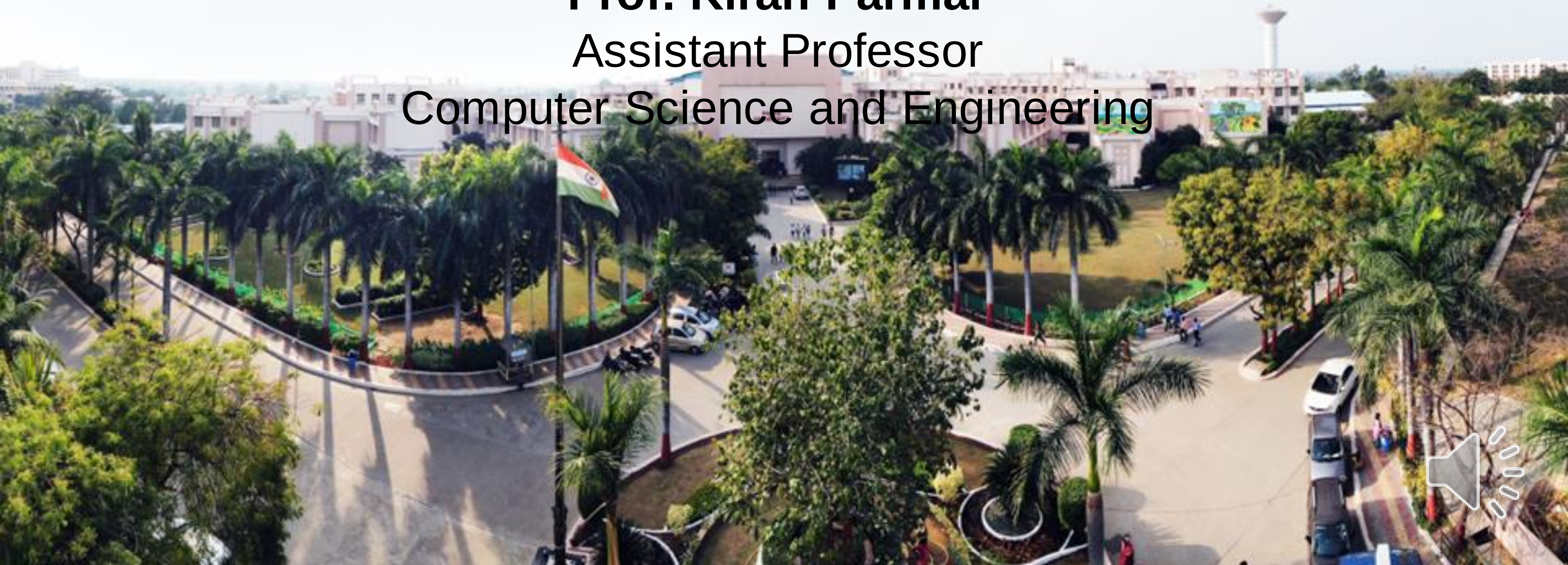
Parul® University

Database Management System 203105251

Prof. Swapnil Umbarkar, Prof. Mitali Acharya,
Prof. Kiran Parmar
Assistant Professor
Computer Science and Engineering



Parul® University

UNIT-2

Relational Query Languages

Relational Algebra

- In order to query the database instances of a relational databases we have Relational Algebra and Calculus Algebra.
- Relational Algebra takes relational database instances as input and yields the instances only as output.
- Fundamental operations included in the Relational Algebra:
  A) Select
  B) Project
  C) Union
  D) Set Difference
  E) Cartesian Product
  F) Rename

Parul® University

DIGITAL LEARNING CONTENT

Relational Algebra

- **Fundamental operations are divided into:**
  A) Unary Operations namely Select, Project, Rename
  B) Binary Operations namely Union, Set Difference, Cartesian Product

Relational Algebra

UNARY OPERATIONS:

•1) Select: Selects the tuples from the database which satisfy the given predicate. Denoted by sigma (σ). The predicates are written in subscript of (σ).

Example: (here, Eno=123 is predicate) $\sigma_{Eno=123}$

•2) Project: Project allow us to project a subset of relations or the whole relation. Denoted by pi (Π). Attributes to be appeared come in subscript of (Π).

Example: (here, Eno and name in subscript are subset of relation to be projected) $\Pi_{Eno,\ name}$

Relational Algebra

- 3) Rename: Rename operator renames the relational-algebraic expression to refer them. Denoted by rho (ρ). Name to be given come in subscript of (ρ).
   Example: (here, stud_record name is given to the output of expression E)
   $\rho_{\text{stud\_record}\,(A1,A2,..,An)}\ (E)$

Relational Algebra

BIANRY OPERATIONS :

•1) UNION: Union performs binary union between two given relations. Denoted using the symbol **U** .

Example: $\Pi_{\text{Eno, name}}$ **U** $\Pi_{\text{Eno, name}}$

•2) SET DIFFERENCE: Set difference is used when we want some set of relation to not be present in other set of relation. Denoted using the symbol (—).

Example: $\Pi_{\text{Eno, name}}$ — $\Pi_{\text{Eno, name}}$

Relational Algebra

- 3) CARTESIAN PRODUCT: It combines information of two different relations. Denoted using the symbol **X**.
  Example: $\sigma_{Address=‘Vadodara’}$(Student X Faculties)

Relational Calculus

- Relational Calculus is non-procedural query language. In this the user is concerning only on how the end results are obtained. Relational calculus never concerns on how results are obtained, it concerns what results are being obtained.
- Types of Relational Calculus:
  A) Tuple Relational Calculus
  B) Domain Relational Calculus

Relational Calculus

A) Tuple Relational Calculus:-

- In this the set of tuples is selected from a relation.
- Expressed as: { T | P (T) }

  Here, T is the resulting tuple

  P(T) is condition used to deduce T

  Example: { T.s_name | Student_name(T) AND T.course = 'CSE' }

Relational Calculus

B) Domain Relational Calculus:-

•Domain relational calculus is same as tuple relational calculus. Operators are same but domain relational calculus uses connectives, and (∧), or (∨) and not (┐). It uses Existential (∃) and Universal Quantifiers (∀) to bind the variable.

Denoted as: { a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}

where, a1, a2 are attributes

P stands for formula built by inner attributes

SQL3

- Sql3 includes data definition and management techniques form object-oriented DBMS, with taking care of relational DBMS platform.
- DBMS which supports SQL3 are known as Object-relational or OR-DBMS.

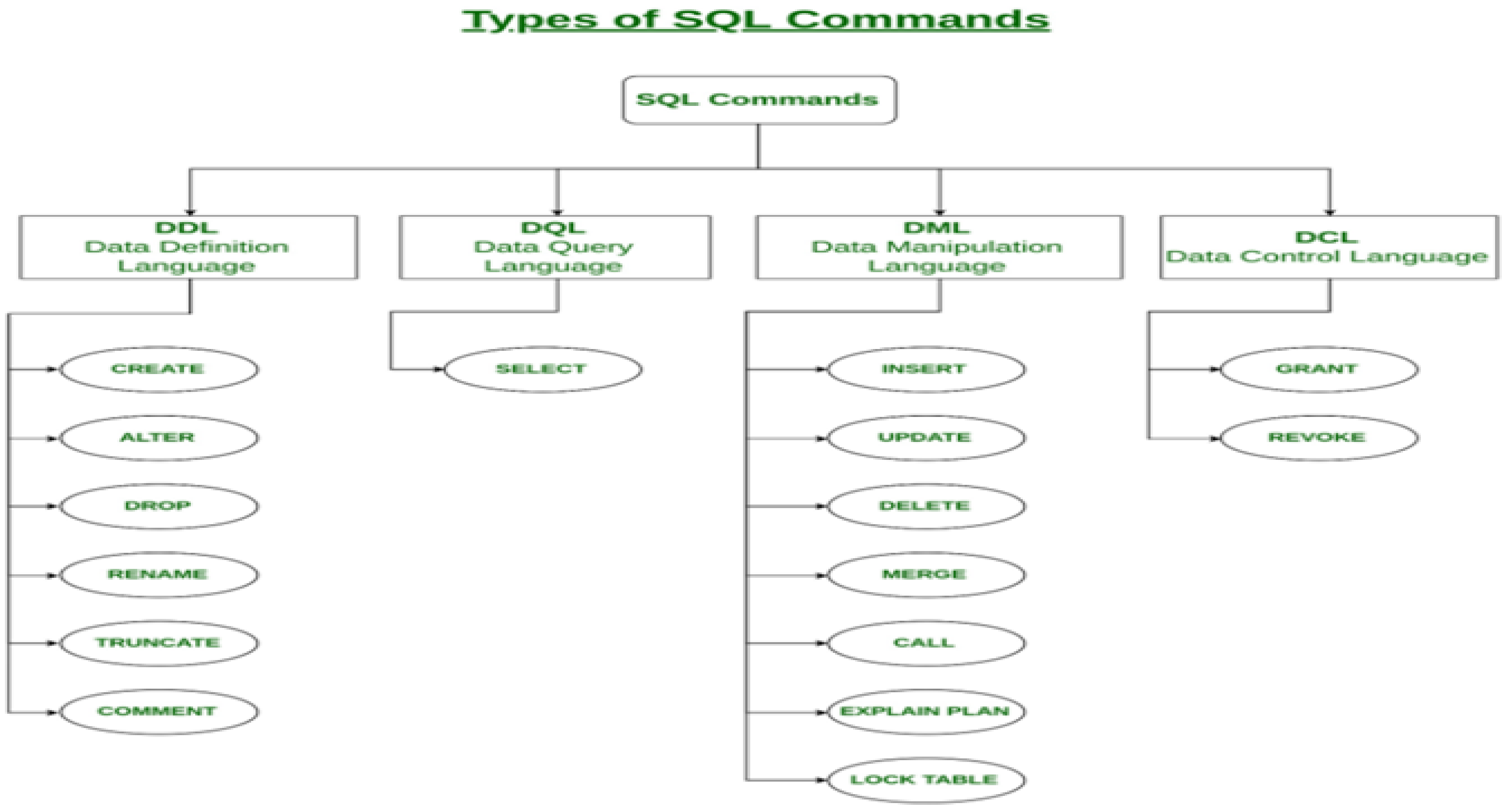
SQL3 features

- Abstract data type
  A) User defined data types.
  B) Equality and ordering functions.
  C) Encapsulation: Public, Private, Protected.
  D) Inheritance.
- Sub-tables that inherit all columns from another table.
- Persistent Stored Modules (Programming Language). Create methods. SQL and extensions. External language.
- User defined operators.
- Triggers for events.
- External language support Call-Level Interface (CLI) Direct access to DBMS Embedded SQL SQL commands in an external language.

SQL (Structed Query Language)

- Structured Query Language(SQL),the database language, is used to perform certain operations on existing database and also create a database. SQL uses commands like Create, Insert, Update etc. to carry out the required tasks.
- These SQL commands are mainly categorized into four categories as:
  A) DDL – Data Definition Language
  B) DQl – Data Query Language
  C) DML – Data Manipulation Language
  D) DCL – Data Control Language

SQL (Structed Query Language)



Types of SQL commands[1]

DDL and DML Constructs

- DDL, Data definition language, is the SQL commands set which are used to define the structure of the data structures itself.
  Example: Create, Alter etc.
- On the other hand, DML, Data manipulation language, is the set of SQL commands used to manipulate the data itself.
  Example: Insert, update, delete etc.

