PRINCIPAL OF DATABASE MANAGEMENT SYSTEMS

SUBJECT CODE: CE 410304

B.E. 3rd Semester

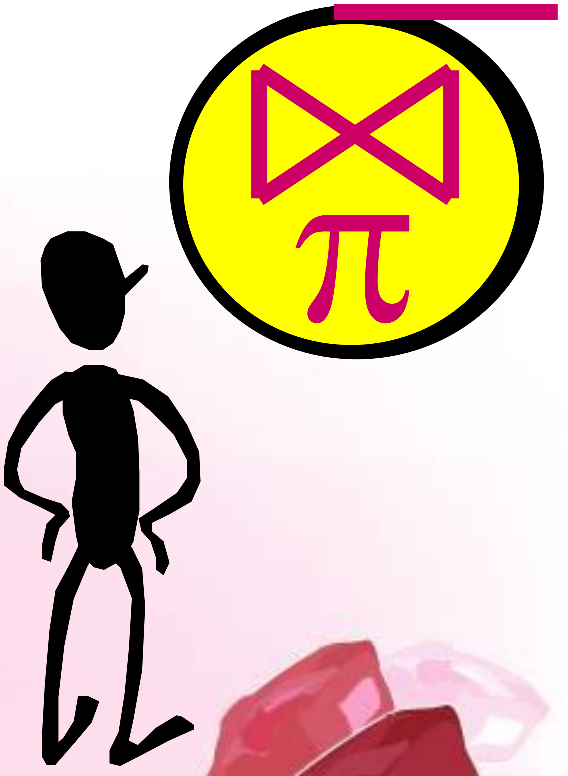
BY.

Prof. Sejal Thakkar

Unit2

- **Relational Model:**
- Structure of relational databases, relational model, relations, relational integrity, Domains, Relational Algebra(fundamental and extended) and query
- **Relation database design: Functional Dependency – definition, trivial and non-trivial FD, closure of FDset, closure of attributes, irreducible set of FD, Normalization – 1Nf, 2NF,3NF, composition using FD- dependency preservation, BCNF, Multivalued dependency, 4NF, Join dependency and 5NF**

Relational Model
Unit 2



Relational Model

- Relational model is a collection of tables representing an E-R database schema. For each entity set and for each relationship set in the database, there is a unique table having the name of the corresponding entity set or relationship set. Each table has multiple columns which correspond to attributes in E-R schema.

- A relational model is a tabular representation of ER model. The ER diagram represents the conceptual level of
- database design intended as a description of real-world entities while a relational schema is at the logical level

  of database design.

