relation does not generally change.

For example, the relation schema for relation customer is express as

Customer (customer_id, customer_name, customer_city)

We may also specify domains of attributes as

Customer (customer_id: integer, customer_name: string,customer_city:string)

The below figure shows the instance of relation

customer_id	customer_name	customer_city
1	ronit	kathmandu
2	sita	pokhara
3	nitu	lalitpur

Let us consider university database example

Each course in a university may be offered multiple times, across different semesters, or even within a semester. We need a relation to describe each individual offering, or section, of the class.

The schema is section (course id, sec id, semester, year, building, room number, time slot id)

We need a relation to describe the association between instructors and the class sections that they teach. The relation schema to describe this association is

teaches (ID, course id, sec id, semester, year)

As we can imagine, there are many more relations maintained in a real university database.

Now, all the database schemas for university database can be listed as

instructor(id,name,dept_name,salary)
course(course_id,title,dept_name,credits)
department (dept_name, building, budget)
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)
advisor (s_id, i_id)
prereq(course_id,prereq_id)
takes (ID, course_id, sec_id, semester, year, grade)
classroom (building, room_number, capacity)
time_slot (time_slot_id, day, start_time, end_time)

Schema diagrams

A database schema, along with primary key and foreign key dependencies, can be depicted by schema diagrams. Figure given below shows the schema diagram for university organization. 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.

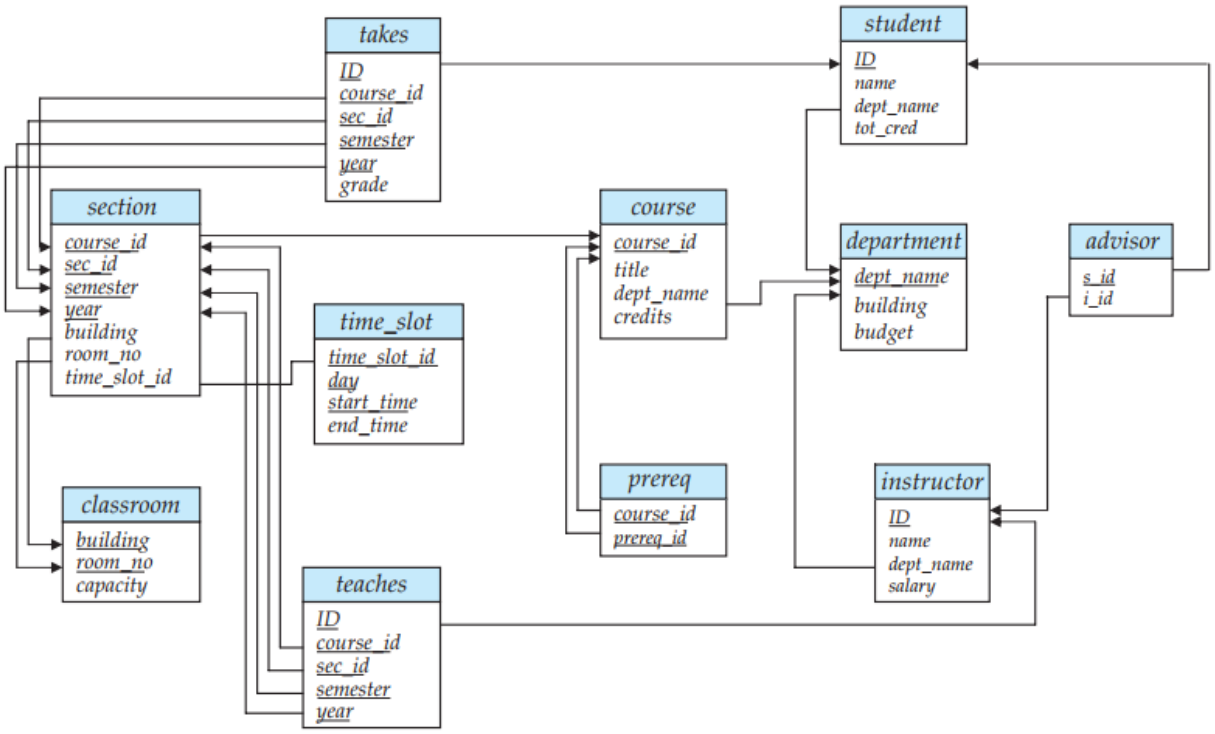


Figure:Schema diagram for the university database.