DDL and DML Constructs

Examples of DML:

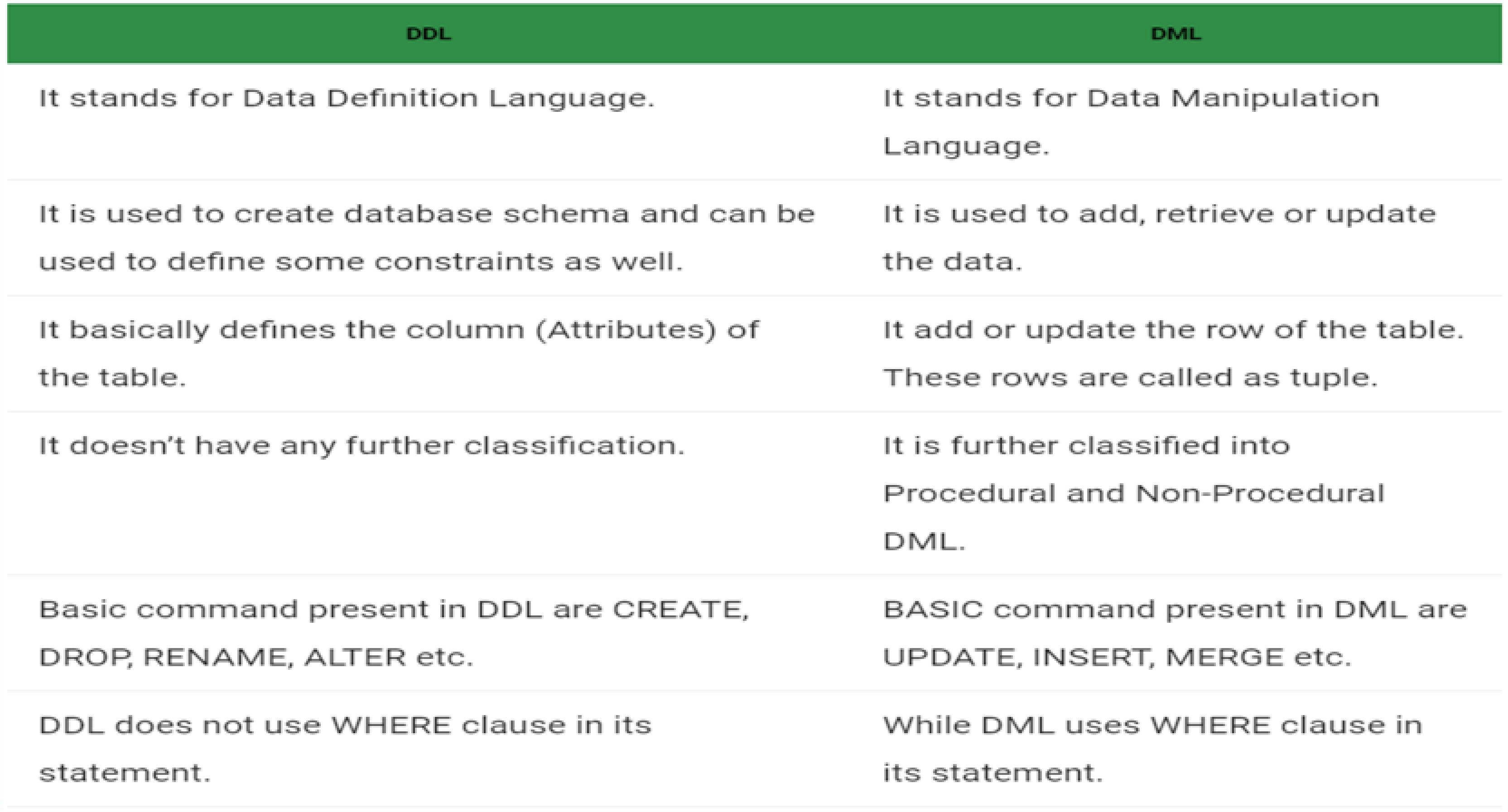
- INSERT – is used to insert data into a table.
- UPDATE – is used to update existing data within a table.
- DELETE – is used to delete records from a database table.

DDL and DML Constructs

Examples of DDL commands:

- CREATE – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
- DROP – is used to delete objects from the database.
- ALTER-is used to alter the structure of the database.
- TRUNCATE–is used to remove all records from a table, including all spaces allocated for the records are removed.
- COMMENT –is used to add comments to the data dictionary.
- RENAME –is used to rename an object existing in the database.

DDL and DML difference



DDL	DML
It stands for Data Definition Language.	It stands for Data Manipulation Language.
It is used to create database schema and can be used to define some constraints as well.	It is used to add, retrieve or update the data.
It basically defines the column (Attributes) of the table.	It add or update the row of the table. These rows are called as tuple.
It doesn't have any further classification.	It is further classified into Procedural and Non-Procedural DML.
Basic command present in DDL are CREATE, DROP, RENAME, ALTER etc.	BASIC command present in DML are UPDATE, INSERT, MERGE etc.
DDL does not use WHERE clause in its statement.	While DML uses WHERE clause in its statement.

Difference between DDL and DML[2]

Commercial and open source DBMSs

COMMERCIAL DBMS

- SQL server (Microsoft)
- IBM DB2 (IBM)
- Oracle
- Sybase
- Informix (IBM)
- Access (Microsoft)
- Cache (Intersystems – nonrelational)

OPEN SOURCE DBMS

- MySQL
- PostgreSQL

Parul® University

DIGITAL LEARNING CONTENT

Relational Algebra Queries

Consider the following given relational schema:
student(S_id, name, enrolled_in, DOB, L_id, division)
lecturer(L_id, L_name, Subject_code)
subject(code, name, lecturer)

TABLE 1: students

S_id	Name	Enrolled_in	DOB	L_id	Division
301	Riya	AI	22/03/1999	454	B1
302	Rohit	ML	21/01/2000	456	B7
303	Raman	DBMS	22/11/2000	564	B10
304	Shweta	OS	11/03/200	563	B8

Relational Algebra Queries

Consider the following given relational schema:
student(S_id, name, enrolled_in, DOB, L_id, division)
lecturer(L_id, L_name, Subject_code)
subject(code, name, lecturer)

TABLE 2: lecturer

L_id	L_name	Subject_code
454	ABC	03485
456	XYZ	03486
564	ASD	03567
563	PQS	03566

Relational Algebra Queries

Consider the following given relational schema:
student(S_id, name, enrolled_in, DOB, L_id, division)
lecturer(L_id, L_name, Subject_code)
subject(code, name, lecturer)

TABLE 3: subject

code	Name	Lecturer
03485	AI	454
03486	ML	456
03567	DBMS	564
03566	OS	563

Relational Algebra Queries

Queries related to given tables.

- Enlist student enrolled in subject AI.

  $\Pi_{S\_id}(\sigma_{enrolled\_in = AI}(students))$

- Who is the FR of student id 302.

  $\Pi_{FR\_id}(\sigma_{S\_id = 302}(students))$

- Enlist the students enrolled in subject AI and ML.

  $\Pi_{name}(\sigma_{enrolled\_in = AI}(students)) \cap \Pi_{name}(\sigma_{enrolled\_in = ML}(students))$

- Enlist the faculties who take AI or ML.

  $\Pi_{L\_name}(\sigma_{enrolled\_in = AI}(students)) \cup \Pi_{name}(\sigma_{enrolled\_in = ML}(students))$

Relational Algebra Queries

Queries related to given tables.

- Enlist the student with their FR having division B7.

  $\Pi_{name,L\_name} (\sigma_{division = B7} (students \bowtie lecturer))$

- Rename table lecturer to faculty.

  $\rho_{faculty} (lecturer)$

- Enlist lecturer who is teaching OS and rename it to XYZ.

  $\rho_{XYZ} (\Pi_{L\_name} (\sigma_{L\_name = OS} (lecturer \bowtie subject)))$

- Select faculty id who is FR of Division B7.

  $\Pi_{L\_id} (\sigma_{division = B7} (students \bowtie lecturer))$

Relational Algebra Queries

Queries related to given tables.

- Enlist the subject taken by faculty ABC.

  $\Pi_{name} (\sigma_{L\_name = ABC} (lecturer \bowtie subject))$

- Enlist lecturer with subject taken by them and rename it to PQS.

  $\rho_{PQS}(\Pi_{L\_name, name} (\sigma_{L\_name = ABC} (lecturer \bowtie subject)))$

Tuple Calculus Queries

Consider the following given relational schema:
student(S_id, name, enrolled_in, DOB, L_id, division)
lecturer(L_id, L_name, Subject_code)
subject(code, name, lecturer)

Queries related to given schema: (Solve using Tuple calculus)

- Give student details whose DOB is greater than 21/03/2000.
  { t | t Ɛ students ^ t[DOB] > 21/03/2000 }

- Enlist the student enrolled in AI and taught by ABC lecturer.
  { t | ∃ s Ɛ students(t[L_id] = s[L_id] ^ t[L_name = ABC }

Tuple Calculus Queries

Queries related to given schema: (Solve using Tuple calculus)

- Find the lecturer who is FR of division B10

  { t | t Ɛ lecturer ^ t[FR_of_Div] = B10 }

- Enlist lecturer and subject who is teaching ML.

  { t | ∃ s Ɛ lecturer(t[code] = s[Subject_code] ^ t[name = ML }

- Enlist student and DOB who took the course DBMS.

  { t | t Ɛ students ^ t[enrolled_in] = DBMS}

SQL Examples

Consider the following tables:

TABLE 1: students

S_id	Name	Enrolled_in	DOB	L_id	Division
301	Riya	AI	22/03/1999	454	B1
302	Rohit	ML	21/01/2000	456	B7
303	Raman	DBMS	22/11/2000	564	B10
304	Shweta	OS	11/03/2001	563	B8
305	Shweta	OS	07/11/2000	563	B8

SQL Examples

TABLE 2: lecturer

L_id	L_name	Subject_code
454	ABC	03485
456	XYZ	03486
564	ASD	03567
451	GHJ	03485
563	PQS	03566

SQL Examples

- Select rows from table 1 where student is not enrolled in AI.

```
Select * from Students
        where NOT enrolled_in = ‘AI’;
```

- Find the list of students name which is 5 character length and ending with t.

```
        Select name from students
        where name LIKE ‘____t’ ;
```

- Find number of students born in year 2000.

```
        Select count(DOB) from students
where DOB LIKE ‘%2000’ ;
```

- Select student id and name who are taught by lecturer ASD.

```
        Select students.S_id, students.name from students
        INNER JOIN lecturer ON students.L_id = lecturer.L_id
Where lecturer.L_name = ‘ASD’;
```

SQL Examples

- Update the course enrolled in of student id 303 to TOC.

```
Update students
set enrolled_in = 'TOC'
where s_id = 303 ;
```

- Insert record in table 2 : 345, FGH, 03105 .

```
Insert into lecturer
values(345, 'FGH', 03105) ;
```

- Select students name enrolled in subject whose code is ending with 66.

```
Select students.name from students
INNER JOIN lecturer ON students.L_id = lecturer.L_id
where lecturer.subject_code LIKE '%66' ;
```

- Select students name from table students with no redundancy.

```
Select DISTINCT(Name)  from students ;
```

SQL Examples

- Count number of students in each division and enlist it in a table.

```
Select COUNT(Name) , division from students
GROUP BY division ;
```

- Select students name and id with L_id as rename L_id to FR.

```
Select s_id, name, L_id AS FR from students ;
```

Select students id where L_id is 454 or 561.

```
Select S_id from students
where L_id = 454 OR L_id = 561 ;
```

- Give student details with DOB in descending order.

```
Select * from students
order by DOB desc ;
```

- Display division where number of students is greater than 1.

```
Select COUNT(division) , name from students
        GROUP BY name ;
```

SQL Examples

- Delete the record from table 2 where subject code is ending with 86.
  Delete from lecturer where subject_code LIKE ‘%86’ ;
- Change the constraint of table 1 at column enrolled_in to ICT.
          Alter table students
  alter enrolled_in SET DEFAULT ‘ICT’ ;
- Add a new column, Phone_no in table lecturer .
  Alter table lecturer
  ADD Phone_no varchar(12) ;
- Use ANY clause and find the details of students who is taught by any lecturer whose name ends with ‘S’.
  Select * from students
          where L_id = ANY (Select L_name from lecturer where L_name LIKE ‘%S’ ;

SQL Examples

- Select student id if there exists a student in division B1.
  ```
  Select S_id from students
  where EXISTS (Select * from students where division = ‘B1’ ) ;
  ```
- Create a virtual table named VT where lecturer name is starting with A.
  ```
  Create VIEW[VT] AS
          Select * from lecturer
  where L_name LIKE ‘A%’ ;
  ```
- Remove primary key S_id from students.
  ```
  Alter table students
          DROP PRIMARY KEY ;
  ```
- Count number of lecturers taking subject 03485.
  ```
  Select count(L_id) from lecturer
  GROUP BY subject_code ;
  ```

SQL Examples

- Select lecturer details where subject code is not 03102, 03103, 03566.
  Select * from lecturer
  where subject_code NOT IN (03102, 03103, 03566) ;
- Select students with DOB greater than 21/03/2000.
  Select * from students
  where DOB > 21/03/2000 ;
- Select details of lecturer who is not teaching any subject i.e., subject_code = NULL.
  Select * from lecturer
  where subject_code IS NULL ;
- Select lecturer who teach subject 03102 and 03485
  Select * from lecturer
  where subject_code = 03102 AND subject_code = 03485 ;

SQL Examples

- Select L_id from table 2 if ALL students in table 1 have division B8.
  Select L_id from lecturer
  where L_id = ALL (Select L_id from students where division = ‘B8’ ;
- Drop table lecturer.
  DROP table lecturer ;
- Insert values 307, Siya, OS to the table students
  Insert into students(S_id, Name, Enrolled_in)
  values(307, ‘Siya’, ‘OS’);

Parul® University

DIGITAL LEARNING CONTENT

Section - 2

Relational Database Design

Domain and data dependency:

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

- Functional Dependency (Single and Multivalued Dependency)
- Transitive 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.

Parul® University

DIGITAL LEARNING CONTENT

Functional Dependency:

- **EXAMPLE:**

Table 1.1: Example of Functional Dependency

Sid	Sname	Cid
S1	A	C1
S1	A	C2
S2	B	C2
S3	C	C3

•Question: If sid → sname is a valid FD?

Functional Dependency:

- **EXAMPLE:**
- As we have discussed, for X → Y, for each value of X there should be one Y value.

Sid	→	Sname
S1		A
S1		A
S1		A
S2		B
S3		B

- As you can see over here, for each value of Sid there is only one Sname value. Hence, Sid → Sname is valid FD.

Types of Functional Dependency:

Trivial FD:

•X→Y is trivial FD if Y is a subset of X.