In relational model

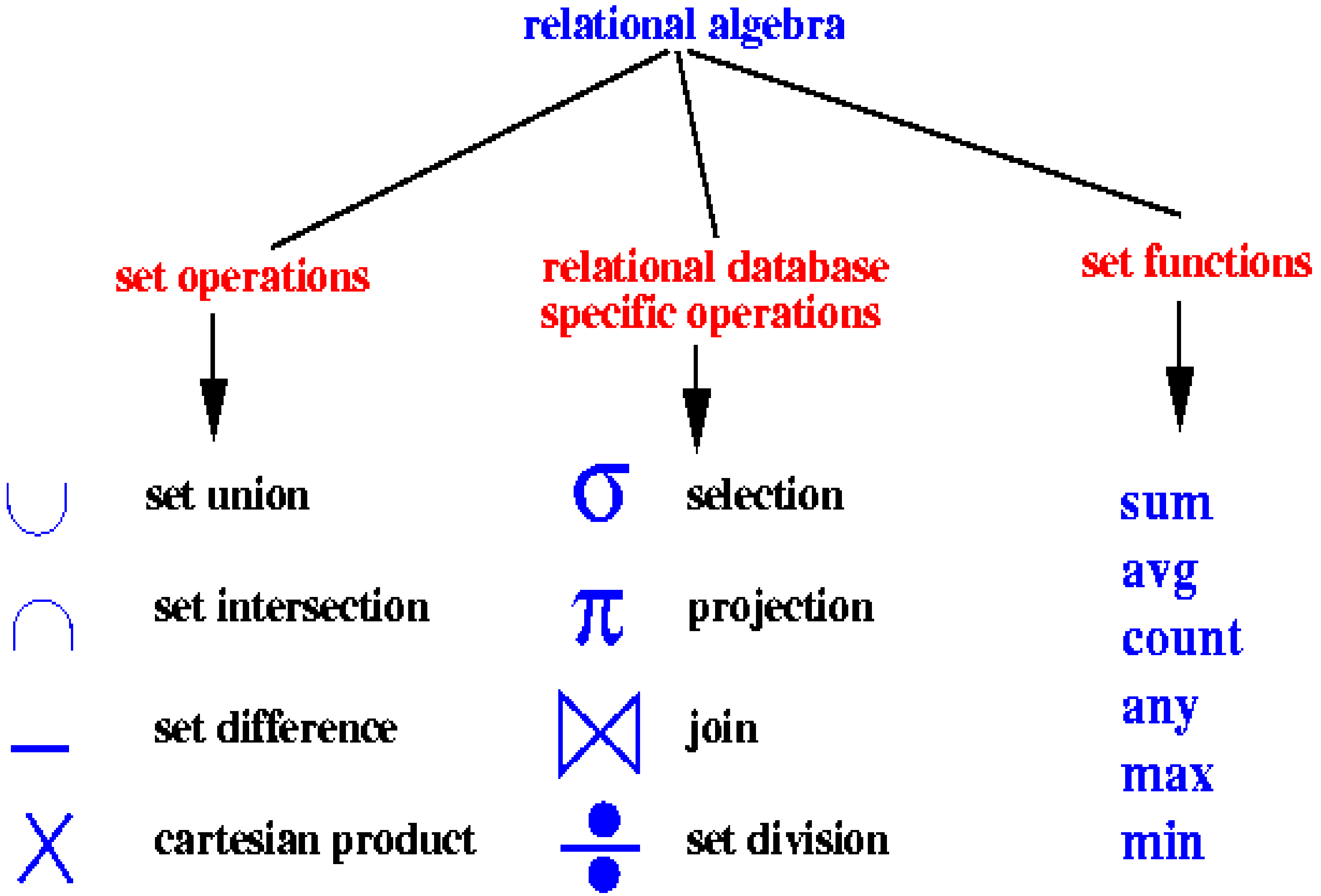
---Table represents a schema/relation
---row represents a relational instance (also called tuple)
---column represents an attribute Column headers are known as *attributes.*
---cardinality represents number of rows
---degree represents number of columns

	Name Major GPA	Name Major GPA	Name Major GPA
1234	**John**	**CS**	**2.8**
5678	**Mary**	**EE**	**3.6**

Degree = 4

Cardinality=2

Relational Algebra



Relational Algebra: 5 Basic Operations

- *Selection* (σ) Selects a subset of *rows* from relation (horizontal).
- *Projection* (π) Retains only wanted *columns* from relation (vertical).
- *Cross-product* (x) Allows us to combine two relations.
- *Set-difference* (–) Tuples in r1, but not in r2.
- *Union* ($\cup$) Tuples in r1 and/or in r2.
- Intersection $\cap$ Tuples in r1 and r2 both

Example Instances

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Projection

- – Column wise selection..
- – Vertical selection
- – Denoted by pi
- – Unary operation
- Examples: ;

$\pi_{age}(S2)$ $\pi_{sname,rating}(S2)$

Projection

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$$\pi_{sname,rating}(S2)$$

age
35.0
55.5

$$\pi_{age}(S2)$$

Selection (σ)

- Selects rows that satisfy *selection condition*.
- Used to find horizontal subset of relation.
- Denoted by sigma
- Unary operation

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

$\sigma_{rating>8}(S2)$

sname	rating
yuppy	9
rusty	10

$\pi_{sname,rating}(\sigma_{rating>8}(S2))$

Union

- All of these operations take two input relations, which must be *union-compatible*:
  – Same number of fields.
  – Corresponding' fields have the same datatype.

duplicate elimination required?

