Database Management Systems

Dr. Shaik Hussain Shaik Ibrahim
Department of CSE
School of Engineering

Malla Reddy University, Hyderabad

Syllabus
UNIT - II

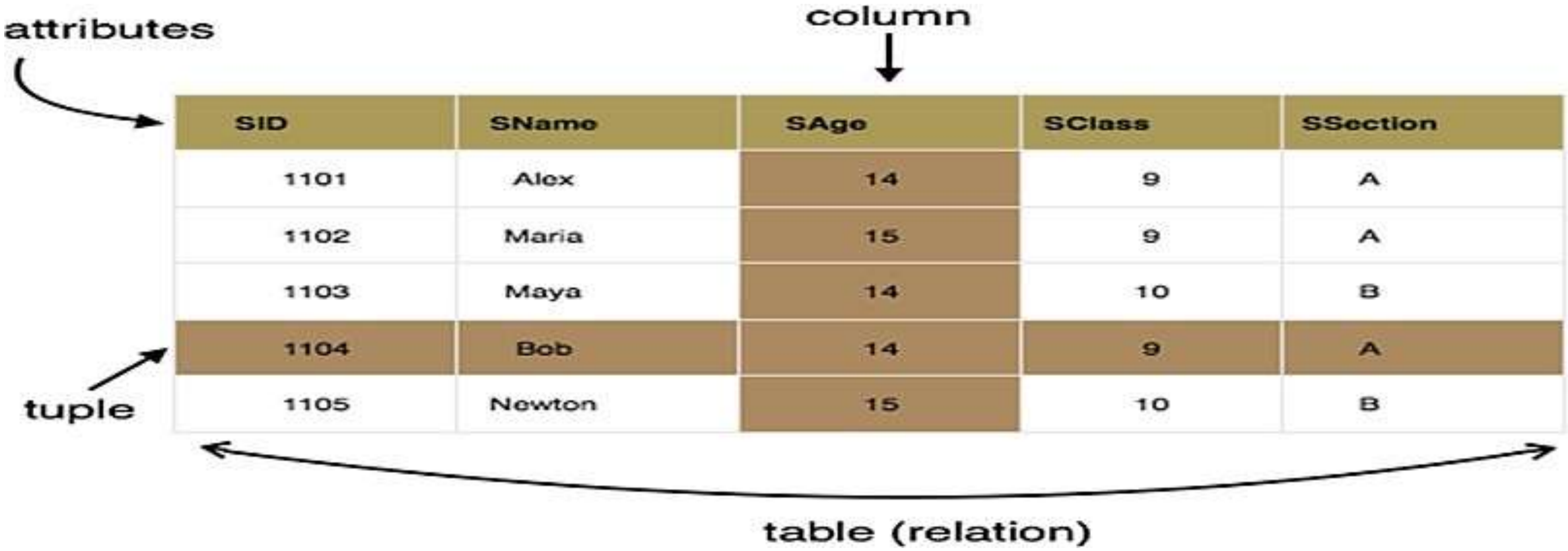
The Relational Model:
Introduction to the Relational Model, Integrity Constraints over Relations, Enforcing Integrity Constraints, Querying Relational Data, Logical Database Design, Introduction to Views, Destroying/Altering Tables and Views.

Relational Algebra And Calculus:
Relational Algebra- Selection and Projection, Set Operations, Renaming, Joins, Division, Examples of Relational Algebra Queries, Relational Calculus- Tuple Relational Calculus, Domain Relational Calculus.

Relational Model

- The main construct for representing data in the relational model is a „relation‟.
- A relation consists of following fields
- **Relation Schema:** The relation schema describes the column heads for the table.
  - The schema specifies the relation‟s name, the name of each field (column, attribute) and the „domain‟ of each field
- **Relation Instance:** This is a table specifying the information.
  - An instance of a relation is a set of „tuples‟, also called „records‟, in which each tuple has the same number of fields as the relation schemas.
  - A relation instance can be thought of as a table in which each tuple is a row and all rows have the same number of fields
- **Degree:** The number of fields is called as „degree‟. This is also called as „arity‟.