Assignment

Consider the following relations for a database that keeps track of student enrollment in courses and the books adopted for each course:

STUDENT(SSN,Name,Major,Bdate)
COURSE(Course#,Cname,Dept)
ENROLL(SSN,Course#,Quarter,Grade)
BOOK_ADOPTION(Course#,Quarter,Book_ISBN)
TEXT(Book_ISBN,Book_Title,Publisher,Author)

Draw a relational schema diagram specifying the foreign keys for this schema.

Views

- ✓ "Views" are virtual relations through which a selective portion of the data from one or more relations can be seen.
- ✓ It does not store any data on its own but provides an alternative way to present the data stored in the underlying relations.
- ✓ Views are defined based on queries, and they can be used to simplify complex queries, restrict access to certain data, or present a customized perspective of the data to different users or applications.

In some cases, it is not desirable for all users to see all the actual relations stored in the database.

For example consider the following relation

Employee(emp_id,emp_name,postion,salary,dept_id)
Department(dep_id,dept_name,location,budget)

Consider a person who needs to know information of employees with name, position and department name but not salary as well as other information, then this person should see a relation described, in the relational algebra, by

$$\Pi_{emp\_name,postion,dept\_name}(\text{Employee} \bowtie \text{Department})$$

But in such cases views can be created.

Views definition

A view is defined using the create view statement which has the form

create view v as <query expression>

where **<query expression>** is any legal relational algebra query expression.

- ☞ The view name is represented by v.
- ☞ Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- ☞ View definition is not the same as creating a new relation by evaluating the query expression
- ☞ Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.

Consider the view (named all_employees) consisting of emp_name, position and department

create view all_employees as
$\Pi_{emp\_name,postion,dept\_name}(\text{Employee} \bowtie \text{Department})$

We can find all employees of civil department by writing

$\Pi_{emp\_name}(\sigma_{dept\_name = \text{“civil”}}(all\_employees))$

Benefits of using views

- ✓ **Data Security:** Views can be used to enforce data security by limiting the access to sensitive information. By creating views that only expose certain columns or rows, we can control what data users can see and ensure that confidential or restricted information remains hidden.
- ✓ **Data Abstraction:** Views provide a level of abstraction, allowing users to interact with the data in a more simplified and intuitive way.
- ✓ **Query Simplification:** Views can be used to encapsulate complex or frequently used queries, making them easier to reuse.

Update through views

Database modifications expressed as views must be translated to modifications of the actual relations in the database.

Consider the person who needs to see all department data in the department relation except budget. The view given to the person, department_info is defined as:

create view department_info as
$\Pi_{dept\_id,dept\_name,location}$(Department)

Since we allow a view name to appear wherever a relation name is allowed, the person may write:

department_info $\leftarrow$ department_info $\cup$ {(1,“computer”, “kathmandu”)}

The previous insertion must be represented by an insertion into the actual relation department from which the view department_info is constructed.

An insertion into department requires a value for budget. The insertion can be dealt with by either.

- ✓ rejecting the insertion and returning an error message to the user.
- ✓ inserting a tuple (1,“computer”, “kathmandu”, null) into department relation

Data Dictionary storage

A relational database system needs to maintain data about the relations, such as the schema of the relations. In general, such "data about data" is referred to as metadata. Relational schemas and other metadata about relations are stored in a structure called the **data dictionary** or system catalog.

Among the types of information that the system must store are these:

- ✓ Names of the relations.
- ✓ Names of the attributes of each relation.
- ✓ Domains and lengths of attributes.
- ✓ Names of views defined on the database, and definitions of those views.
- ✓ Integrity constraints (for example, key constraints).

In addition, many systems keep the following data on users of the system:

- ✓ Names of authorized users.
- ✓ Authorization and accounting information about users.
- ✓ Passwords or other information used to authenticate users.

Further, the database may store statistical and descriptive data about the relations, such as:

- ✓ Number of tuples in each relation.
- ✓ Method of storage for each relation

The data dictionary may also note the storage organization (sequential, hash, or heap) of relations, and the location where each relation is stored:

- ✓ If relations are stored in operating system files, the dictionary would note the names of the file (or files) containing each relation.
- ✓ If the database stores all relations in a single file, the dictionary may note the blocks containing records of each relation in a data structure such as a linked list.

Assignment:

- How relational algebra is different from relational calculus? Define Tuple Relational Calculus and Domain Relational Calculus.

  (Refer: Text Book "Database system concept" by Abraham Silberschatz)