Unit 2

Part-1

Relational query languages

-

Topics

- Relational algebra,
- Tuple and domain Relational calculus,
- SQL3,
- DDL and DML constructs,
- Open source and Commercial DBMS -MYSQL, ORACLE, DB2, SQL server

Introduction

- A **query language** is a language in which a user requests information from the database
- Query languages can be categorized as either procedural or nonprocedural.
- In a **procedural language**, the user instructs the system to perform a sequence of operations on the database to compute the desired result.
- In a **nonprocedural language**, the user describes the desired information without giving a specific procedure for obtaining that information.

Introduction

- The relational algebra is **procedural**, whereas the tuple relational calculus and domain relational calculus are **nonprocedural**.
- The relational algebra consists of a set of operations that take one or two relations as input and produce a **new relation as their result**.
- The relational calculus uses **predicate logic** to define the result desired without giving any specific algebraic procedure for obtaining that result.

Introduction

- All procedural relational query languages provide a set of operations that can be applied to either a single relation or a pair of relations.
- These operations have the nice and desired property that their result is always a single relation.
- This property allows one to combine several of these operations in a modular way.
- Specifically, since the result of a relational query is itself a relation, relational operations can be applied to the results of queries as well as to the given set of relations.

Relational algebra

- The relational algebra is a *procedural* query language, serves as the basis for the SQL language.
- It consists of a set of operations that take one or two relations as input and produce a new relation as their result.

- The fundamental operations in the relational algebra are *select, project*, *union*, *set difference*, *Cartesian product,* and *rename*.
- In addition to the fundamental operations, there are several other operations—namely, *set intersection*, *natural join*, and *assignment*

Relational algebra : Operations

- Select
- Project
- Rename

unary operations

- Set Operators
  - Union
  - Intersect (Intersection)
  - Minus (Set Difference)
- Cartesian-Product
- Join operations

binary operations

The Select Operation

- **Operation:** Selects tuples from a relation that satisfy a given condition.
- It is used to select particular tuples from a relation.
- It selects particular tuples but all attribute from a relation.
- **Symbol:** σ (Sigma)
- **Notation:** $\sigma_{<condition>}$ (Relation)
- **Operators:** The following operators can be used in a condition.

=, !=, <, >, <=,>=, Λ(AND), V(OR)

Display the detail of students belongs to “CE” branch.

Student			
RollNo	**Name**	**Branch**	**SPI**
101	Raj	CE	8
102	Meet	ME	9
103	Harsh	EE	8
104	Punit	CE	9

$$\sigma_{Branch='CE'}(Student)$$

Output			
RollNo	**Name**	**Branch**	**SPI**
101	Raj	CE	8
104	Punit	CE	9

Display the detail of students belongs to "CE" branch and having SPI more than 8.

Student			
RollNo	**Name**	**Branch**	**SPI**
101	Raj	CE	8
102	Meet	ME	9
103	Harsh	EE	8
104	Punit	CE	9

$$\sigma_{Branch='CE' \wedge SPI>8}\ (Student)$$

Output			
RollNo	**Name**	**Branch**	**SPI**
104	Punit	CE	9

Display the detail of students belongs to either “EE” or “ME” branch.

Student			
RollNo	**Name**	**Branch**	**SPI**
101	Raj	CE	8
102	Meet	ME	9
103	Harsh	EE	8
104	Punit	CE	9

$$\sigma_{\text{Branch='EE'} \lor \text{Branch='ME'}} \text{(Student)}$$

Output			
RollNo	**Name**	**Branch**	**SPI**
102	Meet	ME	9
103	Harsh	EE	8

Display the detail of students whose SPI between 7 and 9.

Student			
RollNo	**Name**	**Branch**	**SPI**
101	Raj	CE	8
102	Meet	ME	9
103	Harsh	EE	8
104	Punit	CE	9

$$\sigma_{SPI>7 \wedge SPI<9} \; (Student)$$

Output			
RollNo	**Name**	**Branch**	**SPI**
101	Raj	CE	8
103	Harshan	EE	8

The Project Operation

- **Operation:** Selects specified attributes of a relation.

It selects particular attributes but all unique tuples from a relation.

- **Symbol:** Π (Pi)
- **Notation:** Π (attribute set) <Relation>

- Display rollno, name and branch of all students.

Student

RollNo	Name	Branch	SPI
101	Raj	CE	8
102	Meet	ME	9
103	Harsh	EE	8
104	Punit	CE	9

$$\Pi_{RollNo,\ Name,\ Branch}\ (Student)$$

Output

RollNo	Name	Branch
101	Raj	CE
102	Meet	ME
103	Harsh	EE
104	Punit	CE

- **Example:** Display rollno, name & branch of "ME" branch students.

Student

RollNo	Name	Branch	SPI
101	Raj	CE	8
102	Meet	ME	9
104	Punit	CE	9

Output-1

$$\sigma_{Branch='ME'}(Student)$$

RollNo	Name	Branch	SPI
102	Meet	ME	9

$$\Pi_{RollNo,\ Name,\ Branch}(\sigma_{Branch='ME'}(Student))$$

Output-2

RollNo	Name	Branch
102	Meet	ME

Rename

- **Operation:** It is used to rename a relation or attributes.
- **Symbol:** ρ (Rho)
- **Notation:**
    - $\rho_{x}(E)$
        - Returns a relation E under a new name X.
    - $\rho_{A1, A2. ..., An}(E)$
        - Returns a relation E with the attributes renamed to A1, A2, …., An.
    - $\rho_{x(A1, A2. ..., An)}(E)$
        - Returns a relation E under a new name X with the attributes renamed to A1, A2, …., An.

Example

Student		
Rno	Name	CPI
101	Raj	8
102	Meet	9
103	Suresh	7

$\rho_{\text{(RollNo, StudentName, CPI)}}$ (Student)

Student		
RollNo	StudentName	CPI
101	Raj	8
102	Meet	9
103	Suresh	7

Student		
Rno	Name	CPI
101	Raj	8
102	Meet	9
103	Suresh	7

ρ_{Person} (Student)

Person		
Rno	Name	CPI
101	Raj	8
102	Meet	9
103	Suresh	7

Example

Student		
Rno	Name	CPI
101	Raj	8
102	Meet	9
103	Suresh	7

$$\rho_{\text{Person (RollNo, StudentName)}} (\Pi_{RNo,\ Name} (\text{Student}))$$

Person	
RollNo	StudentName
101	Raj
102	Meet
103	Suresh

Example

Rno	Name	CPI
101	Raj	8
102	Meet	9
103	Suresh	7

Find out maximum CPI from student table.

ρ_A (Student) X ρ_B (Student)

A.Rno	A.Name	A.CPI	B.Rno	B.Name	B.CPI
101	Raj	8	101	Raj	8
101	Raj	8	102	Meet	9
101	Raj	8	103	Suresh	7
102	Meet	9	101	Raj	8
102	Meet	9	102	Meet	9
102	Meet	9	103	Suresh	7
103	Suresh	7	101	Raj	8
103	Suresh	7	102	Meet	9
103	Suresh	7	103	Suresh	7

Cont..

$$\sigma_{A.CPI<B.CPI} (\rho_A (Student) X \rho_B (Student))$$

A.Rno	A.Name	A.CPI	B.Rno	B.Name	B.CPI
101	Raj	8	102	Meet	9
103	Suresh	7	101	Raj	8
103	Suresh	7	102	Meet	9

$$\prod_{A.CPI} (\sigma_{A.CPI<B.CPI} (\rho_A (Student) X \rho_B (Student)))$$

A.CPI
8
7

Cont..

$$\prod_{CPI}(Student) - \prod_{A.CPI}(\sigma_{A.CPI<B.CPI}(\rho_A(Student) \times \rho_B(Student)))$$

CPI
8
9
7

—

A.CPI
8
7

=

CPI
9

Set Operators

- Set operators combine the results of two or more queries into a single result.
- **Condition to perform set operation:**
- Both relations (queries) must be union compatible :

- Relations R and S are union compatible, if
- ✓ Both queries should have same (equal) number of columns, and
- ✓ Corresponding attributes should have the same data type.

- **Types of set operators:**
- 1. Union
- 2. Intersect (Intersection)
- 3. Minus (Set Difference)

Union operator

- **Operation:** Selects tuples those are in either or both of the relations.
- **Symbol :** U (Union)
- **Notation :** Relation1 U Relation2

Customer

Name
Raj
Suresh
Meet

Employee

Name
Meet
Suresh
Manoj

Customer U Employee

Name
Manoj
Meet
Raj
Suresh

Union removes duplicate records.