Relational Model



SID	SName	SAge	SClass	SSection
1101	Alex	14	9	A
1102	Maria	15	9	A
1103	Maya	14	10	B
1104	Bob	14	9	A
1105	Newton	15	10	B

- *Relations* - a table with columns and rows
- *Attributes* - named columns of the relation are called attributes
- *Domains* - domain is the set of values the attributes are allowed to take (or type of the values)

- **Relational database -** collection of relations with distinct relation names
- **Relational database schema -** collection of schemas for the relations in the database

Integrity Constraints

- An *Integrity Constraint* (IC) is a condition that is specified on a database schema and restricts the data that can be stored in an instance of the database
    - It permits only legal instances to be stored in the database
- If the database instance satisfies all the integrity constraints specified on the database schema it is called a *Legal instance*
- Types of constraints:
    1. Domain Constraints
    2. Key Constraints
    3. Entity Integrity Constraints
    4. Referential Integrity Constraints.

Domain Constraints

- Domain Constraints specify the set of possible values that may be associated with an attribute
- It permits only legal instances to be stored in the database
  - *also prohibit the use of null values for particular attributes*

- Relation schema specifies the domain of each field or column in the relation instance
- Domain constraints specify a condition that each instance of the relation to satisfy
- Values that appear in a column must be drawn from the domain associated with that column
- **Example:**
  - Age cannot be a negative number or a character
  - Student roll number cannot a null value

Key Constraints

- Key Constraint is a statement that a certain minimal subset of the fields of a relation is a unique identifier for a tuple
- Types of keys
  - Candidate Key or Key
  - Super Key
  - Primary Key

Candidate Key

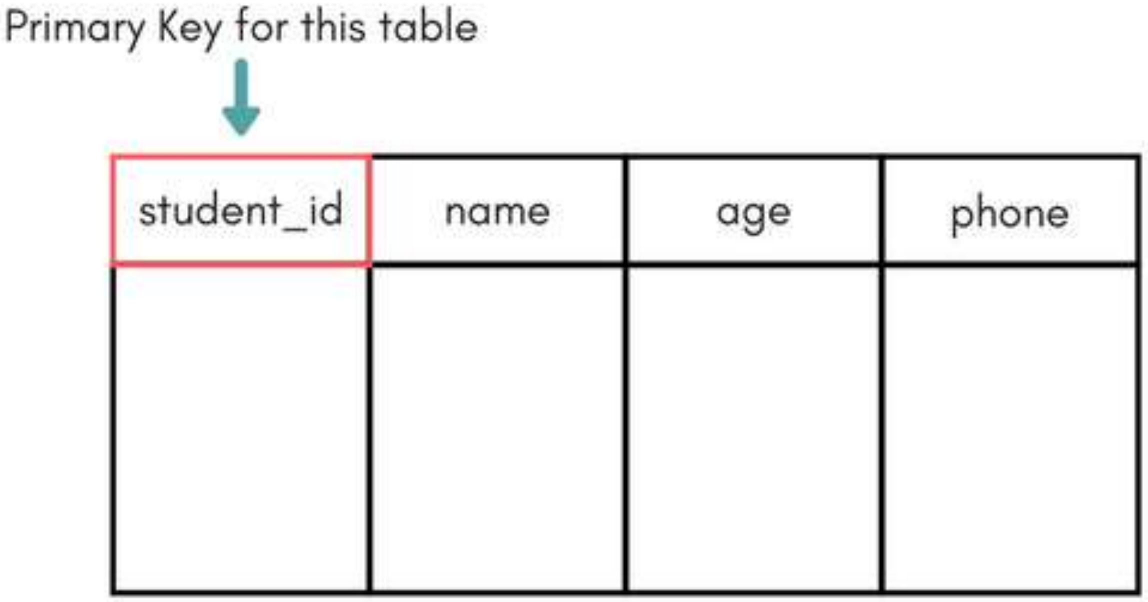
- **Candidate keys** are defined as the minimal set of fields which can uniquely identify each record in a table
  - It is an attribute or a set of attributes that can act as a Primary Key for a table to uniquely identify each record in that table
  - There can be more than one candidate key
  - A candidate key can never be NULL or empty and its value should be unique
  - There can be more than one candidate keys for a table
  - A candidate key can be a combination of more than one columns(attributes)