Union

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

Intersection

- Used to find common tuples between two relations.
- It is denoted by $\cap$

Intersection

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$$S1 \cap S2$$

Set-Difference

- It is a binary operation
- Which is used to find tuples that are present in one relation but not in other relation.

- Denoted by −

Set Difference

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
44	guppy	5	35.0

$S2 - S1$

Cross-Product

- Cartesian product is a binary operation which is used to combine information of any two relations.
- Relation R1 is having m tuple and relation R2 is having n tuples then R1 x R2 hase m x n tuples
- Denoted by X
- R1 x S1 : Each row of R1 paired with each row of S1.

Cross Product Example

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1

R1 X S1 =

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/9
22	dustin	7	45.0	58	103	11/12/9
31	lubber	8	55.5	22	101	10/10/9
31	lubber	8	55.5	58	103	11/12/9
58	rusty	10	35.0	22	101	10/10/9
58	rusty	10	35.0	58	103	11/12/9

Division

A/*B* contains all *x* tuples such that for ***every*** *y* tuple in *B*, there is an *xy* tuple in *A*.

Symbol is /

Examples of Division A/B

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1

pno
p2
p4

B2

pno
p1
p2
p4

B3

sno
s1
s2
s3
s4

A/B1

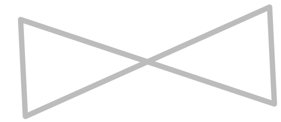
sno
s1
s2
s3
s4

A/B2

sno
s1
s2
s3
s4

A/B3

JOIN

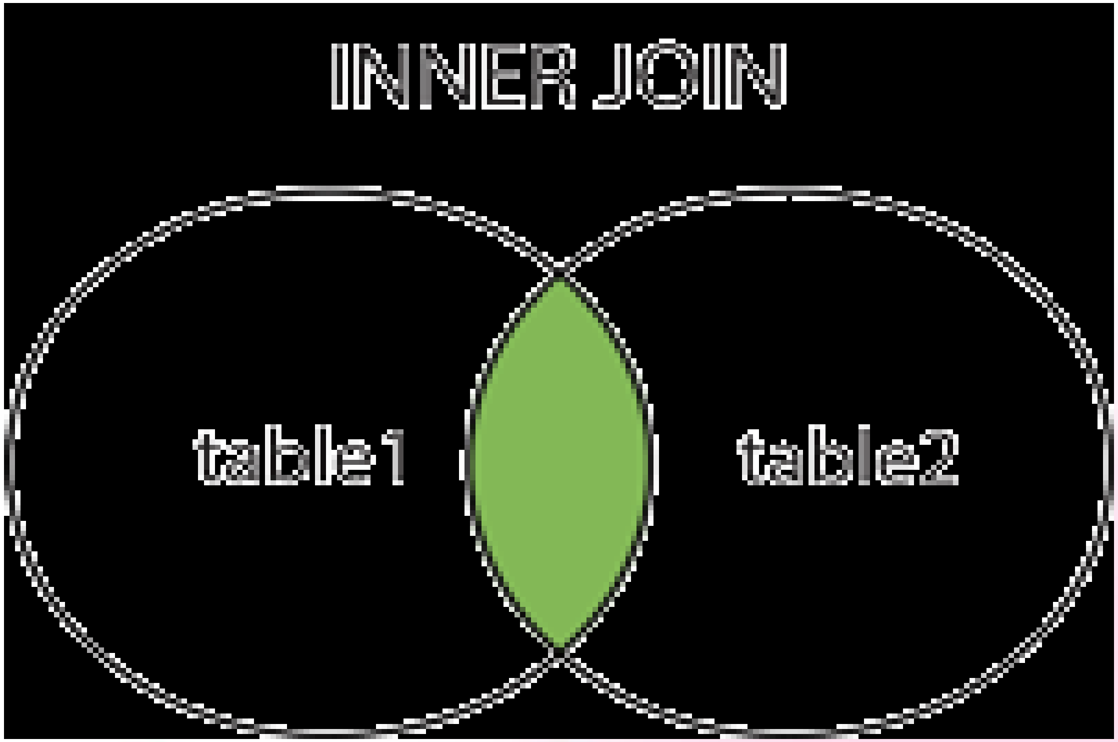


- A SQL **join** clause combines records from two or more tables in a database.
- It creates a set that can be saved as a table or used as is.
- A JOIN is a means for combining fields from two tables by using values common to each.