•Every possible Trivial FD is implied in the relation.

•For example, Consider Table 1.1. Following FDs are Trivial FDs.

- Sid → Sid
- Sname → Sname
- SidSname → Sname
- SidSname → SidSname

Types of Functional Dependency:

Non- Trivial FD:

•X→Y is non- trivial FD if there is no common attribute in X, Y attribute set.

•For example, Consider Table 1.1.

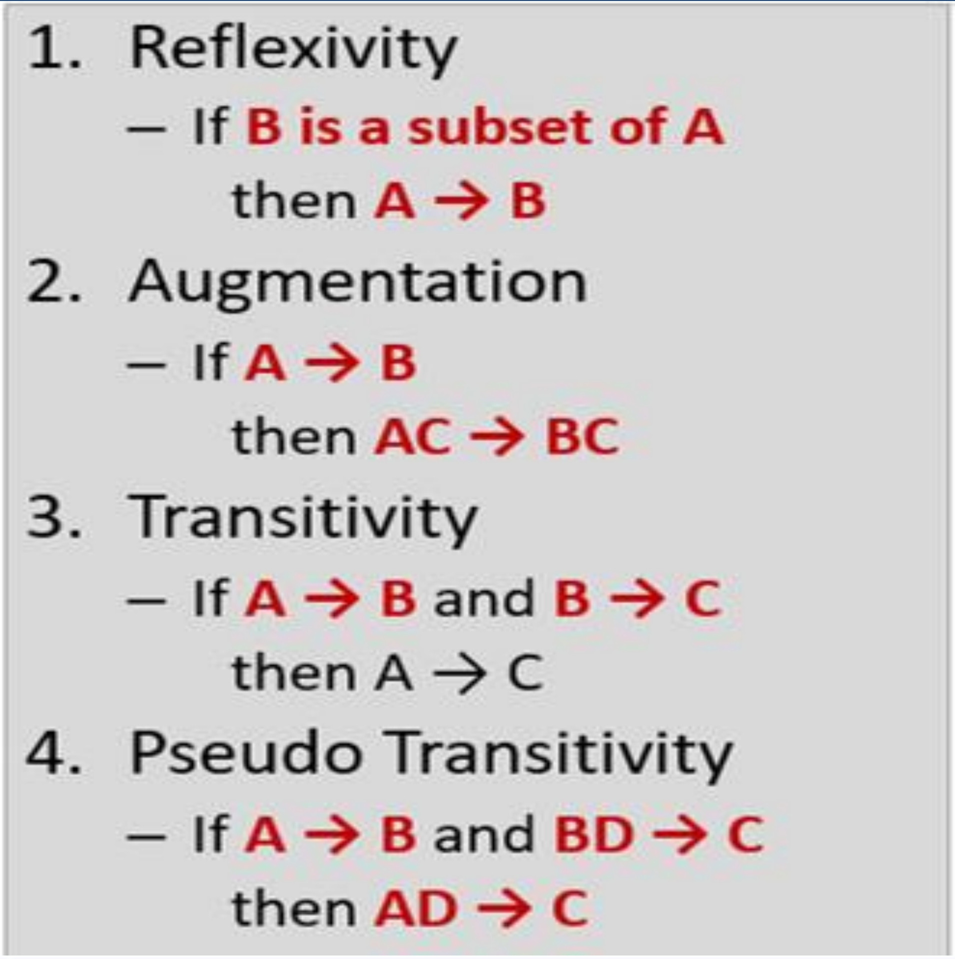
• Sid → Sname is non- trivial FD.

Armstrong's axioms:

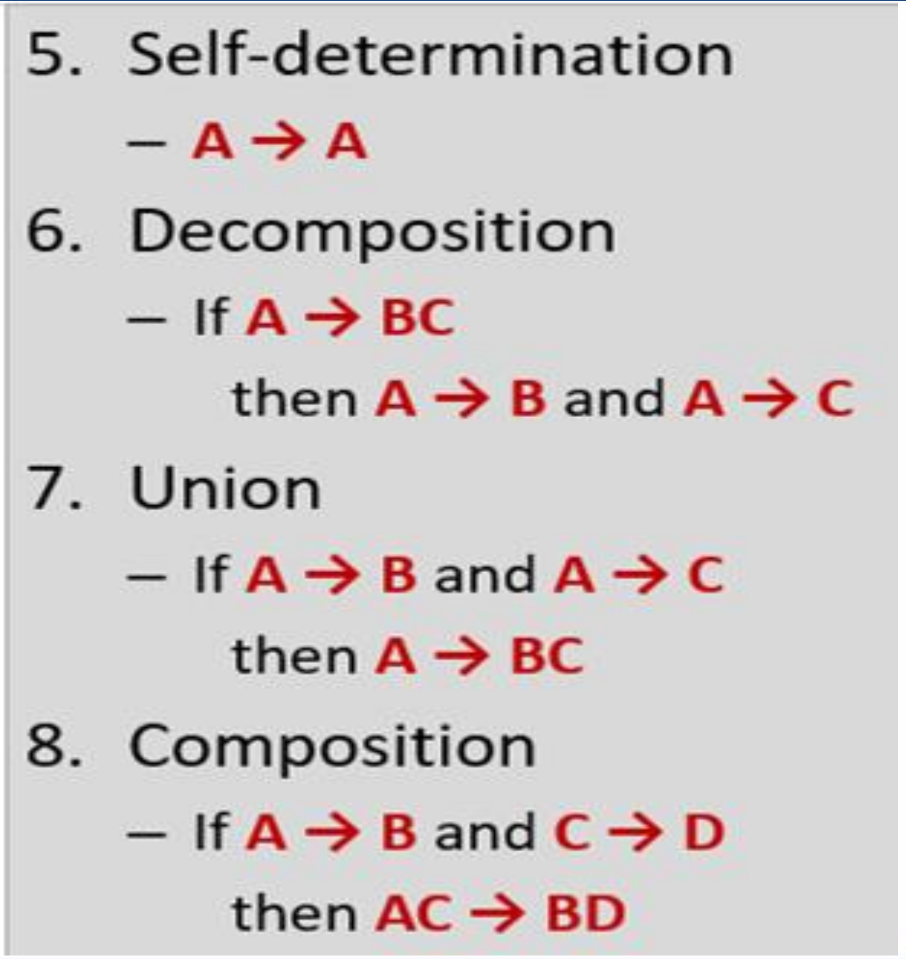
- Armstrong's axioms are a set of rules used to infer (derive) all the functional dependencies on a relational database.
- Let A, B, and C is subsets of the attributes of the relation R.
- Rules are mentioned in the next slide.

Parul® University

Armstrong's axioms:



1. Reflexivity
   - If B is a subset of A then $A \rightarrow B$
2. Augmentation
   - If $A \rightarrow B$ then $AC \rightarrow BC$
3. Transitivity
   - If $A \rightarrow B$ and $B \rightarrow C$ then $A \rightarrow C$
4. Pseudo Transitivity
   - If $A \rightarrow B$ and $BD \rightarrow C$ then $AD \rightarrow C$



5. Self-determination
   - $A \rightarrow A$
6. Decomposition
   - If $A \rightarrow BC$ then $A \rightarrow B$ and $A \rightarrow C$
7. Union
   - If $A \rightarrow B$ and $A \rightarrow C$ then $A \rightarrow BC$
8. Composition
   - If $A \rightarrow B$ and $C \rightarrow D$ then $AC \rightarrow BD$

Attribute Closure:

- Given a set of attributes, the closure of any attribute X under F is the set of attributes that are functionally determined by X under F.
- It is denoted by X^+.
- $[X]^+$ = { Set of attributes determined by X.}

Attribute Closure:

Example:

- Given relation R with attributes A,B,C,D,E,F,G and set of FDs,
  F = { A → B, B → C, AC → D, CD → E, G → A, AF → G }.
  Find the Closure of and AF.

- Answer: A^+ is { A, B, C, D, E)

- $[AF]^+$ = { A, B, C, D, E, F, G}

Super Key:

- FD’s determine Super key.
- X is a Super Key of R iff X^+ should determine all the attribute of Relation R.
- **Example:** R(ABCD), F = { $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$ }
- $[A]^+ = \{ A, B, C, D\}$
- Hence, A is one of the super keys.

Candidate Key:

- Minimal Super key is Candidate Key.

- X is candidate key of Relation R if and only if

1. X must be super key of R.

   AND

2. No proper Subset of attribute X is Super Key.

- **Prime attributes and Non-Prime Attributes:** Attributes of relations which are part of candidate key are known as Prime Attributes and attributes which are not part of candidate keys are known ad non-prime attributes.

Candidate Key:

Example:

- R(ABCD), F = { AB → C, C → D, B → E }
- $[AB]^+$ = { A, B, C, D, E}
- Hence, AB is one of the super keys.

- To check if AB is Candidate key or not, we have to check closure set of minimal subset of AB, i.e $[A]^+$ and $[B]^+$
- $[A]^+$ = {A} (**NOT SUPERKEY/CANDIDATE KEY)**
- $[B]^+$ = {BE} (**NOT SUPERKEY/CANDIDATE KEY)**

Candidate Key:

Note 1: If attribute is in R but not in non-trivial FD set, then that attribute must be part of Candidate key.

Example:

- R(ABCDEF), F = { $A \rightarrow D$, $F \rightarrow E$, $C \rightarrow F$, $F \rightarrow A$ }
- Here, B is not part of non-trivial FD set so it must be part of candidate key.
- $[C]^+$ = { A, C, E, F}
- **Candidate key:** { BC }
- **Prime attributes:** B and C

Candidate Key:

Note 2: If some attribute X is only part of determinant side but not belongs to any determiner of non-trivial FD set of relation R then X must be part of Candidate key.

Example:

- R(ABCDEF), F = { AB → C, E → F, D → C, C → B }
- Here, A,E are at the determinant side only.
- **Candidate key:** { ABE}
- **Prime attributes:** A,B and E

Candidate Key:

Exercise: Find Candidate Keys of below relations.

1.R(ABCDE), F = { AB → B, BC → D, CD → E, E → A}

1. R(ABCD), F = { AB → CD, C → A, D → B}

1.R(ABCDEF), F = { AB → C, C → DE, E → F, EF → B, E → A}

1.R(ABCDEFG), F = { AB → CD, AF → D, DE → F, C → G, F → E, G → A}

Normalization:

- Normalization is the procedure to reduce the Redundancy.
- Redundancy can occur in Relation if two or more independent relations are stored in the single table.
- For example, in the given example, two independent relations Student relation and Course relation is stored in single table.

Normalization:

- **Example of Redundancy:**

Table 1.2: Example of Redundancy in Database table.

Sid	Sname	Age	cid	Cname	fees
S1	A	20	C1	DBMS	5000
S2	A	22	C1	DBMS	5000
S3	B	22	C1	DBMS	5000
S2	A	22	C3	DAA	7000
S2	A	22	C4	DS	7000

Redundancy

Redundancy

Normalization:

Anomalies in Database Design:

- Due to redundancy in Database, Inconsistency (Anomalies) occurs.
- 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:

1. Insert anomalies, occur due to Insert Operation.
2. Delete anomalies, occur due to Delete Operation.
3. Update / Modification anomalies, occur due to update operation.

Normalization:

Insert Anomaly:

- Consider previous example, Student_Course(Sid, Sname, age, Cid, Cname, fees) as Sid as a primary key.
- Let us assume that a new course has been started by the Institute but initially there are no students enrolled for that course.
- Then the tuple for this course cannot be inserted in to this table as the Sid will have NULL value, which is not allowed because Sid is primary key.
- This kind of problem in the relation where **some tuple cannot be inserted** is known as insert anomaly.

Normalization:

Insert Anomaly:

Table 1.3 : Insert Anomaly

	Sid	Sname	Age	cid	Cname	fees
	S1	A	20	C1	DBMS	5000
	S2	A	22	C1	DBMS	5000
	S3	B	22	C1	DBMS	5000
	S2	A	22	C3	DAA	7000
	S2	A	22	C4	DS	7000
Insert Anomaly	NULL	NULL	NULL	C5	CD	8000

Normalization:

Delete Anomaly:

- Consider previous example, Student_Course(Sid, Sname, age, Cid, Cname, fees) as Sid as a primary key.
- Now consider there is one student in and that employee leaves the Institute.
- Then the tuple of that student has to be deleted from the table, but in addition to that information about the course 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.

Normalization:

Delete Anomaly:

- If student C leaves the institute, then we have to delete information related to that particular student. If we delete that, information regarding course C3 will be deleted.

Table 1.4 : Deletion Anomaly

Sid	Sname	Age	cid	Cname	fees
S1	A	20	C1	DBMS	5000
S2	A	22	C1	DBMS	5000
S3	B	22	C1	DBMS	5000
S2	A	22	C3	DAA	7000
S2	A	22	C4	DS	7000

Normalization:

Update Anomaly:

- Consider previous example, Student_Course(Sid, Sname, age, Cid, Cname, fees) as Sid as a primary key.
- Suppose the fees of a course has changed, this requires that the course in all the tuples corresponding to that course must be changed to reflect the new status.
- If we fail to update all the tuples of given course, then two different records of the student enrolled in the same course might show different fees which leads to inconsistency in the database.
- This kind of problem is known as **update or modification anomaly**.