Super Key

- **Super Key** is defined as a set of attributes within a table that can uniquely identify each record within a table
  - Super Key is a superset of Candidate key
  - Example:
    - *Super key (student_id, name)*
    - *Candiate key student_id*

Primary Key

- **Primary key** is a candidate key that is most appropriate to become the main key for any table
  - It is a key that can uniquely identify each record in a table

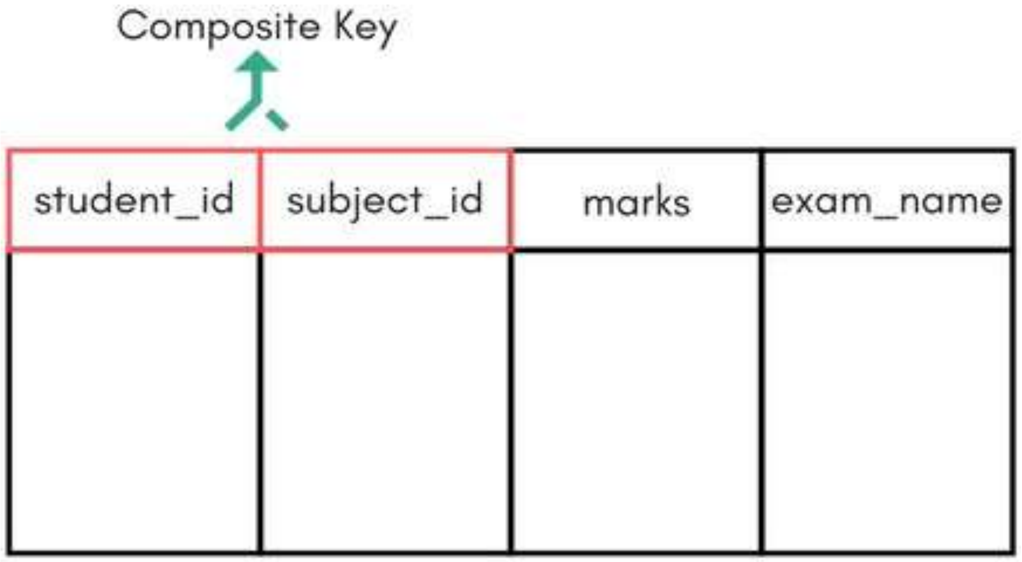


Keys in DBMS

- Super key
- Candidate key
- Primary key
- Composite key
- Secondary or Alternative key

Keys in DBMS

Key that consists of two or more attributes that uniquely identify any record in a table is called **Composite key**



Score Table - To save scores of the student for various subjects.

Secondary or Alternative key

The candidate keys which are not selected as primary key are known as secondary keys or alternative keys.

Entity Integrity Constraint

- **Entity Integrity Constraint** states that no primary key value can be null
- The primary key value is used to identify individual tuples in a relation
- Having null values for the primary key implies that we cannot identify some tuples
- NOTE: Key Constraints, Entity Integrity Constraints are specified on individual relations. PRIMARY KEYS comes under this.

Referential Integrity Constraint

- **Referential Integrity Constraint** is specified between 2 relations and is used to maintain the consistency among tuples of the 2 relations.
- Informally, the referential integrity constraint states that „a tuple in 1 relation that refers to another relation must refer to an existing tuple in that relation.
- We can diagrammatically display the referential integrity constraints by drawing a directed arc from each foreign key to the relation it references.