- ANSI standard SQL specifies four types of JOINs: INNER, OUTER, LEFT, and RIGHT

Employee Table	
LastName	**DepartmentID**
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
John	NULL

Department Table	
DepartmentID	**DepartmentName**
31	Sales
33	Engineering
34	Clerical
35	Marketing

INNER JOIN



- SELECT *
- FROM   employee, department
- WHERE employee.DepartmentID = department.DepartmentID;

Employee Table	
LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
John	NULL

Department Table	
DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Robinson	34	Clerical	34
Jones	33	Engineering	33
Smith	34	Clerical	34
Steinberg	33	Engineering	33
Rafferty	31	Sales	31

Cross join

- CROSS JOIN returns the Cartesian product of rows from tables in the join. In other words, it will produce rows which combine each row from the first table with each row from the second table.
- Example of an explicit cross join:
- SELECT *
- FROM   employee CROSS JOIN department;
- Example of an implicit cross join:
- SELECT *
- FROM   employee, department;

Employee Table	
LastName	**DepartmentID**
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
John	NULL

Department Table	
DepartmentID	**DepartmentName**
31	Sales
33	Engineering
34	Clerical
35	Marketing

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Rafferty	31	Sales	31
Jones	33	Sales	31
Steinberg	33	Sales	31
Smith	34	Sales	31
Robinson	34	Sales	31
John	NULL	Sales	31
Rafferty	31	Engineering	33
Jones	33	Engineering	33
Steinberg	33	Engineering	33
Smith	34	Engineering	33
Robinson	34	Engineering	33
John	NULL	Engineering	33
Rafferty	31	Clerical	34
Jones	33	Clerical	34
Steinberg	33	Clerical	34
Smith	34	Clerical	34
Robinson	34	Clerical	34
John	NULL	Clerical	34
Rafferty	31	Marketing	35