Normalization:

Update Anomaly:

Table 1.5 : Update Anomaly

Sid	Sname	Age	cid	Cname	fees
S1	A	20	C1	DBMS	6000
S2	A	22	C1	DBMS	5000
S3	B	22	C1	DBMS	5000
S2	A	22	C3	DAA	7000
S2	A	22	C4	DS	7000

Update Anomaly

Normalization:

Need of Normalization:

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

Normalization:

Normal forms:

1 NF (First normal form)
2NF (Second normal form)
3NF (third normal form)
BCNF

$\Bigg\}$ 0% redundancy over Single valued FD's. ($X \rightarrow Y$)
Redundancy can be possible because of MVD's. ($X \rightarrow \rightarrow Y$)

4NF (forth normal form)
5NF (fifth normal form)

$\Big\}$ No Redundancy because of MVD's also

Normalization:

1NF:

Relational schema R is in 1 NF if no multivalued attribute in Relation R.

Or

Every attribute of relation R must be single valued.

Normalization:

1NF:

- Consider below relation.

Table 1.6: 1 NF Example

CustomerID	Name	Address
C01	A	Karelibag, Vadodara
C02	B	Satellite, Ahmedabad

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

Normalization:

1NF:

- Consider below relation.

Table 1.6: 1 NF Example

CustomerID	Name	Address
C01	A	Karelibag, Vadodara
C02	B	Satellite, Ahmedabad

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

Normalization:

1NF:

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

Table 1.7: 1 NF Example solution

CustomerID	Name	Area	City
C01	A	Karelibag	Vadodara
C02	B	Satellite	Ahmedabad

Normalization:

1NF:

- Default NF for RDBMS relation is 1 NF.
- Redundancy occurs due to 2NF, 3NF, BCNF in single valued FD's.

Parul® University

DIGITAL LEARNING CONTENT

Normalization:

How Redundancy can occur in any given FD X → Y?

- If non-trivial FD X → Y with X is not Super Key in R then X → Y form Redundancy in R.

NOT SUPERKEY → **X** X → **Y : Non-trivial FD**

Forms redundancy in Relation

Normalization:

How Redundancy can occur in any given FD X → Y?

•For example, consider below relation:

Table 1.8 : Redundancy in database table

Sid	Sname	Cid
S1	A	C1
S1	A	C2
S1	A	C3

Redundancy

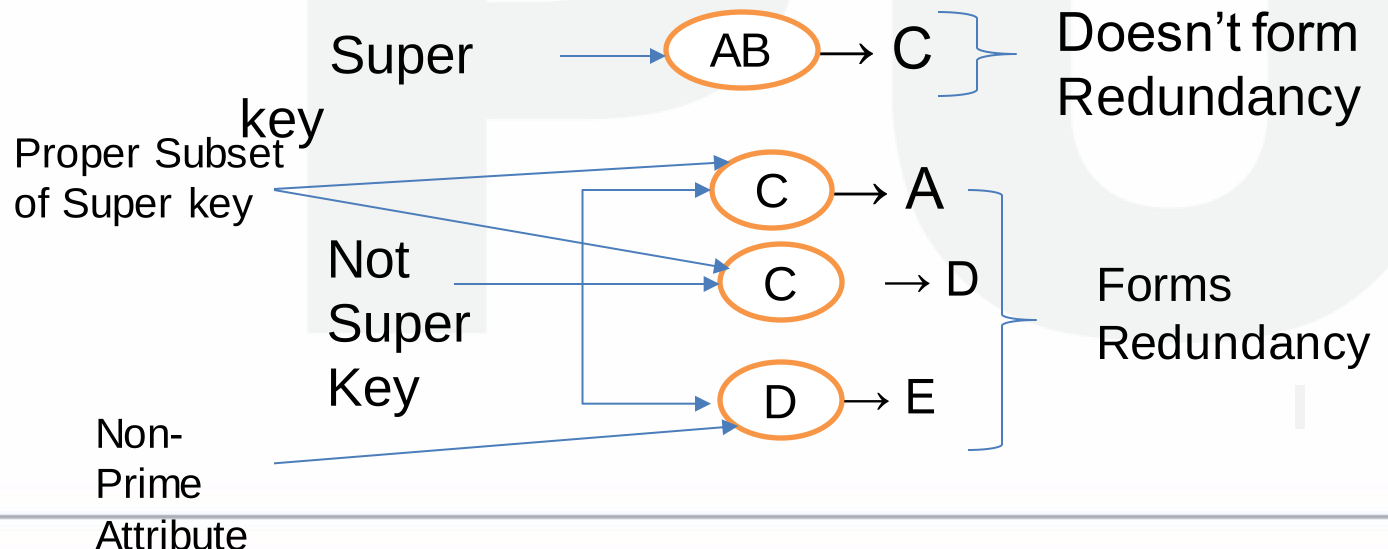
•Super key for this relation is {sidcid}.

•Consider FD, Sid → Sname. Sid is not Super Key for this relation. Hence we can see redundancy is the all the tuples in the relation.

Normalization:

How many FD’s of F form Redundancy?

- R(ABCDE), F = { AB → C, C → A, C → D, D → E }
- **Super key/Candidate key: { AB,BC}**



Normalization:

X → Y, non-trivial FD in R with X not Super key can be,

Case 1:
$$\begin{tikzcd}\boxed{\text{Proper Subset of CK (X)}} \arrow[r] & \boxed{\text{Non-Prime attribute (Y)}}\end{tikzcd}$$
E.g. C → D

Case 2:
$$\begin{tikzcd}\boxed{\text{Non-Prime attribute (X)}} \arrow[r] & \boxed{\text{Non-Prime attribute (Y)}}\end{tikzcd}$$
E.g. D → E

Case 3:
$$\begin{tikzcd}\boxed{\text{Proper Subset of CK (X)}} \arrow[r] & \boxed{\text{Proper Subset of CK (Y)}}\end{tikzcd}$$
E.g. C → A

Normalization:

Summary:

Table 1.9 : Summary of Normal forms

	1 NF	2 NF	3 NF	BCNF
Case 1	Yes	No	No	No
Case 2	Yes	Yes	No	No
Case 3	Yes	Yes	Yes	no

Normalization:

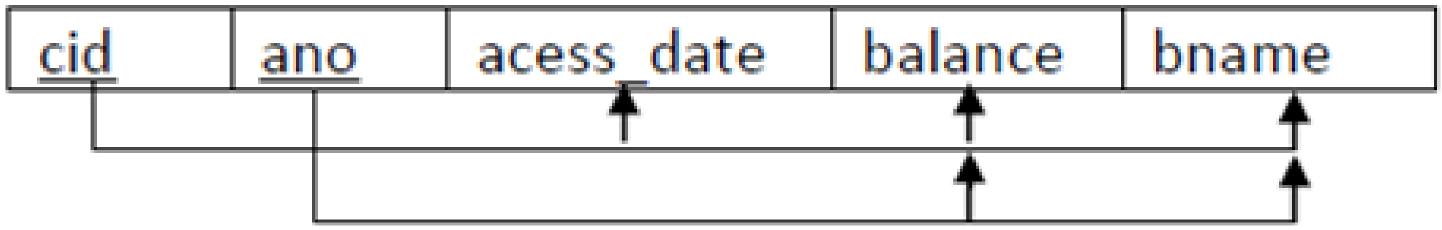
2NF:

A relation schema R is in second normal form (2NF) if and only if it is in 1NF and

no any non-key attribute is **partially dependent on the primary key**.

Normalization:

2NF Example:



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

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

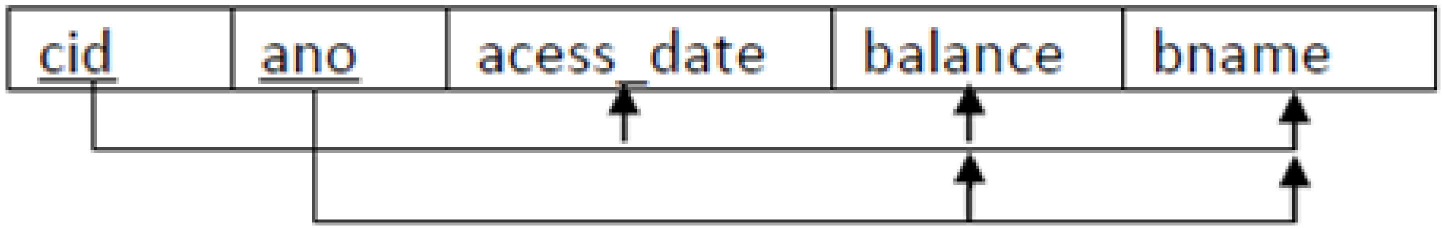
  FD2 : {ano} -> {balance,bname}

Parul® University

DIGITAL LEARNING CONTENT

Normalization:

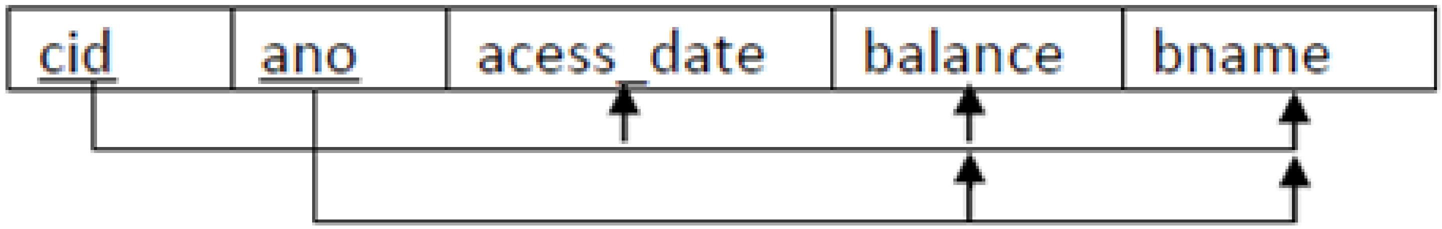
2NF Example:



- 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**.

Normalization:

2NF Example:

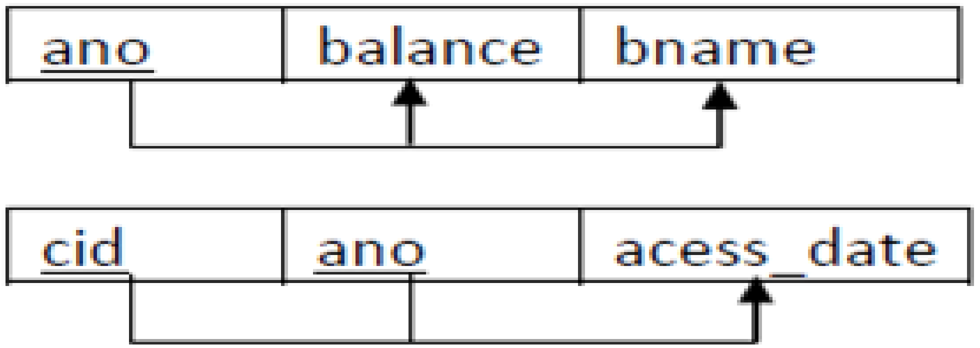


Problem:

- For example in case of joint account multiple customers have common accounts.
- If some account says ‘A02’ is jointly by two customers says ‘C02’ and ‘C04’ then data values for attributes balance and bname will be duplicated in two different tuples of customers ‘C02’ and ‘C04’.

Normalization:

2NF Example:

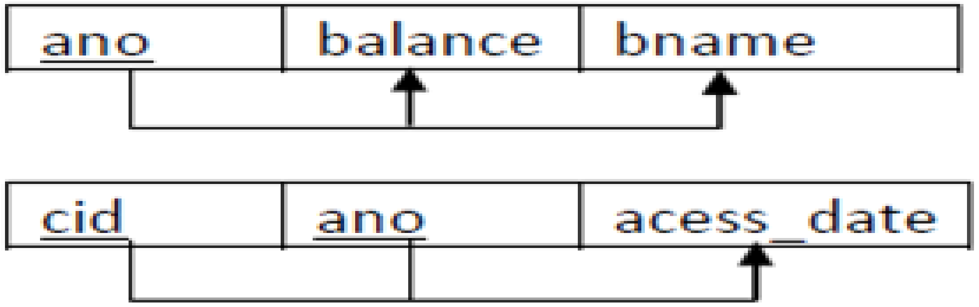


Solution:

- 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**.

Normalization:

2NF Example:



Solution:

- **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.

Normalization:

3NF:

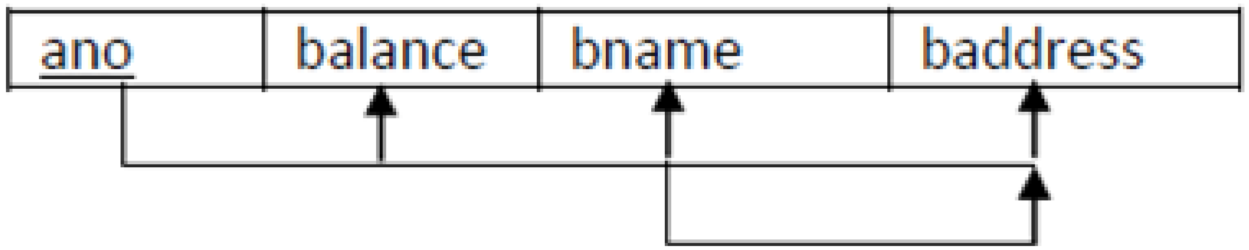
A relation schema R is in third normal form (3NF) if and only if it is in 2NF and no transitive dependencies in R.