Views or (Virtual Table)

- A view is a virtual table based on the result-set of an SQL statement
  - *Contains rows and columns, just like a real table*
- Fields in a view are fields from one or more real tables in the database
  - *Views are created by selecting fields from one or more tables in a database*
- Views are a logical virtual table created by “select query”
  - *they are not stored in the memory – need to write a query every time*

- *Example:*
  - *create view <view_name> as select col 1, col 2,..col n from <table_name>where <condition>;*
  - *create view species_pet as select species, gender, name from pet where gender='f';*
  - *select * from species_pet;*

Creating Views

- Creating View from a single table:
  - *CREATE VIEW Details AS SELECT NAME, ADDRESS FROM SD WHERE S_ID < 5;*
- To see the data in the View, we can query the view in the same manner as we query a table.
  - *SELECT * FROM Details;*

- Creating View from multiple tables:
  - *CREATE VIEW Marks AS SELECT SD.NAME, SD.ADDRESS, SM.MARKS FROM SD, SM WHERE SD.NAME = SM.NAME;*

Relational Algebra & Relational Calculus

- **Query language -** used to store and retrieve data from database
- Two types of query language:

1. Procedural Query language
2. Non-procedural query language

Relational Algebra & Relational Calculus

- **Procedural Query language**
  - *User instructs the system to perform a series of operations to produce the desired results*
  - *Users tells what data to be retrieved from database and how to retrieve it*
- **Non-procedural query language**
  - *User instructs the system to produce the desired result without telling the step by step process*
  - *Users tells what data to be retrieved from database but **doesn't** tell how to retrieve it.*
- Relational algebra and relational calculus are theoretical/mathematical systems for query language - not the practical implementation
- SQL is a practical implementation of relational algebra and relational calculus

Relational Algebra & Relational Calculus

- Relational Algebra is procedural query language, which takes Relation as input and generate relation as output
- Provides *theoretical foundation* for relational databases and SQL
- Describes a *step-by-step procedure* for computing the desired answer
- Basis for implementing and optimizing queries in the query processing and optimization modules
- Some of its concepts *are incorporated into the SQL* standard query language for RDBMS

Relational Algebra – Operations

Basic/Fundamental Operations:

1. Select (σ)
2. Project (∏)
3. Union (∪)
4. Set Difference (-)
5. Cartesian product (X)
6. Rename (ρ)

Derived Operations:

1. Natural Join (⋈)
2. Left, Right, Full outer join (⟕, ⟖, ⟗)
3. Intersection (∩)
4. Division (†)

Relational Algebra – Basic Operations

Basic/Fundamental Operations:

1. Select (σ) – WHERE in SQL (based on *condition)*
2. Project (∏) – SELECT in SQL
3. Union (U) – select all the rows from two tables – displayed only once
4. Intersection Operator (∩) – select common rows from two tables
5. Set Difference (-) – rows that are present in one Table but not present in Second table.
6. Cartesian product (X) – combines each tuple of first relation R1 with the each tuple of second relation R2
7. Rename (ρ) – used to rename a relation or an attribute of a relation

Select (σ) Operation

```
σ Condition/Predicate(Relation/Table name)
```

```
Table: CUSTOMER
---------------

Customer_Id     Customer_Name     Customer_City
-----------     -------------     -------------
C10100           Steve             Agra
C10111           Raghu             Agra
C10115           Chaitanya         Noida
C10117           Ajeet             Delhi
C10118           Carl              Delhi
```

Query:

```
σ Customer_City="Agra" (CUSTOMER)
```

Output:

```
Customer_Id   Customer_Name   Customer_City
-----------   -------------   -------------
C10100        Steve           Agra
C10111        Raghu           Agra
```

Project (∏) Operation

```
∏ column_name1, column_name2, ...., column_nameN(table_name)
```

```
Table: CUSTOMER
---------------

Customer_Id     Customer_Name     Customer_City
-----------     -------------     -------------
C10100           Steve             Agra
C10111           Raghu             Agra
C10115           Chaitanya         Noida
C10117           Ajeet             Delhi
C10118           Carl              Delhi
```

Query:

```
∏ Customer_Name, Customer_City (CUSTOMER)
```

Output:

```
Customer_Name     Customer_City
-------------     -------------
Steve             Agra
Raghu             Agra
Chaitanya         Noida
Ajeet             Delhi
Carl              Delhi
```