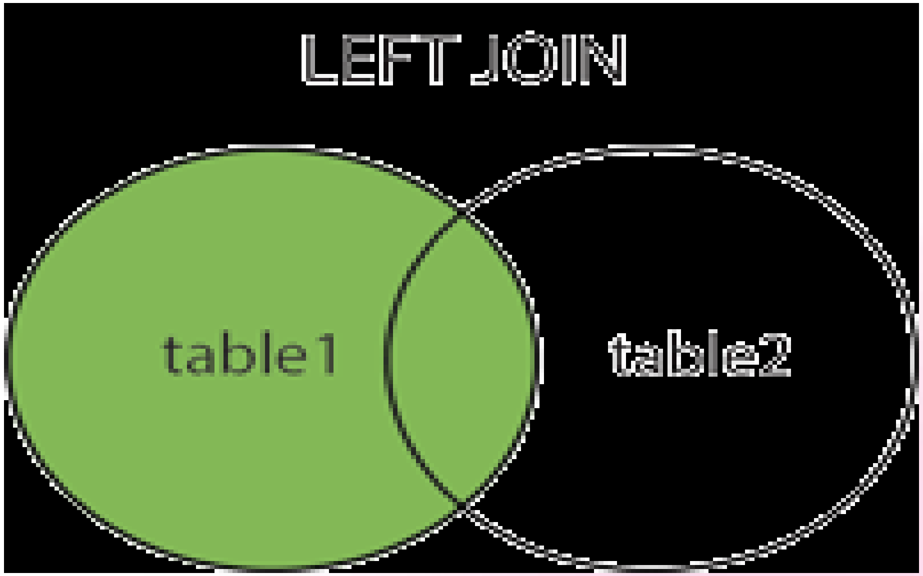
Outer joins

- An **outer join** does not require each record in the two joined tables to have a matching record. The joined table retains each record—even if no other matching record exists. Outer joins subdivide further into left outer joins, right outer joins, and full outer joins, depending on which table(s) one retains the rows from (left, right, or both).

Left outer join

- The result of a *left outer join* (or simply **left join**) for table A and B always contains all records of the "left" table (A), even if the join-condition does not find any matching record in the "right" table (B). This means that if the ON clause matches 0 (zero) records in B, the join will still return a row in the result—but with NULL in each column from B. This means that a **left outer join** returns all the values from the left table, plus matched values from the right table (or NULL in case of no matching join predicate). If the right table returns one row and the left table returns more than one matching row for it, the values in the right table will be repeated for each distinct row on the left table.
- LEFT OUTER JOIN statement can be used as well as (+).

Left outer join



- SELECT * FROM employee LEFT OUTER JOIN department ON employee.DepartmentID = department.DepartmentID;

OR

- SELECT * FROM employee, department WHERE employee.DepartmentID = department.DepartmentID(+)

Employee Table

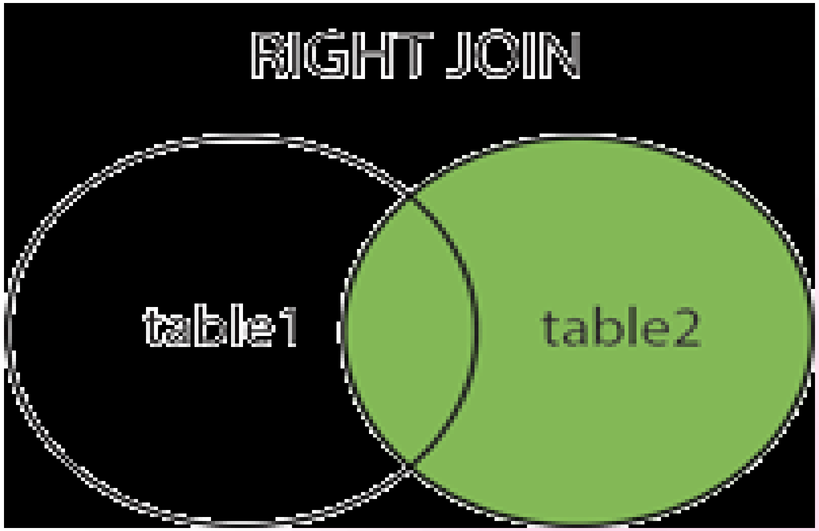
LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
John	NULL

Department Table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Jones	33	Engineering	33
Rafferty	31	Sales	31
Robinson	34	Clerical	34
Smith	34	Clerical	34
John	NULL	NULL	NULL
Steinberg	33	Engineering	33

Right outer join



- Every row from the "right" table (B) will appear in the joined table at least once. If no matching row from the "left" table (A) exists, NULL will appear in columns from A for those records that have no match in B. A right outer join returns all the values from the right table and matched values from the left table (NULL in case of no matching join predicate). For example, this allows us to find each employee and his or her department