Union example

Display **Name** of person who are **either employee or customer.**

Employee

ID	Name	Salary
2	Meet	15000
3	Jay	20000

Customer

ID	Name	Balance
1	Raj	5000
2	Meet	8000

$$\Pi_{Name}(Employee) \cup \Pi_{Name}(Customer)$$

Output

Name
Meet
Jay
Raj

Intersection operator

- **Operation:** Selects tuples those are common in both relations.
- **Symbol :** ∩ (Intersection)
- **Notation :** Relation1 ∩ Relation2

Customer

Name
Raj
Suresh
Meet

Employee

Name
Meet
Suresh
Manoj

Customer ∩ Employee

Name
Meet
Suresh

Intersect Example

- Display **Name** of person who are **employee as well as customer.**

Employee

ID	Name	Salary
2	Meet	15000
3	Jay	20000

Customer

ID	Name	Balance
1	Raj	5000
2	Meet	8000

$$\Pi_{Name}(Employee) \cap \Pi_{Name}(Customer)$$

Output

Name
Meet

The Set-Difference (Minus) Operation

- **Operation:** Selects tuples those are in first (left) relation but not in second (right) relation.
- **Symbol :** — (Minus)
- **Notation :** Relation1 — Relation2

Customer

Name
Raj
Suresh
Meet

Employee

Name
Meet
Suresh
Manoj

Customer – Employee

Name
Raj

Employee – Customer

Name
Manoj

The Set-Difference (Minus) Operation Example

- Display **Name** of person who are **employee but not customer**.

Employee

ID	Name	Salary
2	Meet	15000
3	Jay	20000

Customer

ID	Name	Balance
1	Raj	5000
2	Meet	8000

$$\Pi_{Name}(Employee) - \Pi_{Name}(Customer)$$

Output

Name
Jay

Condition to perform Set operators

- Set operators will take two or more queries as input, which must be union - compatible :

1. Both queries should have same (equal) number of columns

RollNo	Name	Branch	SPI
101	Raj	CE	8
102	Meet	CE	7
103	Neel	ME	9

EmpNo	Name	Branch
101	Patel	CE
102	Shah	CE
103	Ghosh	EE

RollNo	Name	Branch	SPI
101	Raj	CE	8
102	Meet	CE	7
103	Neel	ME	9

EmpNo	Name	Branch	Exp
101	Raj	CE	8
102	Meet	CE	1
103	Ghosh	EE	9

Condition to perform Set operators

- Set operators will take two or more queries as input, which must be union - compatible:

2. Corresponding attributes should have the same data type

RollNo	Name	Branch	SPI
101	Raj	CE	8
102	Meet	CE	7
103	Neel	ME	9

EmpNo	Name	Branch	Subject
101	Raj	CE	DBMS
102	Meet	CE	DS
103	Ghosh	EE	EEM

RollNo	Name	Branch	SPI
101	Raj	CE	8
102	Meet	CE	7
103	Neel	ME	9

EmpNo	Name	Branch	Exp
101	Raj	CE	8
102	Meet	CE	1
103	Ghosh	EE	9

Exercise

1. Check whether following tables are compatible or not:
   - A: (First_name(char), Last_name(char), Date_of_Birth(date))
   - B: (FName(char), LName(char), PhoneNumber(number))
   - (Not compatible) Both tables have 3 attributes but **third attributes datatype is different**.

   - A: (First_name(char), Last_name(char), Date_of_Birth(date))
   - B: (FName(char), LName(char), DOB(date))
   - ✓ (Compatible) Both tables have 3 attributes and of same data type.

The Cartesian-Product (Cross product) Operation

- **Operation:** Combines information of two relations. It will multiply each tuples of first relation to each tuples of second relation.
- **Symbol:** X (Cross)
- **Notation:** Relation1 X Relation2
- **Resultant Relation :**
- If both relations have some attribute having same name, it can be distinguished by combing relation- name. attribute-name.

  - **Attributes of Resultant Relation =** Attributes of R1 + Attributes of R2
  - **Tuples of Resultant Relation =** Tuples of R1 * Tuples of R2

Example:

Student

RollNo	Name	Branch
101	Raj	CE
102	Meet	ME

Result

RollNo	SPI
101	8
103	9

Student × Result

Student.RollNo	Name	Branch	Result.RollNo	SPI

If both relations have some attribute with the same name, it can be distinguished by combing **relation-name.attribute-name.**

Join operations : *Natural Join Operation (⋈)*

- **Operation:** Natural join will retrieve information from multiple relations. It works in three steps.
- 1. It performs Cartesian product
- 2. Then it finds consistent tuples and inconsistent tuples are deleted
- 3. Then it deletes duplicate attributes
- **Notation:** Relation1 ⋈ Relation2
- To perform a natural join there must be **one common attribute (column)** between two relations.

Example

Student

RollNo	Name	Branch
101	Raj	CE
102	Meet	ME

Result

RollNo	SPI
101	8
103	9

Student × Result

Step 1 : Performs Cartesian Product

Student.RollNo	Name	Branch	Result.RollNo	SPI
101	Raj	CE	101	8
101	Raj	CE	103	9
102	Meet	ME	101	8
102	Meet	ME	103	9

Student ⋈ Result

Step 2 : Removes inconsistent tuples

Student.RollNo	Name	Branch	Result.RollNo	SPI
101	Raj	CE	101	8

Step 3 : Removes an attribute

Student ⋈ Result

RollNo	Name	Branch	SPI
101	Raj	CE	8

Example

Write down relational algebras for the following tables

- ***Relations***
  - *Student (Rno, Sname, Address, City, Mobile)*
  - *Department (Did, Dname)*
  - *Academic (Rno, Did, SPI, CPI, Backlog)*
  - *Guide (Rno, PName, Fid)*
  - *Faculty (Fid, Fname, Subject, Did, Salary)*
- List the **name of students** with their **department name** and **SPI** of all student belong to **"CE" department.**

$$\Pi_{Sname,\ Dname,\ SPI}\ (\sigma_{Dname='CE'}\ (Student \bowtie (Department \bowtie Academic)))$$

Example

Write down relational algebras for the following tables

- ***Relations***
  - *Student (Rno, Sname, Address, City, Mobile)*
  - *Department (Did, Dname)*
  - *Academic (Rno, Did, SPI, CPI, Backlog)*
  - *Guide (Rno, PName, Fid)*
  - *Faculty (Fid, Fname, Subject, Did, Salary)*
- Display the **name of students** with their **project name** whose **guide is "A. J. Shah"**.

$$\Pi_{Sname,\ Pname}\ (\sigma_{Fname='A.J.Shah'}\ (Student \bowtie (Guide \bowtie Faculty)))$$

The Outer Join Operation

- In natural join some records are missing if we want that missing records than we have to use outer join.
- The outer join operation can be divided into three different forms:

1. Left outer join (⟕)
2. Right outer join (⟖)
3. Full outer join (⟗)

Left outer join (⟕)

- The left outer join returns all the tuples of the left relation even through there is no matching tuple in the right relation.
- For such kind of tuples having no matching, the attributes of right relation will be padded with null in resultant relation.

Student

RollNo	Name	Branch
101	Raj	CE
102	Meet	ME

Result

RollNo	SPI
101	8
103	9

Student ⟕ Result

RollNo	Name	Branch	SPI
101	Raj	CE	8
102	Meet	ME	**null**

Example

- Display **Name, Salary and Balance** of the of all **employees.**

Employee

ID	Name	Salary
2	Meet	15000
3	Jay	20000

Customer

ID	Name	Balance
1	Raj	5000
2	Meet	8000

$$\Pi_{Name,\ Salary,\ Balance}\ (Employee ⟕ Customer)$$

Output

Name	Salary	Balance
Meet	15000	8000
Jay	20000	**null**

Right outer join (⟖)

- The right outer join returns all the tuples of the right relation even though there is no matching tuple in the left relation.
- For such kind of tuples having no matching, the attributes of left relation will be padded with null in resultant relation.

Student

RollNo	Name	Branch
101	Raj	CE
102	Meet	ME

Result

RollNo	SPI
101	8
103	9

Student ⟖ Result

RollNo	Name	Branch	SPI
101	Raj	CE	8
103	**null**	**null**	9

Example

- Display **Name, Salary and Balance** of the of all **customers.**

Employee

ID	Name	Salary
2	Meet	15000
3	Jay	20000

Customer

ID	Name	Balance
1	Raj	5000
2	Meet	8000

$$\Pi_{Name,\ Salary,\ Balance}\ (Employee \rtimes Customer)$$

Output

Name	Salary	Balance
Raj	**null**	5000
Meet	15000	8000

Full outer join (⟗)

- The full outer join returns all the tuples of both of the relations. It also pads null values whenever required.