Union (U) Operation

Table 1: COURSE

```
Course_Id  Student_Name   Student_Id
---------  ------------   ----------
C101        Aditya          S901
C104        Aditya          S901
C106        Steve           S911
C109        Paul            S921
C115        Lucy            S931
```

Table 2: STUDENT

```
Student_Id     Student_Name   Student_Age
------------   ----------     -----------
S901           Aditya         19
S911           Steve          18
S921           Paul           19
S931           Lucy           17
S941           Carl           16
S951           Rick           18
```

```
table_name1 ∪ table_name2
```

Query:

```
∏ Student_Name (COURSE) ∪ ∏ Student_Name (STUDENT)
```

Output:

```
Student_Name
------------
Aditya
Carl
Paul
Lucy
Rick
Steve
```

Intersection Operator (∩) Operation

Table 1: COURSE

Course_Id	Student_Name	Student_Id
C101	Aditya	S901
C104	Aditya	S901
C106	Steve	S911
C109	Paul	S921
C115	Lucy	S931

Table 2: STUDENT

Student_Id	Student_Name	Student_Age
S901	Aditya	19
S911	Steve	18
S921	Paul	19
S931	Lucy	17
S941	Carl	16
S951	Rick	18

```
table_name1 ∩ table_name2
```

Query:

```
∏ Student_Name (COURSE) ∩ ∏ Student_Name (STUDENT)
```

Output:

Student_Name
Aditya
Steve
Paul
Lucy

Set Difference (-) Operation

Table 1: COURSE

Course_Id	Student_Name	Student_Id
C101	Aditya	S901
C104	Aditya	S901
C106	Steve	S911
C109	Paul	S921
C115	Lucy	S931

Table 2: STUDENT

Student_Id	Student_Name	Student_Age
S901	Aditya	19
S911	Steve	18
S921	Paul	19
S931	Lucy	17
S941	Carl	16
S951	Rick	18

```
table_name1 - table_name2
```

Query:

Lets write a query to select those student names that are present in STUDENT table but not present in COURSE table.

```
∏ Student_Name (STUDENT) - ∏ Student_Name (COURSE)
```

Output:

Student_Name
Carl
Rick

Cartesian product (X) Operation

```
R1 X R2
```

```
R X S
```

Table 1: R

Col_A	Col_B
AA	100
BB	200
CC	300

Table 2: S

Col_X	Col_Y
XX	99
YY	11
ZZ	101

Output:

Col_A	Col_B	Col_X	Col_Y
AA	100	XX	99
AA	100	YY	11
AA	100	ZZ	101
BB	200	XX	99
BB	200	YY	11
BB	200	ZZ	101
CC	300	XX	99
CC	300	YY	11
CC	300	ZZ	101

Rename (ρ) Operation

ρ(new_relation_name, old_relation_name)

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

Query:

```
ρ(CUST_NAMES, ∏(Customer_Name)(CUSTOMER))
```

Output:

CUST_NAMES
Steve
Raghu
Chaitanya
Ajeet
Carl

Relational Algebra – Derived Operations

Derived Operations:

1. Natural Join (⋈)

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

3. Intersection (∩)

4. Division (†)

Natural Join (⋈)

- Natural join is a binary operator → (Cross Product + Condition)
- Will result set of all combination of tuples where they have equal common attribute.

$\prod$EMP_NAME, SALARY (EMPLOYEE ⋈ SALARY)

EMP_CODE	EMP_NAME
101	Stephan
102	Jack
103	Harry

EMP_CODE	SALARY
101	50000
102	30000
103	25000

EMP_NAME	SALARY
Stephan	50000
Jack	30000
Harry	25000

Outer Join

- The outer join operation is an extension of the join operation. It is used to deal with missing information
- Types of Outer join
  - Left (⟕)
  - Right (⟖)
  - Full outer join (⟗)

Outer Join - Left (⟕)

- Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In the left outer join, tuples in R have no matching tuples in S.
  - EMPLOYEE ⟕ FACT_WORKERS

EMPLOYEE

EMP_NAME	STREET	CITY
Ram	Civil line	Mumbai
Shyam	Park street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