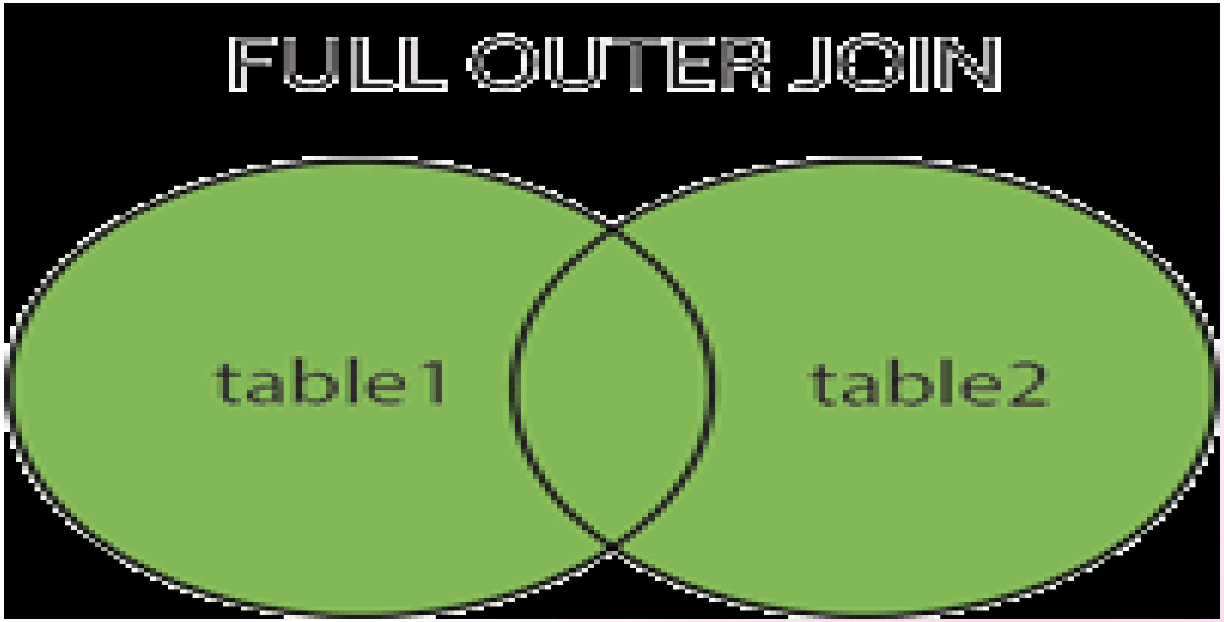
- SELECT *
- FROM employee RIGHT OUTER JOIN department
- ON employee.DepartmentID = department.DepartmentID;

Employee Table	
LastName	**DepartmentID**
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
John	NULL

Department Table	
DepartmentID	**DepartmentName**
31	Sales
33	Engineering
34	Clerical
35	Marketing

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Steinberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	Marketing	35

Full outer join



- Conceptually, a **full outer join** combines the effect of applying both left and right outer joins. Where records in the FULL OUTER JOINed tables do not match, the result set will have NULL values for every column of the table that lacks a matching row. For those records that do match, a single row will be produced in the result set (containing fields populated from both tables).
- For example, this allows us to see each employee who is in a department and each department that has an employee, but also see each employee who is not part of a department and each department which doesn't have an employee.

- SELECT *
- FROM employee
- FULL OUTER JOIN department
- ON employee.DepartmentID = department.DepartmentID;

Employee Table	
LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
John	NULL

Department Table	
DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
John	NULL	NULL	NULL
Steinberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	*Marketing*	*35*

- What is data independence ? Explain the difference between physical and logical data independence with example.
- What are the responsibilities of a DBA ?
- What is join ? Explain various type of joins with example
- Describe various disadvantages of file system compare to Data base management system.