Parul® University

DIGITAL LEARNING CONTENT

Normalization:

3NF Example:



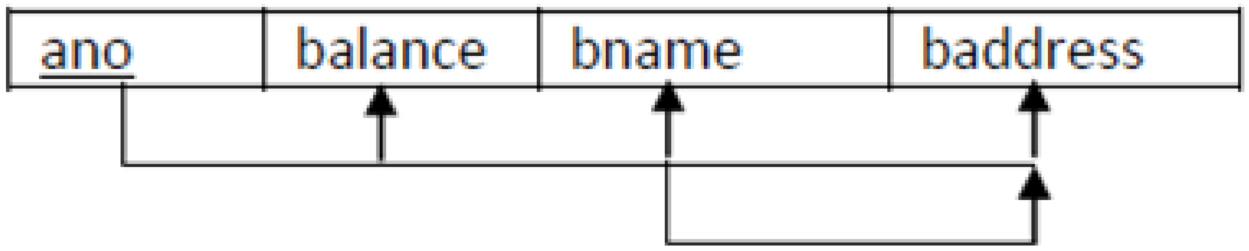
- Above relation has five attributes ano, balance, bname and baddress and two FDS

  FD1 : ano -> {balance, bname, baddress} and

  FD2 : bname -> baddress
- Using transitivity rule, ano -> baddress

Normalization:

3NF Example:

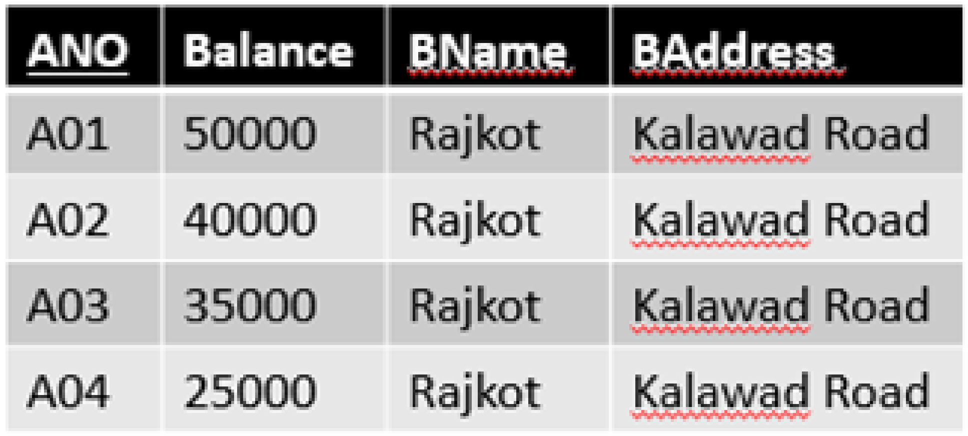


- So there is a **non-prime attribute baddress which is transitively dependent on primary key ano.**
- So above relation is **not in 3NF.**

Normalization:

3NF Example:

Table 1.10 : 3NF Example



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.

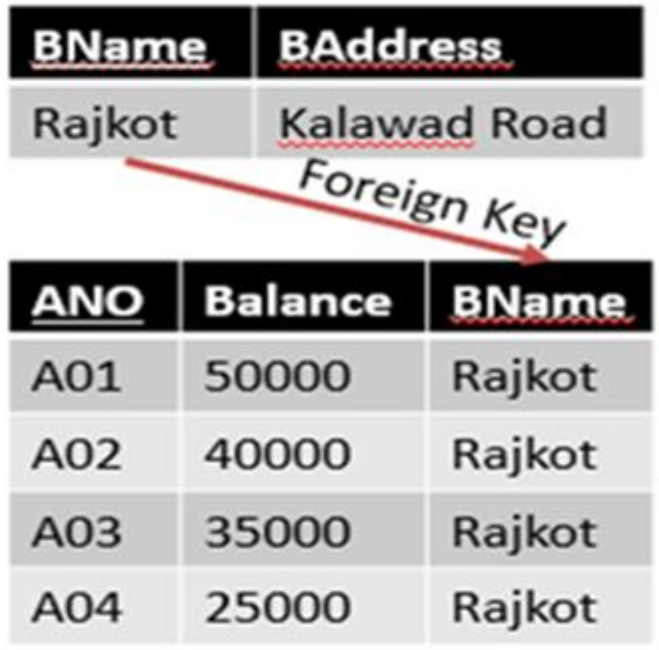
Normalization:

3NF Example:

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.

Table 1.11 : 3NF Solution


Foreign Key

BName	BAddress
Rajkot	Kalawad Road

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

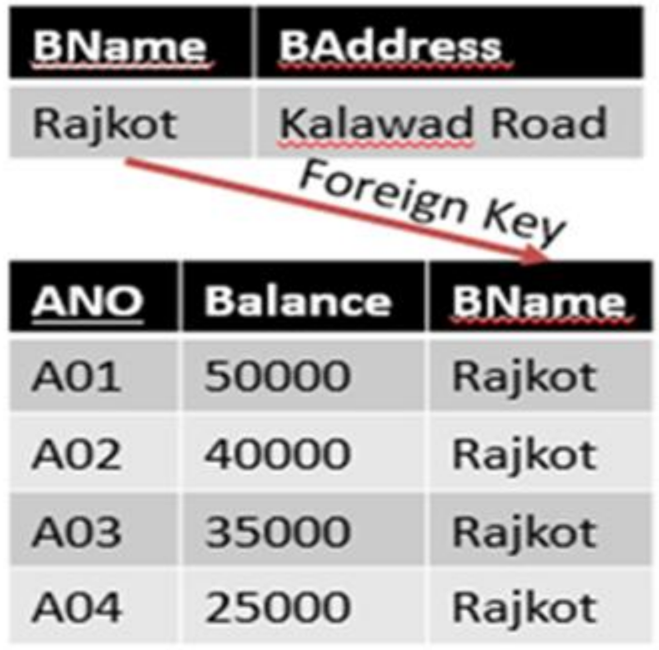
Normalization:

3NF Example:

Solution:

- 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 1.11 : 3NF Solution


Foreign Key

BName	BAddress
Rajkot	Kalawad Road

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

Normalization:

Boyce Codd Normal Form (BCNF):

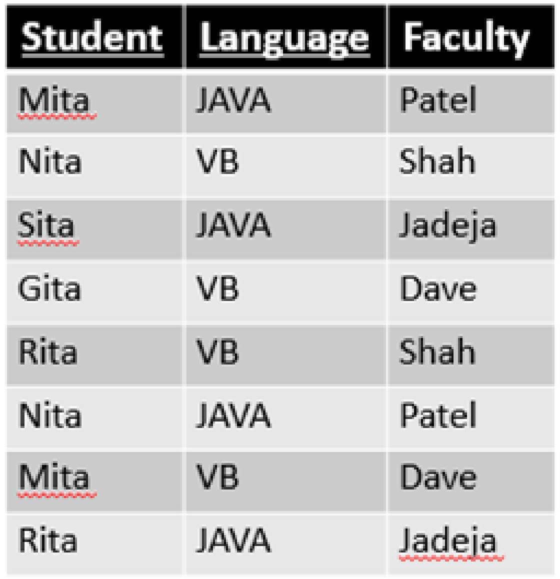
A relation R is in BCNF if and only if it is in 3NF and **for every functional dependency X → Y, X should be the primary key of the table**.

Normalization:

BCNF Example:

- Let R be a relation with following three attributes: student, language, Faculty
- FD1 : **{student, language} → Faculty**
- FD2 : **Faculty → language**
- Using transitivity rule, **student → language**
- So above relation is **not in BCNF.**

Table 1.12: BCNF Example



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

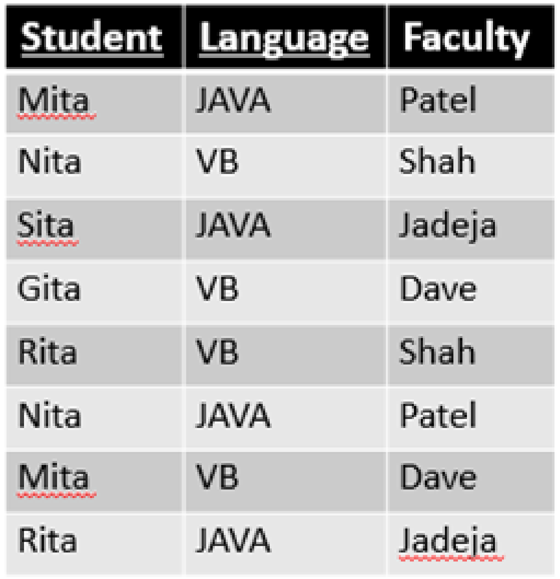
Normalization:

BCNF Example:

Problem:

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

Table 1.12: BCNF Example



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

Normalization:

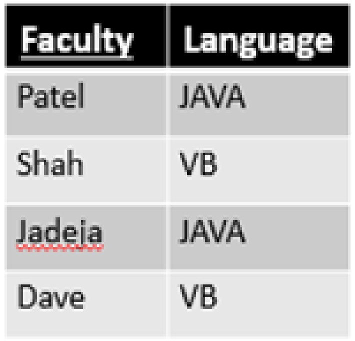
BCNF Example:

Solution:

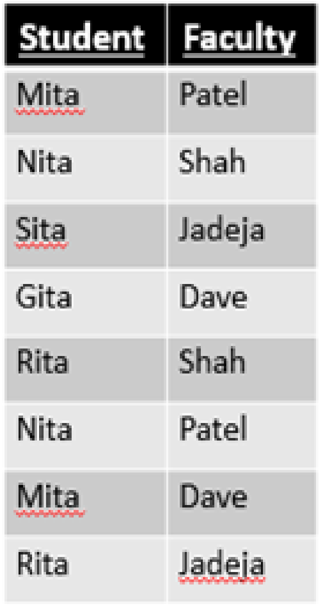
•Decompose relation in such a way that for X → Y, X should be the primary key of the table.

•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.

Table 1.13: BCNF Solution



Faculty	Language
Patel	JAVA
Shah	VB
Jadeja	JAVA
Dave	VB



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

Normalization:

Examples: Find out Highest NF of given Relation.

1.R(ABCDE), F = { ABC → D, D → A, BD → E}

Find out Candidate key: Candidate keys : { ABC, BCD}
Prime attributes: A, B, C, D. Non-Prime attributes: E

	BCNF	3NF	2NF
ABC → D	Yes	Yes	Yes
D → A	No	Yes	Yes
BD → E	No	No	No

Highest NF of given Relation : **1 NF**

Normalization:

Examples: Find out Highest NF of given Relation.

2.R(ABCDE), F = { A → B, B → AC, C → DE}

Find out Candidate key: Candidate keys : {A, B}
Prime attributes: A, B. Non-Prime attributes: C, D, E

	BCNF	3NF	2NF
A → B	Yes	Yes	Yes
B → AC	Yes	Yes	Yes
C → DE	No	No	Yes

Highest NF of given Relation : **2 NF**

Parul® University

DIGITAL LEARNING CONTENT

Normalization:

Examples: Find out Highest NF of given Relation.
3.R(ABCDE), F = { A → BC, CD → E, B → D, E → A}

Find out Candidate key: Candidate keys : {A, E, CD, BC}
Prime attributes: A, B, C, D, E. Non-Prime attributes: none

	BCNF	3NF	2NF
A → BC	Yes	Yes	Yes
CD → E	Yes	Yes	Yes
E → A	Yes	Yes	Yes
B → D	No	Yes	Yes

Highest NF of given Relation : **3 NF**

Normalization:

Question:

- **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.**

Normalization:

Solution:

- UNF

  Lecturer Number, Lecturer Name, Lecturer Grade, Department Code, Department Name, Subject Code, Subject Name, Subject Level

- After removing multi valued attributes we get **1NF**
  - Lecturer Number, Lecturer Name, Lecturer Grade, Department Code, Department Name
  - Lecturer Number, **Subject Code, Subject Name, Subject Level** (Partial Dependency)

Normalization:

Solution:

- After removing partial dependency we get **2NF**
  - Lecturer Number, Lecturer Name, Lecturer Grade, **Department Code, Department Name (**Transitive Dependency)
  - Lecturer Number, Subject Code
  - Subject Code, Subject Name, Subject Level
- After removing transitive dependency we get **3NF**
  - Lecturer Number, Lecturer Name, Lecturer Grade, Department Code
  - Department Code, Department Name
  - Lecturer Number, Subject Code
  - Subject Code, Subject Name, Subject Level

Decomposition:

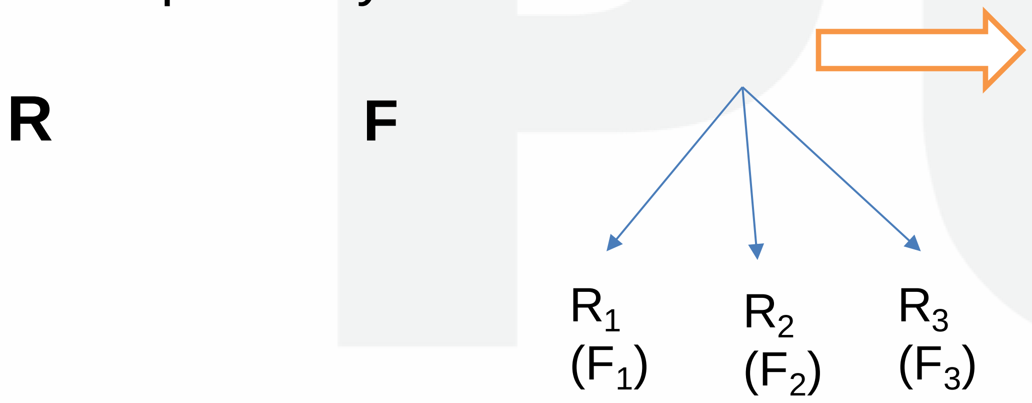
- **Properties of Decomposition:**

1) Dependency Preserving Decomposition
2) Lossless join Decomposition

Decomposition:

Dependency Preserving Decomposition:

- Let R be the Relational schema with FD set F Decomposed into R_1, R_2, R_3…… R_n sub relations with FD set F_1, F_2, F_3,…….. F_n respectively.