Student

RollNo	Name	Branch
101	Raj	CE
102	Meet	ME

Result

RollNo	SPI
101	8
103	9

Student ⟗ Result

RollNo	Name	Branch	SPI
101	Raj	CE	8
102	Meet	ME	**null**
103	**null**	**null**	9

Example

- Display **Name, Salary and Balance** of the of all **employee** as well as **customers.**

Employee

ID	Name	Salary
2	Meet	15000
3	Jay	20000

Customer

ID	Name	Balance
1	Raj	5000
2	Meet	8000

$$\Pi_{\textit{Name, Salary, Balance}} (\textit{Employee} ⟗ \textit{Customer})$$

Output

Name	Salary	Balance
Meet	15000	8000
Jay	20000	**null**
Raj	**null**	5000

Aggregate Function

- **Operation:** It takes a more than one value as input and returns a single value as output (result).
- **Symbol:** G
- **Notation:** G $_{\text{function (attribute)}}$ (relation)
- **Aggregate function**
  1. Sum (It **returns the sum (addition)** of the values of a column.)
  2. Max (It **returns the maximum** value for a column.)
  3. Min (It **returns the minimum** value for a column.)
  4. Avg (It **returns the average** of the values for a column.)
  5. Count (It **returns total number** of values in a given column.)

Aggregate Function Example

Student

Rno	Name	Branch	Semester	CPI
101	Ramesh	CE	3	9
102	Mahesh	EC	3	8
103	Suresh	ME	4	7
104	Amit	EE	4	8
105	Anita	CE	4	8
106	Reeta	ME	3	7
107	Rohit	EE	4	9
108	Chetan	CE	3	8
109	Rakesh	CE	4	9

Example: Find out **sum of CPI** of all students.

- $G_{\text{sum(CPI)}}$ (Student)

sum
73

Aggregate Function Example

Student

Rno	Name	Branch	Semester	CPI
101	Ramesh	CE	3	9
102	Mahesh	EC	3	8
103	Suresh	ME	4	7
104	Amit	EE	4	8
105	Anita	CE	4	8
106	Reeta	ME	3	7
107	Rohit	EE	4	9
108	Chetan	CE	3	8
109	Rakesh	CE	4	9

- **Example:** Find out **maximum & minimum CPI.**
  - G $_{\text{max(CPI), min(CPI)}}$ (Student)

max	min
9	7

Aggregate Function Example

Student

Rno	Name	Branch	Semester	CPI
101	Ramesh	CE	3	9
102	Mahesh	EC	3	8
103	Suresh	ME	4	7
104	Amit	EE	4	8
105	Anita	CE	4	8
106	Reeta	ME	3	7
107	Rohit	EE	4	9
108	Chetan	CE	3	8
109	Rakesh	CE	4	9

- **Example: Count** the number of students.
  - $G_{\text{count(Rno)}}$ (Student)

count
9

Exercise

1. Write down relational algebras for the following table:

   Employee (person-name, street, city)

   Works (person-name, company-name, salary)

   Company (company-name, city)

   Managers (person-name, manager-name)

   i. Find the **names** of all **employees** who **work for "TCS"**.
   ii. Find the **names** and **cities** of residence of all **employees** who **work for "Infosys"**.
   iii. Find the **names**, **street** and **city** of residence of all **employees** who **work for "ITC"** and **earn more than $10,000** per annum.
   iv. Find the **names** of all **employees** in this database who **live in the same city** as the **company** for which they **work.**

Exercise

1. Write down relational algebras for the following table:

   Employee (person-name, street, city)

   Works (person-name, company-name, salary)

   Company (company-name, city)

   Managers (person-name, manager-name)

   v. Find the **names** of all **employees working in "TCS"** who **earn more than 25000** and **less than 40000.**

   vi. Find the **name** of **employee** whose **manager is "Ajay Patel"** and **salary is more than 50000.**

   vii. Display the **name** of **employee** with **street**, **city**, **company name**, **salary** and **manager name staying** in "**Rajkot**" and **working** in "**Ahmedabad**".

   viii. Find **maximum**, **minimum** and **average salary** of all employee.

   ix. Find out the **total number** of **employee.**

Exercise

Consider the following relational database, where the primary keys are underlined. Give an expression in the relational algebra to express each of the following queries

employee (ssn, name, dno, salary, hobby, gender)
department (dno, dname, budget, location, mgrssn)
works_on (ssn, pno)
project (pno, pname, budget, location, goal)

1. List all pairs of employee names and the project numbers they work on.
2. List out department number, department name and department budget.
3. List all projects that Raj Yadav works on by project name.
4. List the names of employees who supervise themselves.

Exercise

Consider the following relational database, where the primary keys are underlined. Give an expression in the relational algebra to express each of the following queries

course (course-id, title, dept_name, credits)
instructor (id, name, dept_name, salary)
section (course-id, sec-id, semester, year, building, room_no, time_slot_id)
teaches (id, course-id, sec-id, semester, year)

1. Find the name of all instructors in the physics department.
2. Find all the courses taught in the fall 2009 semester but not in Spring semester.
3. Find the names of all instructors in the Comp. Sci. department together with the course titles of all the courses that the instructors teach.
4. Find the average salary in each department.

Tuple Relational calculus

- The tuple relational calculus is a **nonprocedural** query language. It describes the desired information without giving a specific procedure for obtaining that information.
- A query in the tuple relational calculus is expressed as:

$$\{t \mid P(t)\}$$

"set of all tuples *t* such that predicate *P* is true for *t*"

$$\exists\ t \in r\ (Q(t))$$

"there exists a tuple *t* in relation *r* such that predicate *Q*(*t*) is true"

- Student(rollno,name,dept,gender)
- 1) find out all details of students
- Sql:select * from student

TRC:{t / student(t)}

2)Display all details of students whose dept no is 120

TRC: {t/ student(t)

3) Find details of male students whose dept no is 100

4)Find roll numbers of all the students.

Example Queries

- Find the *ID*, *name*, *dept name*, *salary* for instructors whose salary is greater than \$80,000:

$$\{t \mid t \in instructor \land t[salary] > 80000\}$$

- Find the instructor *ID* for each instructor with a salary greater than \$80,000

$$\{t \mid \exists s \in instructor\ (t[ID] = s[ID] \land s[salary] > 80000)\}$$

"The set of all tuples *t* such that there exists a tuple *s* in relation *instructor* for which the values of *t* and *s* for the *ID* attribute are equal, and the value of *s* for the *salary* attribute is greater than \$80,000."

Domain Relational calculus

- **It** uses *domain* variables that take on values from an attributes domain, rather than values for an entire tuple.
- An expression in the domain relational calculus is of the form

$$\{< x1, x2, \ldots, xn > \mid P(x1, x2, \ldots, xn)\}$$

where $x1, x2, \ldots, xn$ represent domain variables.

Example Queries

- Find the instructor *ID*, *name*, *dept name*, and*salary* for instructors whose salary is greater than $80,000:

$$\{< i, n, d, s > \mid < i, n, d, s > \in instructor \wedge s > 80000\}$$

Expressive Power of Languages

- All three of the following are equivalent:

- The basic relational algebra (without the extended relational-algebra operations such as generalized projection and aggregation (*G*))
- The tuple relational calculus restricted to safe expressions
- The domain relational calculus restricted to safe expressions

SQL

- The SQL language has several parts:

- **Data-definition language** (DDL). The SQL DDL provides commands for defining relation schemas, deleting relations, and modifying relation schemas.

- **Data-manipulation language** (DML). The SQL DML provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.

- **Integrity**. The SQL DDL includes commands for specifying integrity constraints that the data stored in the database must satisfy. Updates that violate integrity constraints are disallowed.

- **View definition**. The SQL DDL includes commands for defining views.

- **Authorization**: The SQL DDL includes commands for specifying access rights to relations and views.