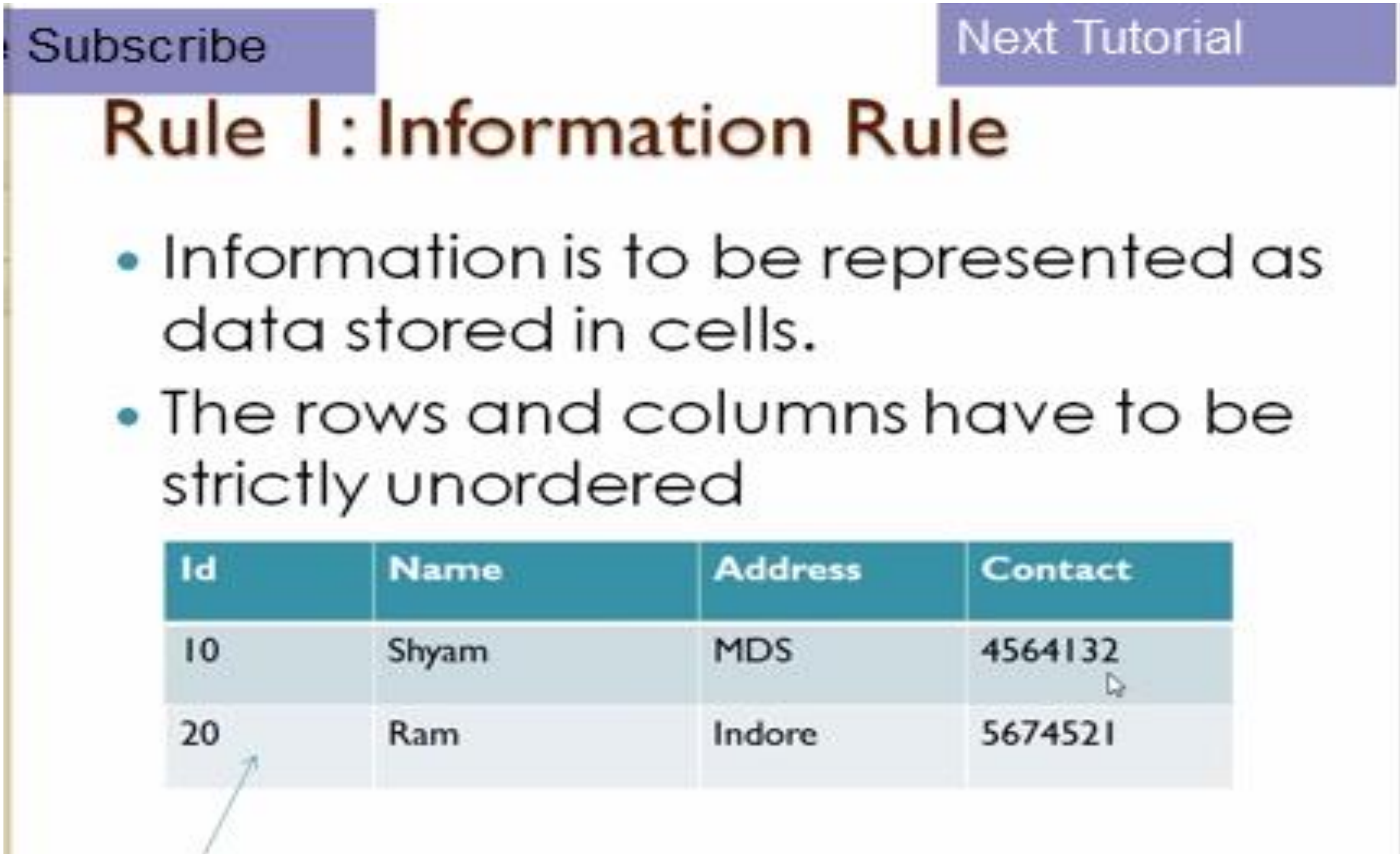
- Explain database system architecture with diagram in detail.
- List the benefits of database approach.
- List relational algebra operators and explain any two with example.