Decomposition:

Dependency Preserving Decomposition:

In general,

$[F_1 \cup F_2 \cup F_3 \cup \ldots\ldots F_n] \subseteq F$

If $[F_1 \cup F_2 \cup F_3 \cup \ldots\ldots F_n] = F$ then it is DP Decomposition.

If $[F_1 \cup F_2 \cup F_3 \cup \ldots\ldots F_n] \subset F$ then it is NOT DP Decomposition.

Decomposition:

Example 1:

R(ABCDE) is decomposed into D = { AB, BC, CD, DE }
F = { $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$, $D \rightarrow E$, $D \rightarrow B$ }

Step 1 : Find FD's of Sub relations:
R_1 (AB) : (F_1)

$[A]^+$ = { A, B, C, D, E} [We can derive B from A. So $A \rightarrow B$. C,D,E are not in R_1]
$[B]^+$ = { B, C, D, E} [We cannot derive A from B.]

Decomposition:

Example 1:

R_2 (BC) : (F_2)

$[B]^+$ = { B, C, D, E} [We can derive C from B. So B → C. A,D,E are not in R_2]

$[C]^+$ = { B, C, D, E} [We can derive B from C. So C → B. A,D,E are not in R_2]

R_3 (CD) : (F_3)

$[C]^+$ = { B, C, D, E} [We can derive D from C. So C → D. A,B,E are not in R_2]

$[D]^+$ = { B, C, D, E} [We can derive C from D. So D → C. A,B,E are not in R_2]

R_4 (DE) : (F_4)

Decomposition:

Example 1:

$\{F_1 \cup F_2 \cup F_3 \cup F_4\} = \{A \rightarrow B,\ B \rightarrow C,\ C \rightarrow B,\ C \rightarrow D,\ D \rightarrow C,\ D \rightarrow E\}$

Now, $F = \{A \rightarrow B,\ B \rightarrow C,\ C \rightarrow D,\ D \rightarrow E,\ D \rightarrow B\}$

We can say that If $[F_1 \cup F_2 \cup F_3 \cup \ldots\ldots F_n] = F$.

This Decomposition is **Dependency Preserving Decomposition.**

Decomposition:

Example 2:

R(ABCD) is decomposed into D = { ABC, AD, BCD }
F = { AB $\rightarrow$ CD, D $\rightarrow$ A}

Step 1 : Find FD's of Sub relations:

R_1 (ABC) : (F_1)
$[AB]^+$ = { A, B, C, D} [We can derive C from A and B. So AB $\rightarrow$ C. D is not in R_1]

Decomposition:

Example 2:

R_2 (AD) : (F_2)
$[A]^+ = \{A\}$ [We cannot derive A from D.]
$[D]^+ = \{A,D\}$ [We can derive A from d. So $D \rightarrow A$. A,D are not in R_2]

R_3 (BCD) : (F_3)
$[B]^+ = \{B\}$ [We can derive D from C. So $C \rightarrow D$. A,B,E are not in R_2]
$[C]^+ = \{C\}$ [We can derive C from D. So $D \rightarrow C$. A,B,E are not in R_2]
$[BD]^+ = \{A, B, C, D\}$ [We can derive C from B and D. So $BD \rightarrow C$. A is not in R_3]

Decomposition:

Example 2:

$\{F_1 \cup F_2 \cup F_3\} = \{AB \rightarrow C, D \rightarrow A, BD \rightarrow C\}$

Now, $F = \{AB \rightarrow CD, D \rightarrow A\}$

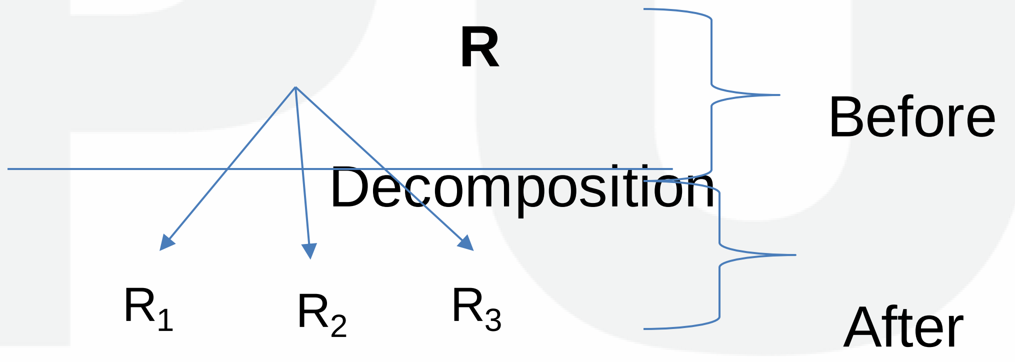
We can say that If $[F_1 \cup F_2 \cup F_3 \cup \ldots\ldots F_n] \subset F$.

This Decomposition is **NOT Dependency Preserving Decomposition.**

Decomposition:

Lossless join Decomposition:

- Let R be the Relational schema with FD set F Decomposed into R_1, R_2, R_3…… R_n sub relations.



Decomposition

Decomposition:

Lossless join Decomposition:

In general,

$[R_1 \bowtie R_2 \bowtie R_3 \bowtie \ldots\ldots \bowtie R_n] \supseteq R$

If $[R_1 \bowtie R_2 \bowtie R_3 \bowtie \ldots\ldots \bowtie R_n] = F$ then it is lossless join Decomposition.

If $[R_1 \bowtie R_2 \bowtie R_3 \bowtie \ldots\ldots \bowtie R_n] \supset F$ then it is lossy Decomposition.

Decomposition:

Lossless join Decomposition:

•Consider the below Relation,

Sid	Sname	Cid
S1	A	C1
S1	A	C2
S2	B	C2
S3	B	C3

{ Sid → Sname}

Candidate key: {SidCid}

Decomposition:

Lossless join Decomposition:

- This relation is decomposed into R_1 (SidSname) and R_2 (SidCid)

R_1 (SidSname)

Sid	Sname
S1	A
S2	B
S3	B

{Sid}: key

R_2 (SidCid)

Sid	Cid
S1	C1
S1	C2
S2	C2
S3	C3

{SidCid}: key

$R_1 \bowtie R$ $\Rightarrow$

R(SidSnameCid)

Sid	Sname	Cid
S1	A	C1
S1	A	C2
S2	B	C2
S3	B	C3

$R_1 \bowtie R_2$ = R, so it is LOSSLESS JOIN DECOMPOSTION.

Decomposition:

Lossless join Decomposition:

- This relation is decomposed into R_1 (SidSname) and R_2 (SnameCid)

Sid	Sname
S1	A
S2	B
S3	B

{Sid}: key

Sname	Cid
A	C1
A	C2
B	C2
B	C3

{SidCid}: key

$R_1 \bowtie R$

Sid	Sname	Cid
S1	A	C1
S1	A	C2
S2	B	C2
S2	B	C3
S3	B	C3
S3	B	C2
S3	B	C3

$R_1 \bowtie R_2 \supset R$, so LOSSY JOIN DECOMPOSTION.

Decomposition:

Lossless join Decomposition:

•Common attribute should be the candidate key of one of the table.

•Relational schema R with FD set F decomposed into R_1 & R_2
Decomposition is lossless if and only if,
a) $R_1 \cap R_2 \rightarrow R_1$ $(R_1 \cap R_2)$: Candidate key for R_1
OR
b)$R_1 \cap R_2 \rightarrow R_2$ $(R_1 \cap R_2)$: Candidate key for R_2

•Union of attributes of R_1 and R_2 must be equal to the attributes of R.

•Intersection of attributes in R_1 and R_2 must not be NULL.

Decomposition:

Example:

R(ABCDE), F = { $AB \rightarrow C$, $C \rightarrow D$, $B \rightarrow E$}

Relation R is decomposed into,

1)D = { ABC, CD }

$R_1(ABC) \cap R_2(CD) = C$

$[C]^+ = \{ C,D \}$

Here, union of attributes of R_1 and R_2 is not equal to attributes of R.

Hence, this decomposition is lossy join decomposition.

Decomposition:

Example:

R(ABCDE), F = { AB → C, C → D, B → E}
Relation R is decomposed into,
2)D = { ABC, DE }

Here, Intersection of attributes in R_1 and R_2 is NULL.

Hence, this decomposition is lossy join decomposition.

Decomposition:

Example:

R(ABCDE), F = { AB → C, C → D, B → E}
Relation R is decomposed into,
3)D = { ABC, CDE }

$R_1(ABC) \cap R_2(CDE) = C$

$[C]^+ = \{ C,D \}$

CD is NOT candidate key for either of R_1 & R_2.

Hence, this decomposition is lossy join decomposition.

Decomposition:

Example:

R(ABCDE), F = { AB → C, C → D, B → E}
Relation R is decomposed into,
4)D = { ABCD, BE }

$R_1(ABCD) \cap R_2(BE) = B$

$[B]^+ = \{ B,E \}$

CD is candidate key for $R_2(BE)$.

Hence, this decomposition is lossless join decomposition.

Decomposition:

Example:

R(ABCDE), F = { AB → C, C → D, B → E}

Relation R is decomposed into,

5)D = { ABC, ABDE }

$R_1(ABC) \cap R_2(ABDE) = AB$

$[AB]^+ = \{ A,B,C,D,E \}$

AB is candidate key for both tables R_1 (ABC) and R_2(ABDE).

Hence, this decomposition is lossless join decomposition.

Decomposition:

Example:

R(ABCDE), F = { AB → C, C → D, B → E}

Relation R is decomposed into,

6)D = { ABC, CD, BE }

$R_1(ABC) \cap R_2(CD) = C$

$[C]^+ = \{ C,D \}$

CD is candidate key for either of $R_2(CD)$.

$R_1 \bowtie R_2\ (ABCD) \cap R_3(BE) = B$, $[B]^+ = \{ B,E \}$

B is candidate key for $R_3(BE)$.

Hence, this decomposition is lossless join decomposition.

Decomposition:

Example:

R(ABCDE), F = { AB → C, C → D, B → E}
Relation R is decomposed into,
7)D = { ABC, CD, DE }

$R_1(ABC) \cap R_2(CD) = C$
$[C]^+ = \{ C,D \}$

CD is candidate key for either of $R_2(CD)$.
$R_1 \bowtie R_2\ (ABCD) \cap R_3(DE) = D$, $[D]^+ = \{ D \}$
D is NOT candidate key for any table.
Hence, this decomposition is lossy join decomposition.