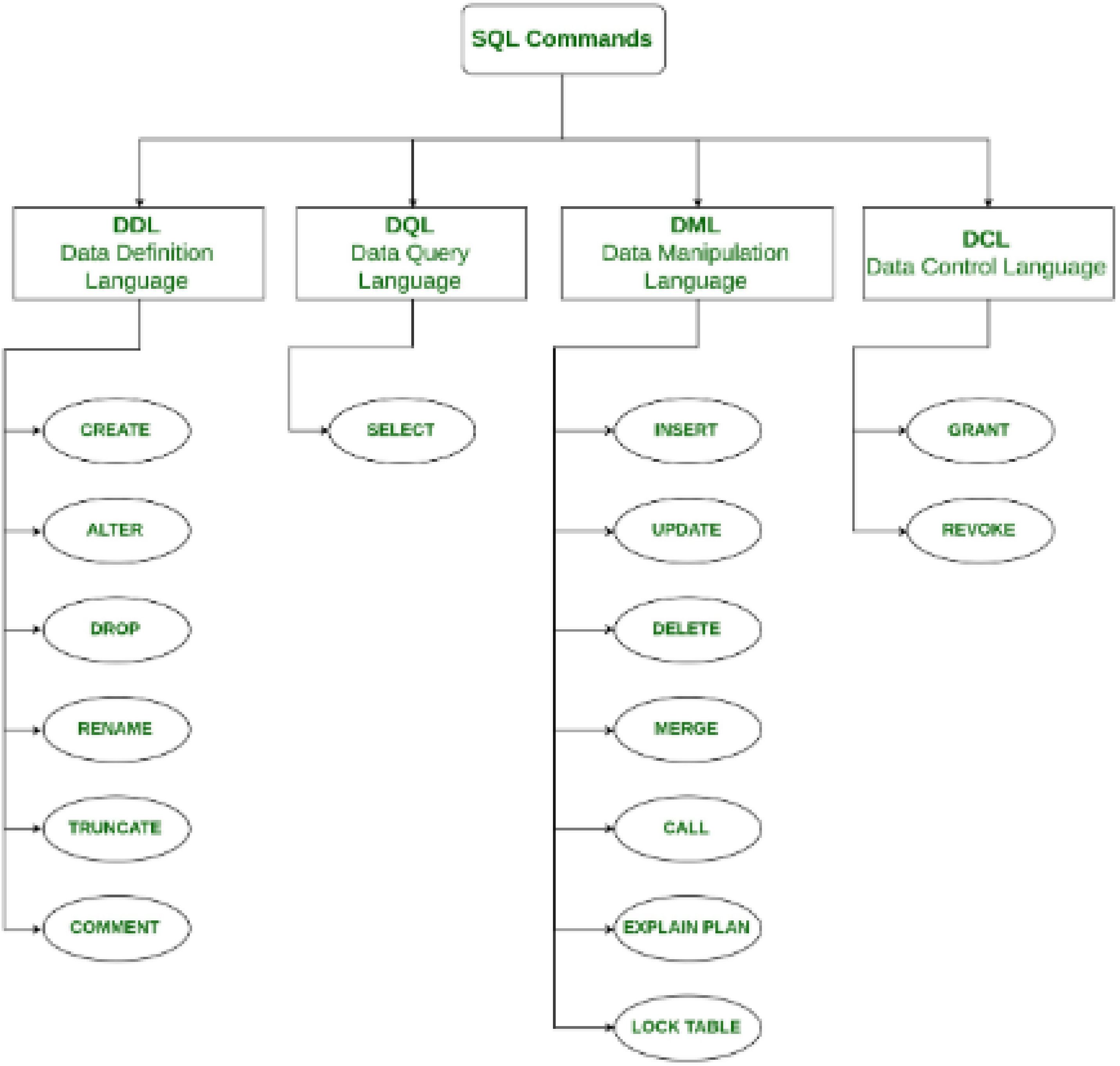
- **Embedded SQL** and **dynamic SQL**: define how SQL statements can be embedded within general-purpose programming languages, such as C, C++, and Java.

- **Transaction control**. SQL includes commands for specifying the beginning and ending of transactions.

sql

- These SQL commands are mainly categorized into four categories as:

- DDL – Data Definition Language
- DQl – Data Query Language
- DML – Data Manipulation Language
- DCL – Data Control Language


SQL Commands
DDL
Data Definition
Language
DQL
Data Query
Language
DML
Data Manipulation
Language
DCL
Data Control Language
CREATE
ALTER
DROP
RENAME
TRUNCATE
COMMENT
SELECT
INSERT
UPDATE
DELETE
MERGE
CALL
EXPLAIN PLAN
LOCK TABLE
GRANT
REVOKE

DDL constructs

- The SQL DDL allows specification of a set of relations and information about each relation, including:
- The schema for each relation.
- The types of values associated with each attribute.
- The integrity constraints.
- The set of indices to be maintained for each relation.
- The security and authorization information for each relation.
- The physical storage structure of each relation on disk.

Cont..

- It provides commands like:

✓CREATE: to create objects in a database.

✓ALTER: to alter the schema, or logical structure of the database.

✓DROP: to delete objects from the database.

✓TRUNCATE: to remove all records from the table.

Create Table

- The CREATE TABLE statement is used to create a new table in a database.
- **Syntax:**

```
CREATE TABLE table_name(
Column1 Datatype(Size) [ NULL | NOT NULL ],
Column2 Datatype(Size) [ NULL | NOT NULL ], ... );
```

- **Example:**

```
CREATE TABLE Students(
Roll_No int(3) NOT NULL,
Name varchar(20),
Subject varchar(20) );
```

Basic Types

- The SQL standard supports a variety of built-in types, including:

- **char**(*n*): A fixed-length character string with user-specified length *n*. The full form, **character**, can be used instead.
- **varchar**(*n*): A variable-length character string with user-specified maximum length *n*. The full form, **character varying**, is equivalent.
- **int**: An integer (a finite subset of the integers that is machine dependent). The full form, **integer**, is equivalent.
- **smallint**: A small integer (a machine-dependent subset of the integer type).

Basic Types

• **numeric**(*p, d*):Afixed-point numberwith user-specified precision. The number consists of *p* digits (plus a sign), and *d* of the *p* digits are to the right of the decimal point. Thus, **numeric**(3,1) allows 44.5 to be stored exactly, but neither 444.5 or 0.32 can be stored exactly in a field of this type.

• **real, double precision**: Floating-point and double-precision floating-point numbers with machine-dependent precision.

• **float**(*n*): A floating-point number, with precision of at least *n* digits.

- Each type may include a special value called the **null** value. A null value indicates an absent value that may exist but be unknown or that may not exist at all.

Basic Types

- The **char** data type stores fixed length strings. Consider, for example, an attribute *A* of type **char**(10). If we store a string “Avi” in this attribute, 7 spaces are appended to the string to make it 10 characters long. In contrast, if attribute *B* were of type **varchar**(10), and we store “Avi” in attribute *B*, no spaces would be added.

- When comparing a **char** type with a **varchar** type, even if the same value “Avi” is stored in the attributes *A* and *B* above, a comparison *A*=*B* may return false. We recommend you always use the **varchar** type instead of the **char** type to avoid these problems.

ALTER

- ALTER TABLE statement is used to add, modify, or drop columns in a table.

- **Add Column:** The ALTER TABLE statement in SQL to add new columns in a table.
- **Syntax:**

ALTER TABLE table_name ADD Column1 Datatype(Size), Column2 Datatype(Size), … ;

- **Example:**

ALTER TABLE Students ADD Marks int;

ALTER

- **Drop Column:** The ALTER TABLE statement in SQL to drop a column in a table.
- **Syntax:**

  ALTER TABLE table_name DROP COLUMN column_name;

- **Example:**

  ALTER TABLE Students DROP COLUMN Subject;

- **Modify Column :** The ALTER TABLE statement in SQL to change the data type/size of a column in a table.
- **Syntax:**

  ALTER TABLE table_name ALTER COLUMN column_name datatype(size);

- **Example:**

  ALTER TABLE Students ALTER COLUMN Roll_No float;

DROP

- Drop is used to drop the database or its objects like table, view, index etc.
- **Drop Table:** The DROP TABLE statement is used to drop an existing table in a database.
- **Syntax:**

```
DROP TABLE table_name;
```

- **Example:**

```
DROP TABLE Students;
```

TRUNCATE

- Truncate is used to remove all records from the table.
- **Syntax:**

TRUNCATE TABLE table_name;

- **Example:**

TRUNCATE TABLE Students;

DML constructs

- It is a set of SQL commands used to insert, modify and delete data in a database.
- It is normally used by general users who are accessing database via pre-developed
- applications.
- It provides commands like:
  - INSERT: to insert data into a table.
  - UPDATE: to modify existing data in a table.
  - DELETE: to delete records from a table.

DQL constructs

- *DQL (Data Query Language)*
- It is a component of SQL that allows data retrieval from the database.
- It provides command like SELECT. This command is a heart of SQL, and allows data retrieval in different ways.

- **SELECT:** The SELECT statement is used to select data from a database. The data returned is stored in a result table, called the result-set.

- **Syntax:**
- SELECT column1, column2, ...
- FROM table_name
- WHERE condition;

DCL

- *DCL (Data Control Language)*
- It is set of SQL commands used to control access to data and database. Occasionally DCL
- commands are grouped with DML commands.

- It provides commands like:
  - GRANT: to give access privileges to users on the database.
  - REVOKE: to withdraw access privileges given to users on the database.

TCL

- *TCL (Transaction Control Language)*
- TCL is abbreviation of Transactional Control Language. It is used to manage different transactions occurring within a database.

- COMMIT – Saves work done in transactions
- ROLLBACK – Restores database to original state since the last COMMIT command in transactions
- SAVE TRANSACTION – Sets a save point within a transaction.

Integrity constraints

- SQL supports a number of different integrity constraints. For example
- **primary key** (*Aj*1 *, Aj*2*, . . . , Ajm*): The **primary-key** specification says that attributes *Aj*1 *, Aj*2*, . . . , Ajm* form the primary key for the relation. The primarykey attributes are required to be *nonnull* and *unique*

- **foreign key** (*Ak*1 *, Ak*2*, . . . , Akn*) **references** *s*: The**foreign key** specification says that the values of attributes (*Ak*1 *, Ak*2*, . . . , Akn*) for any tuple in the relation must correspond to values of the primary key attributes of some tuple in relation *s*.