12 Codd's Rule

Rules that a DBMS should follow to be classified as fully relational

- 1985 Dr. E.F. Codd the originator
- Proposed to test DBMSs for confirmation to concept of Codd's Relational model
- Hardly any commercial product follows all

One data in one cell no change in data due to order



Find age of student using roll no pk

Subscribe

Next Tutorial

Rule 2: Guaranteed Access Rule

- All data must be accessible
- Each unique piece of data (atomic value) should be accessible by the combination of :

  TableName +Primary Key (Row) +Attribute (Column)

MySql> Select name from student where id=10;

Rule 3: Systematic Treatment of null Values

- RDBMS must allow each attribute to remain null, Specifically ,it must support a representation of missing information and inapplicable information
- NULLs may mean: Missing data, Not applicable, No value
- Should be handled consistently - Not Zero or Blank
- Primary keys — Not NULL

Rule 4: Active Online Catalog Based on the Relational model

- Data dictionary should be stored as relational tables and accessible through the regular data access language.
- Database dictionary (Catalog) to have description of the Database
- The same query language to be used on catalog as on the application database

Note: SQL is used for both the purpose.

Subscribe

Rule 5: The Data Sublanguage Rule

- One well defined language to provide all manners of access to data
- Example: SQL
- It support data definition, data manipulation, security, integrity constraints and transaction management

Rule 6: The View Updating Rule

- **All views that are theoretically updatable should be updatable**
- View = "Virtual table", temporarily derived from base tables
- Example: If a view is formed from tables, changes to view should be reflected in base tables.

Rule7: High Level Insert, Update and Delete

- The System must support set at a time insert ,update, and delete operations.

- Set operations like Union, Intersection and Minus should be supported

Rule 8: Physical Data Independence

- The physical storage of data should not matter to the system
- If say, some file supporting table was renamed or moved from one disk to another, it should not effect the applications

Rule 10: Integrity Independence

- The database should be able to enforce its own integrity rather than using other programs
- Integrity rules = Filter to allow correct data, should be stored in Data Dictionary
- Key and check constraints, triggers etc should be stored in Data Dictionary

Rule 11: Distribution Subversion

- The distribution of portion of the database to various location should be invisible to the user of the database.
- A database should work properly regardless of its distribution across a network
- This lays foundation of Distributed databases

Subscribe

Rule 12: The Non Subversion Rule

- If low-level access is allowed, it must not bypass security nor integrity rules
- If low level access is allowed to a system it should not be able to subvert or bypass integrity rules to change data
- This may be achieved by some sort of locking or encryption

- **Relational Calculus**
- In contrast to Relational Algebra, Relational Calculus is a non-procedural query language, that is, it tells what to do but never explains how to do it.
- Relational calculus exists in two forms −

- **Tuple Relational Calculus (TRC)**
- Filtering variable ranges over tuples
- **Notation** − {T | Condition}
- Returns all tuples T that satisfies a condition.
- **For example** −
- { T.name | Author(T) AND T.article = 'database' }
- **Output** − Returns tuples with 'name' from Author who has written article on 'database'.
- TRC can be quantified. We can use Existential (∃) and Universal Quantifiers (∀).
- **For example** −
- { R| ∃T ∈ Authors(T.article='database' AND R.name=T.name)}
- **Output** − The above query will yield the same result as the previous one.

- **Domain Relational Calculus (DRC)**
- In DRC, the filtering variable uses the domain of attributes instead of entire tuple values (as done in TRC, mentioned above).
- **Notation –**
- $\{ a_1, a_2, a_3, ..., a_n \mid P (a_1, a_2, a_3, ... ,a_n)\}$
- Where a1, a2 are attributes and **P** stands for formulae built by inner attributes.

- **For example −**
- {< article, page, subject > | ∈ TutorialsPoint ∧ subject = 'database'}
- **Output −** Yields Article, Page, and Subject from the relation TutorialsPoint, where subject is database.
- Just like TRC, DRC can also be written using existential and universal quantifiers. DRC also involves relational operators.
- The expression power of Tuple Relation Calculus and Domain Relation Calculus is equivalent to Relational Algebra.