Parul® University

Section - 3

Query Processing and Optimization

Topics

- Query processing
- Steps in query processing
- Measures of query cost
- Selection operation
- Evaluation of expressions
- Query optimization
- Transformation of relational expressions
- Sorting and join

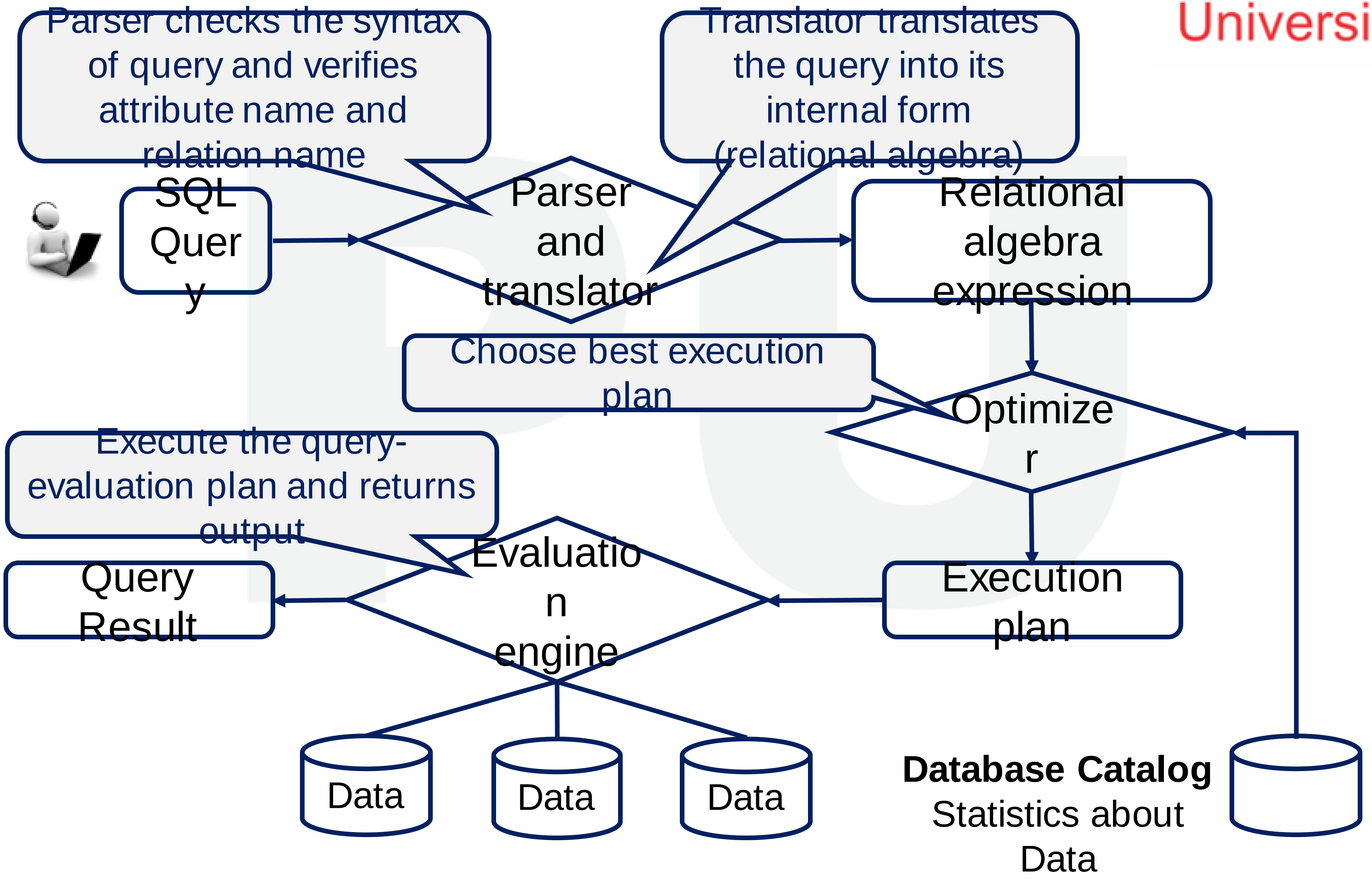
Query Processing

- **Query Processing** is process to convert high level queries to low level so machine can understand and perform the action that are requested by the user.

- It is used to extract data from database and to fetch data it takes three steps:

1. Parsing and Translation
2. Optimization
3. Evaluation

Steps in Query Processing


Parser checks the syntax of query and verifies attribute name and relation name
Translator translates the query into its internal form (relational algebra)
SQL Query
Parser and translator
Relational algebra expression
Choose best execution plan
Optimizer
Execute the query-evaluation plan and returns output
Query Result
Evaluation engine
Execution plan
Data
Data
Data
Database Catalog
Statistics about Data

Step in Query Processing

1. **Parsing and Translation:**

   - SQL is suitable for humans.
   - Relational Algebra is suitable for system.
   - First step in query processing is to convert SQL to Relational Algebra Expression.

- **Parsing(Parser):**
  - Check Syntax
  - Check Schema Elements
- **Translation(Translator)**
  - Parse Tree $\longrightarrow$ Relational Algebra

Step in Query Processing

2. Optimization(Optimizer):

- Selects the best Query Evaluation Plan to evaluate the query.
- Generate Query Evaluation Plan for all possible option
    - **Query Evaluation plan** = Query Tree + Algorithms

Step in Query Processing

3. **Evaluation Engine:**

    - Evaluates the Query Plan (selected by Optimizer) and fetches the data from database.

Measures of Query cost

- Total time taken by statement/query to execute and to fetch data from database is Query Cost.
- Some factors are:
  - **Communication cost:**
    - Applicable to distributed/parallel system.
  - **CPU Cycles:**
    - Difficult to calculate
    - CPU speed improves at much faster rate as compared to Disk speed

Measures of Query cost

- **Disk Access:**
  - Dominates the total time to execute a query

- **Disk Access Cost:**
  - No. of seeks
  - No. of blocks Read
  - No. of Blocks Write

Note: Generally cost of writing is greater than cost of reading.

Selection operation

- **File Scan:** Search algorithm that used to locate and retrieve data that satisfy a selection condition in a file.

- **Symbol for Selection operator:** σ (Sigma)

- **Syntax:** $\sigma_{\mathit{condition}}$ (Relation)

- Searching algorithm for Selection Operation:

  1. Linear Search(A1)
  2. Binary Search(A2)

Selection operation

1. **Linear Search(A1):**
   - This algorithm will search and scan all blocks available and tests all records/data to determine whether or not they satisfy the selection condition.
     - **Cost(A1)** = B_R (**worst case**)

       where B_R denotes number of blocks
   - If the condition is on a **Key(primary) attribute**, then system can stop searching if desired record found.
     - **Cost(A1)** = $B_R/2$ (**best case**)
   - If the condition is on **non (primary) key attribute**, then multiple blocks may contain desired records, then the price of scanning such blocks have to be added to the estimate value.
   - This is slower than Binary Search.

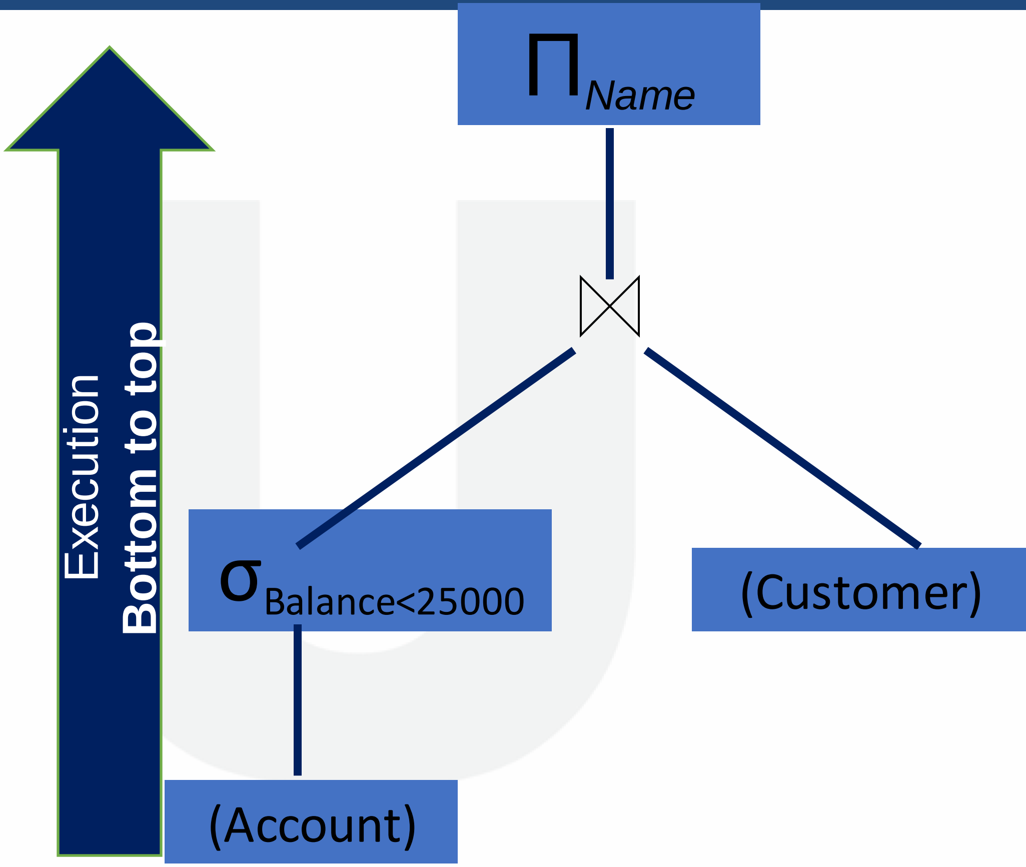
Selection operation

2. **Binary Search(A2):**
   - File (relation) ordered based on attribute A (primary index).
     - **Cost(A2)** = $\log_2(B_R)$
   - This is faster than Linear Search.

Evaluation of expressions

- Query(Expression) may contain multiple operations and due to that solving query (Expression) will be difficult.
- To evaluate such type of Query we have to solve one by one in proper order.
- There are two methods to evaluate multiple operations expression:

1. Materialization
2. Pipelining



Materialization

- **Materialization** starts the bottom of the expression and performs a single operation at a time.
- **Materialized**(store in temporary relation) each intermediate result of all operations performed and use this result as input to evaluate next-level operations.
- The **cost of materialization** can be quite high as overall cost can be compute as:

  Overall Cost = Sum of Costs of individual operations + Cost of writing intermediate results to the disk
- **Disadvantages of Materializations are:**
  - Due to intermediate results, it creates lots of temporary relations.
  - It performs many Input/Output operations.

Pipelining

- **In Pipelining**, the output of one operation is passed as input to another operation. i.e. it forms a queue.
- As the output of one operation is passed to the next operation in the **Pipelines, the number of intermediate temporary relations will be reduced.**
- Performing operations in Pipeline **eliminates the cost of writing and reading temporary relations.**

- It can be executed in two ways:
  - Demand Driven(Lazy Evaluation)
  - Producer Driven(Eager Pipelining)

Pipelining(Cont..)

- **Demand Driven(Lazy Evaluation):**

  System repeatedly requests for tuples from operation at the top of pipeline.

- **Producer Driven(Eager Pipelining):**

  Operations do not wait for request to produce tuples, but generate the tuples eagerly.

Parul® University

DIGITAL LEARNING CONTENT

Query Optimization

- **Query optimization** is the process of choosing the best evaluation plan having lowest cost from the available multiple plans.
- $$\Pi_{Name}\left(\underbrace{\sigma_{Balance<25000}(Account)}_{\text{2 records}} \bowtie \underbrace{(Customer)}_{\text{4 records}}\right)$$

Efficient plan

$$\Pi_{Name}\left(\sigma_{Balance<25000}(\underbrace{Account}_{\text{4 records}} \bowtie \underbrace{Customer}_{\text{4 records}})\right)$$

Customer

CID	ANO	Name
CSE1	A1	Jay
CSE2	A2	Abhi
CSE3	A3	Parth
CSE4	A4	Pratik

Account

ANO	Balance
A1	30000
A2	10000
A3	20000
A4	40000

Query Optimization Approaches

- **Cost Based Optimization (Exhaustive Search Optimization):**
  - In this, it initially generates all possible plans and then select the best plan from it.
  - Its provides the best solution.

- **Heuristic Based Optimization:**
  - These technique is less expensive.
  - To decide optimized query execution plan there are some heuristic rules:

1. To reduce the number of tuples, **Perform selection as early as possible.**
2. To reduce the number of attributes, **Perform projection as early as possible.**
3. Perform most restrictive selection and join operations (i.e. with smallest result size) before other similar operations.

Transformation of relational expressions

- Two relational algebra expressions are said to be **equivalent** if the two expressions generate the same set of tuples on every legal database instance.

- An **equivalence rule** says that expressions of two forms are equivalent.
  - Can replace expression of first by second, or vice versa.