Open source and Commercial DBMS -MYSQL, ORACLE, DB2, SQL server

Unit 2

Part-II
Relational database design

-

Prof Vaibhavi Patel

Topics

- Domain and data dependency
- Armstrong's axioms
- Lossless design
- Normal forms

Domain and data dependency

- Dependencies in DBMS is a relation between two or more attributes. It has the following types in DBMS:

- Functional Dependency
- Fully-Functional Dependency
- Transitive Dependency
- Multivalued Dependency
- Partial Dependency

Functional Dependency

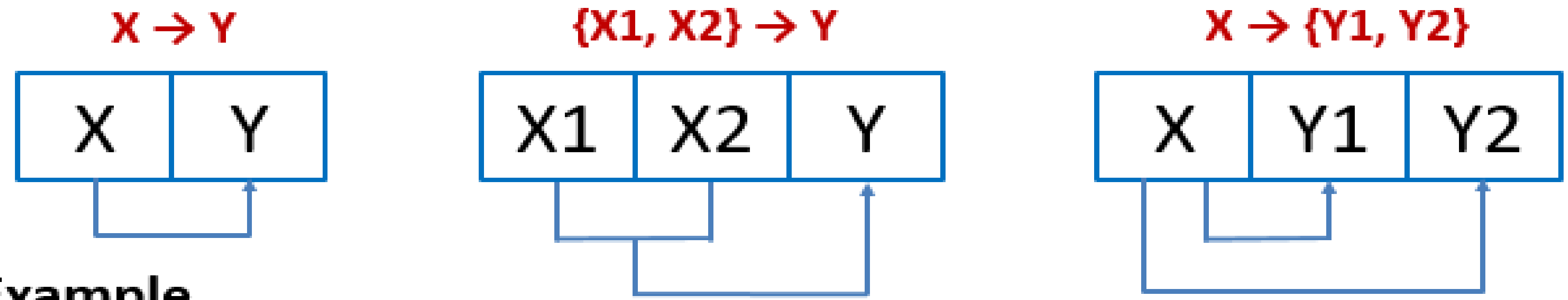
- Let R be a relation schema having n attributes A1, A2, A3,…, An.
- Let attributes X and Y are two subsets of attributes of relation R.
- If the values of the X component of a tuple uniquely (or functionally) determine the values of the Y component, then, there is a functional dependency from X to Y.

- This is denoted by $X \rightarrow Y$.

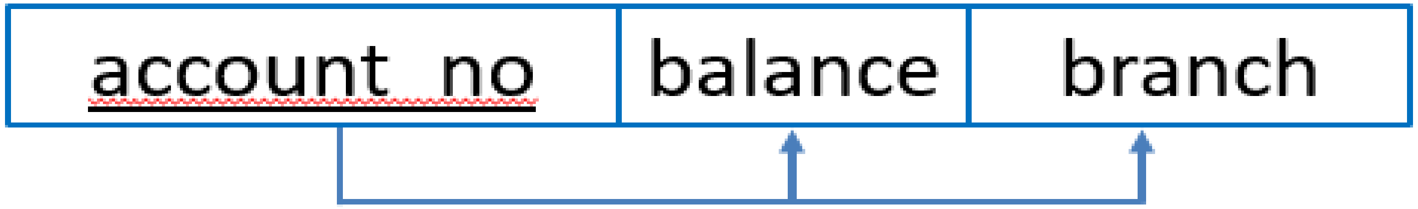
It is referred as: Y is functionally dependent on the X, or X functionally determines Y.

- The left hand side of the FD is also referred as determinant whereas the right hand side of the FD is referred as dependent.

Diagrammatic representation



- **Example**
- Consider the relation Account(account_no, balance, branch).
- **account_no** can **determine balance** and **branch**.
- So, there is a functional dependency from **account_no** to **balance** and **branch**.
- This can be denoted by **account_no → {balance, branch}**.



Types of Functional Dependencies

➢ *Full Dependency*

- In a relation, the attribute B is fully functional dependent on A if B is functionally dependent on A, but not on any proper subset of A.
- Eg. {Roll_No, Department_Name, Semester} →SPI
- We need all three {Roll_No, Department_Name, Semester} to find SPI

Types of Functional Dependencies

➢ *Partial Dependency*

- In a relation, the attribute B is partial functional dependent on A if B is functionally dependent on A as well as on any proper subset of A.
- If there is some attribute that can be removed from A and the still dependency holds.
- Eg. {Enrollment_No, Department_Name} → SPI
- Enrollment_no is sufficient to find SPI, Department_name is not required.

Types of Functional Dependencies

- ***Transitive Dependency***

- In a relation, if attribute(s) A→B and B→C, then C is transitively depends on A via B .
    - Eg. Staff_No → Branch_No and Branch_No → Branch_Address

- ***Trivial FD: It is always valid fd***

- X→Y is trivial FD if Y is a subset of X
    - Eg.{Roll_No, Department_Name} → Roll_No

- ***Nontrivial FD***

- X→Y is nontrivial FD if Y is not a subset of X
    - Eg.. {Roll_No, Department_Name} → Student_Name

Armstrong's axioms (inference rules)

- Armstrong's axioms are a set of rules used to infer (derive) all the functional dependencies on a relational database.

- Let A, B, and C is subsets of the attributes of the relation R.

- *Reflexivity :* If B is a subset of A then A → B
- *Augmentation:* If A → B then AC → BC
- *Transitivity:* If A → B and B → C then A → C

Armstrong's axioms (inference rules)

1. Reflexivity
   - If B is a subset of A
     then A → B
2. Augmentation
   - If A → B
     then AC → BC
3. Transitivity
   - If A → B and B → C
     then A → C
4. Pseudo Transitivity
   - If A → B and BD → C
     then AD → C
5. Self-determination
   - A → A
6. Decomposition
   - If A → BC
     then A → B and A → C
7. Union
   - If A → B and A → C
     then A → BC
8. Composition
   - If A → B and C → D
     then AC → BD

closure of a set of FDs

- Given a set F set of functional dependencies, there are certain other functional dependencies that are logically implied by F.
- E.g.: F = {A → B and B → C}, then we can infer that A → C
- The set of functional dependencies (FDs) that is logically implied by F is called the closure of F.
- It is denoted by F+.

Example-1

- **Suppose a relation R is given with attributes A, B, C, G, H and I. Also, a set of functional dependencies F is given with following FDs.**
- **F = {A → B, A → C, CG → H, CG → I, B → H}**
- **Closure of F.**

We have	Using rule	Derived FD
A → B and B → H	Transitivity rule	A → H
CG → H and CG → I	Union rule	CG → HI
A → C and CG → I	Pseudo-transitivity rule	AG → I
A → C and CG → H	Pseudo-transitivity rule	AG → H

F^+ = { A → H, CG → HI, AG → I, AG → H }

Example-2

- **Compute the closure of the following set F of functional dependencies for relational schema**
- **R = (A, B, C, D, E, F):**
- **F = (A → B, A → C, CD → E, CD → F, B → E).**

We have	Using rule	Derived FD
A → B and A → C	Union rule	A → BC
CD → E and CD → F	Union rule	CD → EF
A → B and B → E	Transitivity rule	A → E
A → C and CD → E	Pseudo- Transitivity rule	AD → E
A → C and CD → F	Pseudo- Transitivity rule	AD → F

- F^+ = { A → BC, CD → EF, A → E, AD → E, AD → F }

Example-3

- **Compute the closure of the following set F of functional dependencies for relational schema**
- **R = (A, B, C, D, E):**
- **F = (AB → C, D → AC, D → E).**

We have	**Using rule**	**Derived FD**
D → AC	Decomposition rule	D → A and D → C
D → AC and D → E	Union rule	D → ACE

F^+ = { D → A, D → C, D → ACE }

Closure of a set of attributes

- *Closure of set of attributes*
- Given a set of attributes α, the closure of α under F is the set of attributes that are
- functionally determined by α under F.
- OR IT CONTAINS SET OF ATTRIBUTES DETERMINE BY α.
- It is denoted by α+.

- **Given relation R with attributes A,B, C,D,E,F and set of FDs as**
- **A → BC, E → CF, B → E and CD → EF.**
- **Find out closure {A, B}+ of the set of attributes**

- Ans: Closure of {A, B}+ is {A, B, C, E, F}.