Transformation of relational

- **For Example,**

Customer

CID	ANO	Name
CSE1	A1	Jay
CSE2	A2	Abhi
CSE3	A3	Parth
CSE4	A4	Pratik

Account

ANO	Balance
A1	30000
A2	10000
A3	20000
A4	40000

Π_{Name} ($\sigma_{Balance<25000}$ (Account) $\bowtie$ (Customer)) $\equiv$ Π_{Name} ($\sigma_{Balance<25000}$ (Account $\bowtie$ Customer))

Customer

Name
Meet
Jay

Equivalence Rules

1. **Conjunctive(Combined) selection operations can be deconstructed** into sequence of individual selections. This is known as **Cascade of σ.**

Customer

CID	ANO	Name	Balance
CS1	1	Jay	30000
CS2	2	Abhi	10000
CS3	3	Parth	20000
CS4	4	Pratik	40000

$\sigma_{ANO<3 \wedge Balance<20000}$ (Customer)

$\equiv$

$\sigma_{ANO<3}$ ($\sigma_{Balance<20000}$ (Customer))

OUTPUT

Output

CID	ANO	Name	Balance
CSE2	2	Abhi	10000

$$\sigma_{\theta1 \wedge \theta2} (E) = \sigma_{\theta1} (\sigma_{\theta2} (E))$$

Equivalence Rules

2. **Selection operations are commutative**

$$\sigma_{\theta 1} (\sigma_{\theta 2} (E)) = \sigma_{\theta 2} (\sigma_{\theta 1} (E))$$

2. If **many projection used in expression** then only **the last in a sequence of projection operations is required. So Omit all other projection operation.**

$$\Pi_{L1} (\Pi_{L2} (\ldots(\Pi_{Ln} (E))\ldots)) = \Pi_{L1} (E)$$

2. **Selection operation can be combined with Cartesian products and theta joins.**

$$\sigma_{\theta} (E1 \bowtie E2)) = E1 \bowtie_{\theta} E2$$

$$\sigma_{\theta 1} (E1 \bowtie_{\theta 2} E2)) = E1 \bowtie_{\theta 1 \wedge \theta 2} E2$$

Equivalence Rules

5. Theta-join operations (and natural joins) are commutative.

$$E1 \bowtie_{\theta} E2 = E2 \bowtie_{\theta} E1$$

5. Natural join operations are associative

$$(E1 \bowtie E2) \bowtie E3 = E1 \bowtie (E2 \bowtie E3)$$

Equivalence Rules

7. The selection operation distributes over the theta join operation under the following two conditions:

(a) When all the attributes in θ_0 involve only the attributes of one of the expressions (E_1) being joined.

$$\sigma_{\theta 0}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta 0}(E_1)) \bowtie_\theta E_2$$

(b) When θ_1 involves only the attributes of E_1 and θ_2 involves only the attributes of E_2.

$$\sigma_{\theta 1 \wedge \theta 2}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta 1}(E_1)) \bowtie_\theta (\sigma_{\theta 2}(E_2))$$

Equivalence Rules

8. **The projection operation distributes over the theta join operation as follows:**

(a) if θ involves only attributes from $L_1 \cup L_2$:

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_{\vartheta} E_2) = (\Pi_{L_1}(E_1)) \bowtie_{\vartheta} (\Pi_{L_2}(E_2))$$

(b) Consider a join $E_1 \bowtie_\theta E_2$.

- Let L_1 and L_2 be sets of attributes from E_1 and E_2, respectively.
- Let L_3 be attributes of E_1 that are involved in join condition θ, but are not in $L_1 \cup L_2$, and
- Let L_4 be attributes of E_2 that are involved in join condition θ, but are not in $L_1 \cup L_2$.

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_{\vartheta} E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_{\vartheta} (\Pi_{L_2 \cup L_4}(E_2)))$$

Equivalence Rules

9. **The set operations union and intersection are commutative**

$$E_1 \cup E_2 = E_2 \cup E_1$$
$$E_1 \cap E_2 = E_2 \cap E_1$$

Note: set difference is not commutative

10. **Set union and intersection are associative.**

$$(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$$
$$(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$$

Equivalence Rules

11. The selection operation distributes over ∪, ∩ and –.

$$\sigma_\theta (E_1 - E_2) = \sigma_\theta (E_1) - \sigma_\theta(E_2)$$

and similarly for ∪ and ∩ in place of –

Also: $\sigma_\theta (E_1 - E_2) = \sigma_\theta(E_1) - E_2$

and similarly for ∩ in place of –, but not for ∪

12. The projection operation distributes over union

$$\Pi_L(E_1 \cup E_2) = (\Pi_L(E_1)) \cup (\Pi_L(E_2))$$

Sorting

- We can build an index on the relation, so use the index to read the relation in sorted order. May cause one disk block access for every tuple.

- For relations that slot(fit) in memory, techniques like quicksort may be used. For relations that don't slot in memory, external sort-merge is used.

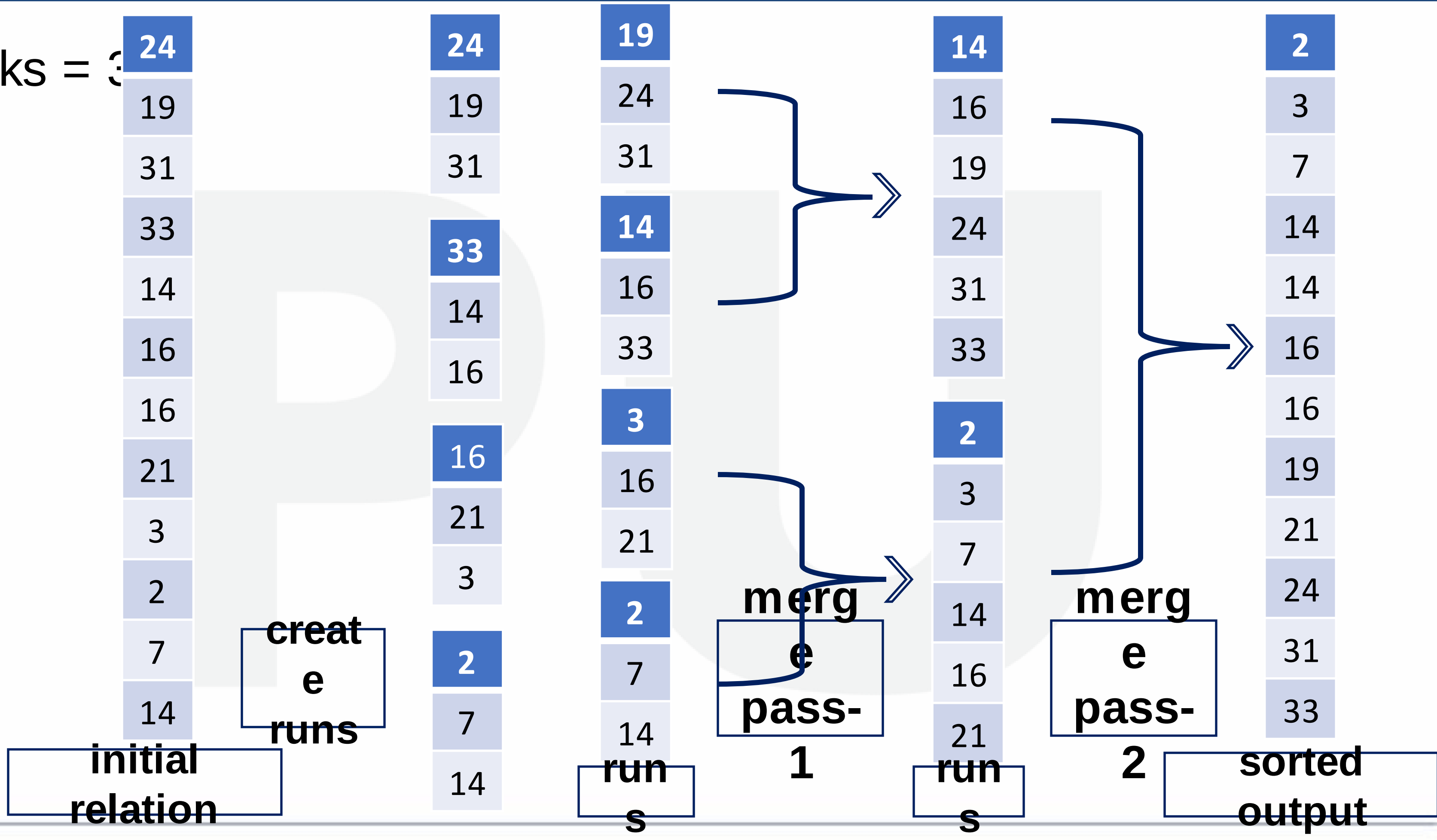
External Sort-Merge

Let M denote memory size (in pages).

1. Create sorted runs as follows. Let i be 0 initially. Repeatedly do the following till the end of the relation:
   (a) Read M blocks of relation into memory
   (b) Sort the in-memory blocks
   (c) Write sorted data to run Ri; increment i.
2. Merge the runs; suppose for now that i < M. In a single merge step, use i blocks of memory to buffer input runs, and 1 block to buffer output. Repeatedly do the following until all input buffer pages are empty:
   (a) Select the first record in sort order from each of the buffers
   (b) Write the record to the output
   (c) Delete the record from the buffer page; if the buffer page is empty, read the next block (if any) of the run into the buffer

Example - External Sorting using Sort-Merge

- Blocks = 3



External Sort-Merge(Conti..)

- If i ≥ M, several merge passes are required.
  - In each pass, contiguous groups of M − 1 runs are merged.
  - A pass reduces the number of runs by a factor of M − 1, and creates runs longer by the same factor.
  - Repeated passes are performed till all runs have been merged into one.
- **Cost analysis:**
  - Disk accesses for initial run creation as well as in each pass is $2b_r$ (except for final pass, which doesn't write out results)
  - Total number of merge passes required: **$[\log_{M-1}(b_r/M)]$.**

    Thus total number of disk accesses for external sorting:

    $$\mathbf{b_r(2[\log_{M-1}(b_r/M)] + 1)}$$

Join Strategies

- Several different algorithms to implement joins
  1. Nested-loop join
  2. Block nested-loop join
  3. Indexed nested-loop join
  4. Merge-join
  5. Hash-join

Nested-Loop join

- To compute the join $r \bowtie_\theta s$

```
for each tuple $t_r$ in r do begin
    for each tuple $t_s$ in s do begin
            test pair ($t_r$, $t_s$) to see if they satisfy the join condition $\theta$
            if they do, add $t_r \cdot t_s$ to the result.
    end
end
```

  - **r** is called the **outer relation** and **s** the **inner relation** of the join
  - Requires no indices and can be used with any kind of join condition
  - Expensive since it examines every pair of tuples in the two relations

Nested-Loop join (conti..)

- Assuming **worst case** memory availability and the following given statistics for the relations customer and depositor
    - Number of records of customer: 10,000 ($n_{customer}$)
    - Number of records of depositor: 5,000 ($n_{depositor}$)
    - Number of blocks of customer: 400 ($b_{customer}$)
    - Number of blocks of depositor: 100 ($b_{depositor}$)

- **Estimate the cost**
    1. with depositor as outer relation
    2. with customer as outer relation

Nested-Loop join (conti..)

- Assuming **worst case** memory availability and the following given statistics for the relations customer and depositor
  - Number of records of customer: 10,000 ($n_{customer}$)
  - Number of records of depositor: 5,000 ($n_{depositor}$)
  - Number of blocks of customer: 400 ($b_{customer}$)
  - Number of blocks of depositor: 100 ($b_{depositor}$)

- **Estimate the cost**
  1. with depositor as outer relation

     No. of blocks access $= n_{depositor} * b_{customer} + b_{depositor}$

     $= 5000 * 400 + 100$

Nested-Loop join (conti..)

- Assuming **worst case** memory availability and the following given statistics for the relations customer and depositor
  - Number of records of customer: 10,000 ($n_{customer}$)
  - Number of records of depositor: 5,000 ($n_{depositor}$)
  - Number of blocks of customer: 400 ($b_{customer}$)
  - Number of blocks of depositor: 100 ($b_{depositor}$)

- **Estimate the cost**
  1. with customer as outer relation

No. of blocks access $= n_{customer} * b_{depositor} + b_{customer}$

$= 10000 * 100 + 400$

Nested-Loop join (conti..)

- Assuming **best case** memory availability and the following given statistics for the relations customer and depositor
  - Number of records of customer: 10,000 ($n_{customer}$)
  - Number of records of depositor: 5,000 ($n_{depositor}$)
  - Number of blocks of customer: 400 ($b_{customer}$)
  - Number of blocks of depositor: 100 ($b_{depositor}$)

- **Estimate the cost**
  1. with customer as outer relation

     No. of blocks access $= b_{depositor} + b_{customer}$

     $= 100 + 400$

     $= 500$

Cost of computing for all joins

- **P is the outer relation** and **Q is the inner relation** of the join.

  Number of records of P: (N_P)
  Number of records of Q: (N_Q)
  Number of blocks of P: (B_P)
  Number of blocks of Q: (B_Q)

Join	Worst Case	Best Case
Nested-Loop Join	$B_P + N_P * B_Q$	$B_P + B_Q$
Block Nested-Loop Join	$B_P + B_P * B_Q$	$B_P + B_Q$
Index Nested-Loop Join	$B_P + N_P * c$	
Merge Join	$B_P + B_Q$	
Hash-Join	$3 * (B_P + B_Q)$	

c is the cost of a single selection on Q using the join condition.

DIGITAL LEARNING CONTENT

References

[1] Abraham Silberschatz, Henry F. Korth and S. Sudarshan, Database System Concepts, McGraw-Hill Education (Asia), Seventh Edition, 2019.

[2] C. J. Date, A. Kannan and S. Swamynathan, An Introduction to Database Systems, Pearson Education, Eighth Edition, 2009.

[3] Database Management Systems, CSE, DIET, https://www.darshan.ac.in/DIET/CE/GTU-Computer-Engineering-Study-Material

[4] Database management systems by Raghu Ramakrishnan and Johannes Gehrke http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed.html

[5] Database management system tutorial, https://www.tutorialspoint.com/dbms/index.htm

[6] Vishal Hatmode, Sonali Rangdale, 2014, Query Optimization Based on

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.i