Exercise

- Given functional dependencies (FDs) for relational schema **R = (A,B,C,D,E)**:

  $F = \{A \rightarrow BC,\ CD \rightarrow E,\ B \rightarrow D,\ E \rightarrow A\}$

  1. Find Closure for A
  2. Find Closure for CD
  3. Find Closure for B
  4. Find Closure for BC
  5. Find Closure for E

$A^+ = ABCDE$

$CD^+ = ABCDE$

$B^+ = BD$

$BC^+ = ABCDE$

$E^+ = ABCDE$

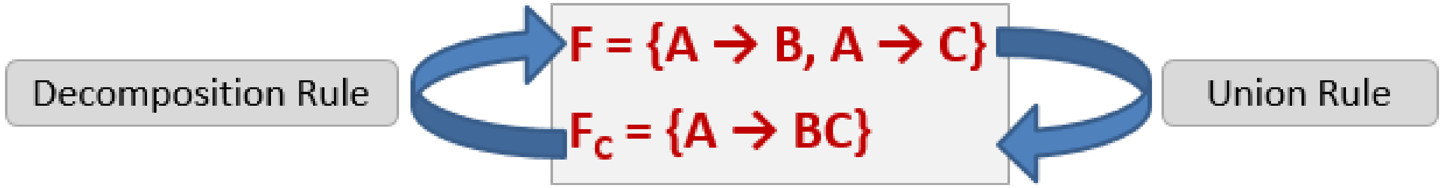
Exercise

- R(A,B,C,D,E)
- FD { A->B, B->C, C->D,D->E}
- FIND CLOSUER SET OR ATTRIBUTE CLOSURE OF
- 1)A 2) B 3) C 4) D 5) E 6) AB 7) ABCD 8) ACD

Canonical cover

- A canonical cover of F is a minimal set of functional dependencies equivalent to F, having no redundant dependencies or redundant parts of dependencies.
- It is denoted by Fc.

- A canonical cover for F is a set of dependencies Fc such that
  - 1) F logically implies all dependencies in Fc, and
  - 2) Fc logically implies all dependencies in F, and
  - 3) No functional dependency in Fc contains an extraneous attribute, and
  - 4) Each left side of functional dependency in Fc is unique.



STEPS TO FIND CANONICAL COVER FORM

- 1. Simplify all RHS (Decomposition)
- 2. For all FDs on LHS find For all FDs on LHS find a redundant redundant (extraneous extraneous) attribute
- 3. Eliminate all redundant FDs
- 4. Apply Union if needed
- 5. The result is Fc
- **extraneous attributes :If we are able to remove an attribute from a functional dependency without changing the closure of the set of functional dependencies, that attribute is called as Extraneous Attribute**

Example

- **Consider following set F of functional dependencies on schema R(A,B,C) and compute canonical cover for F.**
- **A -> BC, B -> C, A -> B and AB -> C**
- Ans: canonical cover is: A → B, B → C
- 2)R{A,B,C}

  FD={A->BC,B->C,A->B,AB->C}

3) R = {A,B,C,D,E,F,G,H}

   F = {AC →G, D →EG, BC →D, CG →BD, ACD →B, CE →AG} Find the canonical cover of F.

* Different order of considering the extraneous attributes can result in different FC

Decomposition

- Decomposition is the process of breaking down given relation into two or more relations.
- Here, relation R is replaced by two or more relations in such a way that -
- 1. Each new relation contains a subset of the attributes of R, and
- 2. Together, they all include all tuples and attributes of R.

- There are two types of decomposition
- 1. lossy decomposition
- 2. lossless decomposition (non-loss decomposition)

Lossy Decomposition

- The decomposition of relation R into R1 and R2 is lossy when the join of R1 and R2 does not yield the same relation as in R.
- This is also referred as lossy-join decomposition.

- The **disadvantage** of such kind of decomposition is that some information is lost during retrieval of original relation.
- From practical point of view, decomposition should not be lossy decomposition.

Lossy Decomposition

Ano	Balance	Bname
A01	5000	Rajkot
A02	5000	Surat

Ano	Balance
A01	5000
A02	5000

Balance	Bname
5000	Rajkot
5000	Surat

Not Same

Ano	Balance	Bname
A01	5000	Rajkot
A01	5000	Surat
A02	5000	Rajkot
A02	5000	Surat

Extra records

Here, it is not possible to specify that in which branch account A01 or A02 belongs

Lossless (Non-loss) Decomposition

- The decomposition of relation R into R1 and R2 is lossless when the join of R1 and R2 produces the same relation as in R.
- This is also referred as a non-additive (non-loss) decomposition.
- All decompositions must be lossless.

Lossless (Non-loss) Decomposition

Ano	Balance	Bname
A01	5000	Rajkot
A02	5000	Surat

Ano	Balance
A01	5000
A02	5000

Ano	Bname
A01	Rajkot
A02	Surat

Same

Ano	Balance	Bname
A01	5000	Rajkot
A02	5000	Surat

Same as original table
No extra records

- **To check for lossless join decomposition using FD set, following conditions must hold:**
- Union of Attributes of R1 and R2 must be equal to attribute of R. Each attribute of R must be either in R1 or in R2.

  Att(R1) U Att(R2) = Att(R)

- Intersection of Attributes of R1 and R2 must not be NULL.

  Att(R1) ∩ Att(R2) ≠ Φ

- Common attribute must be a key for at least one relation (R1 or R2)

  Att(R1) ∩ Att(R2) -> Att(R1) or Att(R1) ∩ Att(R2) -> Att(R2)

- For Example, A relation R (A, B, C, D) with FD set{A->BC} is decomposed into R1(ABC) and R2(AD) which is a lossless join decomposition as:
- First condition holds true as Att(R1) U Att(R2) = (ABC) U (AD) = (ABCD) = Att(R).
- Second condition holds true as Att(R1) ∩ Att(R2) = (ABC) ∩ (AD) ≠ Φ
- Third condition holds true as Att(R1) ∩ Att(R2) = A is a key of R1(ABC) because A->BC is given.
-

- **Dependency Preserving Decomposition**
- If we decompose a relation R into relations R1 and R2, All dependencies of R either must be a part of R1 or R2 or must be derivable from combination of FD’s of R1 and R2.
- For Example, A relation R (A, B, C, D) with FD set{A->BC} is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of R1(ABC).

- **GATE Question: Consider a schema R(A,B,C,D) and functional dependencies A->B and C->D. Then the decomposition of R into R1(AB) and R2(CD) is [GATE-CS-2001]**
  A. dependency preserving and lossless join
  B. lossless join but not dependency preserving
  C. dependency preserving but not lossless join
  D. not dependency preserving and not lossless join

Anomaly in database design

- Anomalies are problems that can occur in poorly planned, un-normalized database where all the data are stored in one table.

- There are three types of anomalies that can arise in the database because of redundancy are
- Insert anomalies
- Delete anomalies
- Update / Modification anomalies

Insert anomaly

- Consider a relation

emp_dept (E#, Ename, Address, D#, Dname, Dmgr#) with E# as a primary key.

E#	Ename	Address	D#	Dname	Dmgr#
1	Raj	Rajkot	1	CE	1
2	Meet	Surat	1	CE	1

Want to insert new department detail (IT)

- Let us assume that a new department has been started by the organization but initially there is no employee appointed for that department
- Then the tuple for this department cannot be inserted in to this table as the E# will have NULL value, which is not allowed because E# is primary key.
- This kind of problem in the relation where some tuple cannot be inserted is known as insert anomaly.

Delete anomaly

- Consider a relation

emp_dept (E#, Ename, Address, D#, Dname, Dmgr#) with E# as a primary key.

Want to delete Meet employee's detail

E#	Ename	Address	D#	Dname	Dmgr#
1	Raj	Rajkot	1	CE	1
2	Meet	Surat	1	IT	2

- Now consider there is only one employee in some department and that employee leaves the organization
- Then the tuple of that employee has to be deleted from the table, but in addition to that information about the department also will be deleted.
- This kind of problem in the relation where deletion of some tuples can lead to loss of some other data not intended to be removed is known as delete anomaly.

Update / Modification anomaly

- Consider a relation

emp_dept (E#, Ename, Address, D#, Dname, Dmgr#) with E# as a primary key.

Want to update CE department's manager

E#	Ename	Address	D#	Dname	Dmgr#
1	Raj	Rajkot	1	CE	M1
2	Meet	Surat	2	IT	M2
3	Jay	Rajkot	2	C.E	M2

- Suppose the manager of a department has changed, this requires that the Dmgr# in all the tuples corresponding to that department must be changed to reflect the new status.
- If we fail to update all the tuples of given department, then two different records of employee working in the same department might show different Dmgr# lead to inconsistency in the database.
- This kind of problem is known as update or modification anomaly.

Anomaly (Summary)

EmpID	EmpName	Address	DeptID	DeptName	DeptMngr
E1	Raj	Rajkot	D1	C.E.	Patel
E2	Samir	Rajkot	D2	Civil	Shah
E3	Meet	Baroda	D1	Computer	Patel
E4	Deepak	Surat	D1	C.E	Patel
E5	Suresh	Surat	D3	Electrical	Joshi
null	null	null	D4	Chemical	null

Delete Anomaly
If we delete Employee having ID "E2" then Civil department will also delete because there is only one record of Civil dept.

Insert Anomaly
Do not allow to insert new Department "Chemical" until an employee is assign to it.

Update Anomaly
An update anomaly exists when one or more records of duplicated data is updated, but not all.

How anomalies in database design can be solved

- Such type of anomalies in database design can be solved by using **normalization**.

Normalization

Normalization

- Database normalization is the process of removing redundant data from your tables to improve storage efficiency, data integrity, and scalability.

1. storage efficiency: ability to store and manage data that consumes the least amount of data.
2. data integrity: completeness, accuracy and consistency of data.
3. Scalability: ability of a system to continue to function well in a growing amount of work.

- *Need of Normalization*

- Eliminates redundant data
- Reduces chances of data errors
- Reduces disk space
- Improve data integrity, scalability and data consistency.

- *What we do in normalization?*

- Normalization generally involves splitting existing tables into multiple ones, which must be re-joined or linked each time a query is issued.

Normal forms

1. 1NF (First normal form)
2. 2NF (Second normal form)
3. 3NF (Third normal form)
4. BCNF (Boyce–Codd normal form)
5. 4NF (Forth normal form)
6. 5NF (Fifth normal form)

As we **move from 1NF to 5NF number of tables** and **complexity increases** but **redundancy decreases**.

First normal form (1NF)

Conditions for 1NF

Each **cells of a table should contain a single value.**

- A relation R is in first normal form (1NF) if and only if it does not contain any composite or multi valued attributes or their combinations.

1NF (composite attribute)

Customer		
CustomerID	Name	Address
C01	Raj	Jamnagar Road, Rajkot
C02	Meet	Nehru Road, Jamnagar

- Above relation has three attributes CustomerID, Name, Address.
- Here address is composite attribute which is further divided in to sub attributes as Society and City.
- So above relation is not in 1NF.

1NF (composite attribute)

Customer		
CustomerID	Name	Address
C01	Raj	Jamnagar Road, Rajkot
C02	Meet	Nehru Road, Jamnagar

- Problem:
- It if difficult to retrieve the list of Customers living in 'Jamnagar' from above table.
- Reason is address attribute is composite attribute which contains road name as well as city name in single cell.

- *Solution*
- Divide composite attributes into number of sub- attribute and insert value in proper sub attribute.

Customer			
CustomerID	Name	Road	City
C01	Raj	Jamnagar Road	Rajkot
C02	Meet	Nehru Road	Jamnagar

1NF (multi-valued attribute)

Student		
RollNo	Name	FailedinSubjects
101	Raj	DS, DBMS
102	Meet	DBMS, DS
103	Jeet	DS, DBMS, DE
104	Harsh	DBMS, DE, DS
105	Nayan	DE, DBMS, DS

- Above relation FailedinSubjects is a multivalued attribute which can store more than one values.
- So above relation is not in 1NF.

1NF (multi-valued attribute)

Student

RollNo	Name	FailedinSubjects
101	Raj	DS, DBMS
102	Meet	DBMS, DS
103	Jeet	DS, DBMS, DE
104	Harsh	DBMS, DE, DS
105	Nayan	DE, DBMS, DS

- Problem:
- It if difficult to retrieve the list of students failed in 'DBMS' and 'DS' but not in other subjects from above table.
- Reason is attribute is a multivalued attribute which can store more than one values

- *Solution:* Split the table into two tables in such a way that
- first table contains all attributes except multi-valued attribute and
- other table contains multi-valued attribute and
- Insert primary key of first table in second table as a foreign key

Student

RollNo	Name
101	Raj
102	Meet
103	Jeet
104	Harsh
105	Nayan

Result

RID	RollNo	Subject
1	101	DS
2	101	DBMS
3	102	DBMS
4	102	DS
5	103	DS
…	…	…

Second normal form (2NF)

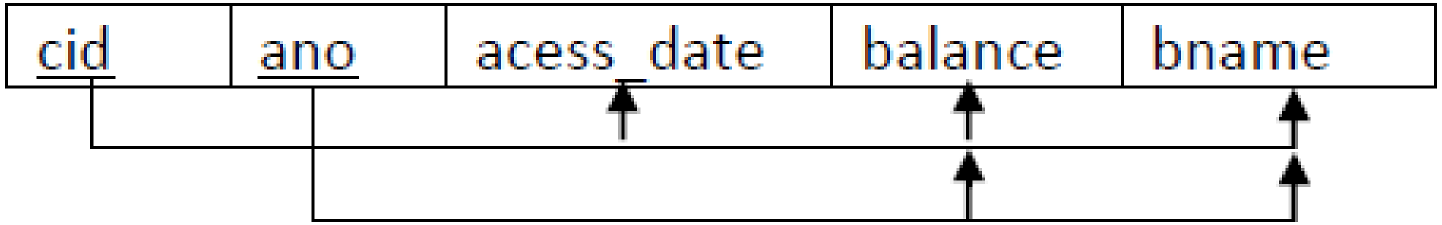
Condition for 2NF:

- A relation R is in second normal form (2NF)
- if and only if it is in 1NF and
- every non-prime attribute is fully dependent on the candidate key.

OR

- A relation R is in second normal form (2NF)
- if and only if it is in 1NF and
- no any non-prime attribute is partially dependent on the candidate key.

Second normal form (2NF)



- Above relation has five attributes cid, ano, acess_date, balance, bname and two FDS

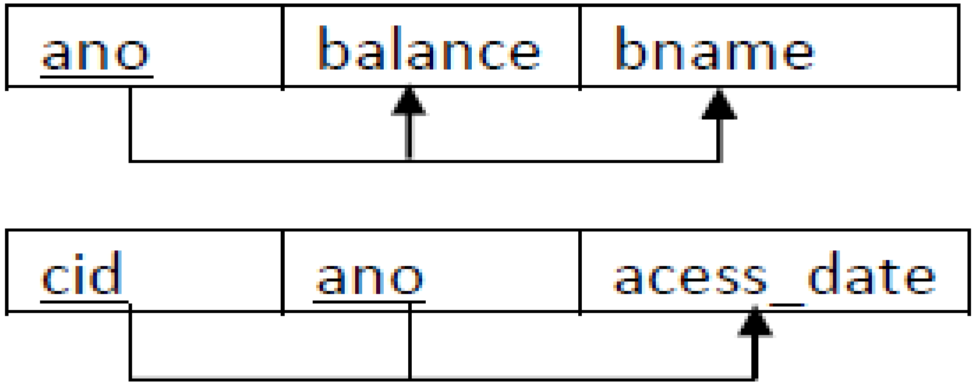
  FD1 {cid,ano} -> {acess_date,balance,bname} and

  FD2 ano -> {balance,bname}

- We have cid and ano as primary key.
- As per FD2 balace and bname are only depend on ano not cid.
- In above table balance and bname are partial dependent on primary key.
- So above relation is not in 2NF.

Second normal form (2NF)- Solution

- Decompose relation in such a way that resultant relation does not have any partial FD.
- For this purpose remove partial dependent attribute that violets 2NF from relation.
- Place them in separate new relation along with the prime attribute on which they are full dependent.
- The primary key of new relation will be the attribute on which it if fully dependent.
- Keep other attribute same as in that table with same primary key.



Third normal form (3NF)

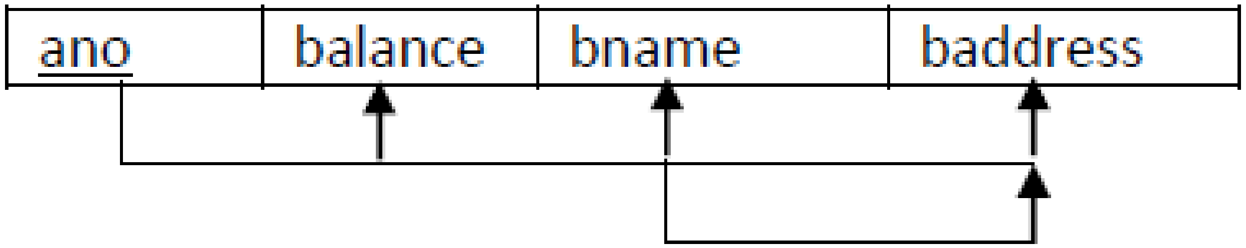
Conditions for 3NF

It is **in 2NF** and there is **no transitive dependency**.

(Transitive dependency???) $A \rightarrow B$ & $B \rightarrow C$ then $A \rightarrow C$.

- A relation R is in third normal form (3NF) if and only if it is in 2NF and every non-prime attribute is non-transitively dependent on the primary key.

Third normal form (3NF)



- Above relation has four attributes ano, balance, bname, baddress and two FDS
- FD1 ano -> {balance, bname, baddress} and
- FD2 bname -> baddress
- So ano -> baddress (using transitivity rule)
- So there is a non-prime attribute baddress which is transitively dependent on primary key ano.
- So above relation is not in 3NF.

Third normal form (3NF)

ANO	Balance	BName	BAddress
A01	50000	Rajkot	Kalawad Road
A02	40000	Rajkot	Kalawad Road
A03	35000	Rajkot	Kalawad Road
A04	25000	Rajkot	Kalawad Road

- *Problem*
- Transitively dependency results in data redundancy.
- In this relation branch address will be stored repeatedly from each account of same branch which occupy more space.

Third normal form (3NF)

- Solution: Decompose relation in such a way that resultant relation does not have any non-prime attribute that are transitively dependent on primary key.
- For this purpose remove transitively dependent attribute that violets 3NF from relation.
- Place them in separate new relation along with the non-prime attribute due to which transitive dependency occurred. primary key of new relation will be this non-prime attribute.
- Keep other attributes same as in that table with same primary key.

Table 1

BName	BAddress
Rajkot	Kalawad Road

Foreign Key

Table 2

ANO	Balance	BName
A01	50000	Rajkot
A02	40000	Rajkot
A03	35000	Rajkot
A04	25000	Rajkot

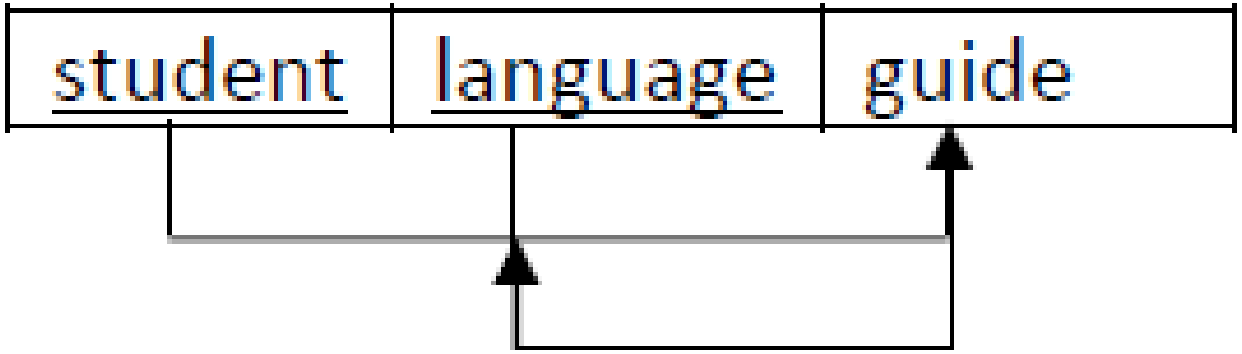
BCNF

- A relation R is in BCNF
  - if and only if it is in 3NF and
  - no any prime attribute is transitively dependent on the primary key.

OR

- A relation R is in BCNF
  - if and only if it is in 3NF and
  - for every functional dependency $X \rightarrow Y$, X should be the primary key of the table.

BCNF



- Above relation has five attributes cid, ano, acess_date, balance, bname and two FDS
- FD1 {student,language} -> guide and
- FD2 guide -> language
- So student -> language (using transitivity rule)
- So prime attribute (language) which is transitively dependent on primary key (student).
- So above relation is not in BCNF.

BCNF

Student	Language	Faculty
Mita	JAVA	Patel
Nita	VB	Shah
Sita	JAVA	Jadeja
Gita	VB	Dave
Rita	VB	Shah
Nita	JAVA	Patel
Mita	VB	Dave
Rita	JAVA	Jadeja

- *Problem:* Transitively dependency results in data redundancy.
- In this relation one student have more than one project with different guide then records will be stored repeatedly from each student and language and guides combination which occupies more space.

- Solution: Decompose relation in such a way that for X → Y, X should be the primary key of the table.
- For this purpose remove prime attribute, place them in separate new relation along with the non prime attribute on which it is dependent. The primary key of new relation will be this nonprime attribute.
- So above table can be decomposed as per following

Table 1

Faculty	Language
Patel	JAVA
Shah	VB
Jadeja	JAVA
Dave	VB

Table 2

Student	Faculty
Mita	Patel
Nita	Shah
Sita	Jadeja
Gita	Dave
Rita	Shah
Nita	Patel
Mita	Dave
Rita	Jadeja

4NF (fourth normal form)

- A table is in the 4NF
  - if it is in BCNF and
  - has no non multivalued dependencies.

- The multi-valued dependency X ->->Y holds in a relation R if for a dependency X → Y, if for a single value of X, multiple (more than one) values of Y exists.
- two attributes (or columns) in a table are independent of one another, but both depend on a third attribute. A **multivalued dependency** always requires at least three attributes because it consists of at least two attributes that are dependent on a third.

- For a dependency A -> B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency. The table should have at least 3 attributes and B and C should be independent for A ->> B multi
- valued dependency. For example,

PERSON	MOBILE	FOOD_LIKES
Mahesh	9893/9424	Burger / pizza
Ramesh	9191	Pizza

```
Person->-> mobile,
Person ->-> food_likes
```

4NF (fourth normal form)- *Example1*

- Suppose a student can have more than one subject and more than one activity.

Student_Info		
Student_Id	**Subject**	**Activity**
100	Music	Swimming
100	Accounting	Swimming
100	Music	Tennis
100	Accounting	Tennis
150	Math	Jogging

- Note that all three attributes make up the Primary Key.
- Note that Student_Id can be associated with many subject as well as many activities (multi-valued dependency).

4NF (fourth normal form)- *Example1*

- Here are the tables Normalized

Student_Info		
Student_Id	**Subject**	**Activity**
100	Music	Swimming
100	Accounting	Swimming
100	Music	Tennis
100	Accounting	Tennis
150	Math	Jogging

Decompose

Student_Id	Subject
100	Music
100	Accounting
150	Math

StudentId	Activity
100	Swimming
100	Tennis
150	Jogging

4NF (fourth normal form)- *Example2*

A table can have both functional dependency as well as multi-valued dependency to gather.

- **studentID → address**
- **studentID →→ subject**
- **studentID →→ activity**

studentID	address	subject	activity
101	C. G. Road	DS	Cricket
101	C. G. Road	DBMS	Cricket
101	C. G. Road	DS	Football
101	C. G. Road	DBMS	Football

4NF (fourth normal form)- *Example2*

studentID	address	subject	activity
101	C. G. Road	DS	Cricket
101	C. G. Road	DBMS	Cricket
101	C. G. Road	DS	Football
101	C. G. Road	DBMS	Football

Decompose

studentID	subject
101	DS
101	DBMS

studentID	address
101	C. G. Road

studentID	activity
101	Cricket
101	Football

5NF (fifth normal form)

- A table is in the 5NF
  - if it is in 4NF and
  - if it cannot have a lossless decomposition in to any number of smaller tables (relations).
- It is also known as Project-join normal form (PJ/NF).

5NF (fifth normal form)

ResultID	RollNo	StudentName	SubjectName	Result
1	101	Raj	DBMS	Pass
2	101	Raj	DS	Pass
3	101	Raj	DE	Pass
4	102	Meet	DBMS	Pass
5	102	Meet	DS	Fail
6	102	Meet	DE	Pass
7	103	Suresh	DBMS	Fail
8	103	Suresh	DS	Pass
9	103	Suresh	DE	Fail

- Above table is not in 5NF because we can decompose into sub tables.

5NF (fifth normal form)

- If we decompose above table into multiple table as per follows:

RollNo	StudentName
101	Raj
102	Meet
103	Suresh

SubjectID	SubjectName
1	DBMS
2	DS
3	DE

ResultID	RollNo	SubjectID	Result
1	101	1	Pass
2	101	2	Pass
3	101	3	Pass
4	102	1	Pass
5	102	2	Fail
6	102	3	Pass
7	103	1	Fail
8	103	2	Pass
9	103	3	Fail

Example

- **A college maintains details of its lecturers' subject area skills. These details comprise: Lecturer Number, Lecturer Name, Lecturer Grade, Department Code, Department Name, Subject Code, Subject Name and Subject Level.**
- **Assume that each lecturer may teach many subjects but may not belong to more than one department.**
- **Subject Code, Subject Name and Subject Level are repeating fields.**
- **Normalize this data to Third Normal Form.**

Solution

- *UNF*
- Lecturer Number, Lecturer Name, Lecturer Grade, Department Code, Department Name, Subject Code, Subject Name, Subject Level
- *1NF*
- Lecturer Number, Lecturer Name, Lecturer Grade, Department Code, Department Name
- Lecturer Number, Subject Code, Subject Name, Subject Level
- *2NF*
- Lecturer Number, Lecturer Name, Lecturer Grade, Department Code, Department Name
- Lecturer Number, Subject Code
- Subject Code, Subject Name, Subject Level
- *3NF*
- Lecturer Number, Lecturer Name, Lecturer Grade, Department Code
- Department Code, Department Name
- Lecturer Number, Subject Code
- Subject Code, Subject Name, Subject Level


Terima Kasih

धन्यवाद।


merci
spasiba
kam ouen
gratzias
manana
mahalo
cheers
toda
grassie
thank you
danki
hvala
gracias
grazie
tak
kitos
welalin
THANK YOU
mahalo
danki
thanks
takk
domo arrigato
gracias
danke
grattitude
merci
dankon
talofa
migwetch
kitos
thanks
na gode
mesi
modupe
takk
dziekuje