FACT_WORKERS

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Shyam	Wipro	20000
Kuber	HCL	30000
Hari	TCS	50000

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL

Outer Join - Right (⟖)

- Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In right outer join, tuples in S have no matching tuples in R.
  - EMPLOYEE ⟖ FACT_WORKERS

EMPLOYEE

EMP_NAME	STREET	CITY
Ram	Civil line	Mumbai
Shyam	Park street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

FACT_WORKERS

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Shyam	Wipro	20000
Kuber	HCL	30000
Hari	TCS	50000

EMP_NAME	BRANCH	SALARY	STREET	CITY
Ram	Infosys	10000	Civil line	Mumbai
Shyam	Wipro	20000	Park street	Kolkata
Hari	TCS	50000	Nehru street	Hyderabad
Kuber	HCL	30000	NULL	NULL

Outer Join - Full (⟗)

- Full outer join is like a left or right join except that it contains all rows from both tables.
- In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name.
    - EMPLOYEE ⟗ FACT_WORKERS

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL
Kuber	NULL	NULL	HCL	30000

Natural Join

- Find Employees who are working in Dept
  - Select E_name from Emp, Dept where Emp.E_No=Dept.E-No

E_No	E-Name	Add
1	Ram	Delhi
2	Ravi	Hyd
3	Raju	Chennai
4	Raghu	Bangalore

Dept_no	D_Name	E_No
D1	HR	1
D2	IT	2
D3	Finance	4

E-No	E-Name	Add	D_No	E_No
1	Ram	Delhi	D1	1
1	Ram	Delhi	D2	2
1	Ram	Delhi	D3	4
2	Ravi	Hyd	D1	1
2	Ravi	Hyd	D2	2
2	Ravi	Hyd	D3	4
3	Raju	Chennai	D1	1
3	Raju	Chennai	D2	2
3	Raju	Chennai	D3	4
4	Raghu	Bangalore	D1	1
4	Raghu	Bangalore	D2	2
4	Raghu	Bangalore	D3	4

E_Name
Ram
Ravi
Raghu

Self Join

- Find Students enrolled in at least two courses
  - Select T1.S_Id from Study as T1, Study as T2 where T1.S_Id=T2.S_Id AND T1.C_Id!=T2.C_Id

S_Id	C_Id	Since
S1	C1	2019
S2	C2	2020
S1	C2	2020

S_Id	C_Id	Since	S_Id	C_Id	Since
S1	C1	2019	S1	C1	2019
S1	C1	2019	S2	C2	2020
S1	C1	2019	S1	C2	2020
S2	C2	2020	S1	C1	2019
S2	C2	2020	S2	C2	2020
S2	C2	2020	S1	C2	2020
S1	C2	2020	S1	C1	2019
S1	C2	2020	S2	C2	2020
S1	C2	2020	S1	C2	2020

S_Id
S1

Left Outer Join

- Find Employees who are working in Dept
- Select E_No, E_Name, D_Name, Loc from Emp LEFT Outer Join Dept ON(Emp.Dept_No=Dept.Dept.No)

E_No	E-Name	Dept_No
1	Ram	D1
2	Ravi	D2
3	Raju	D3
4	Raghu	--

Dept_no	D_Name	Loc
D1	HR	Delhi
D2	IT	Hyd
D3	Finance	Chennai

E-No	E-Name	D_No	D_No	D_Name	Loc
1	Ram	D1	D1	HR	Delhi
1	Ram	D1	D2	IT	Hyd
1	Ram	D1	D3	Finance	Chennai
2	Ravi	D2	D1	HR	Delhi
2	Ravi	D2	D2	IT	Hyd
2	Ravi	D2	D3	Finance	Chennai
3	Raju	D2	D1	HR	Delhi
3	Raju	D3	D2	IT	Hyd
3	Raju	D3	D3	Finance	Chennai
4	Raghu	--	Null	Null	Null
4	Raghu	--	Null	Null	Null
4	Raghu	--	Null	Null	Null

Right Outer Join

- Find Employees who are working in Dept
- Select E_No, E_Name, D_Name, Loc from Emp RIGHT Outer Join Dept ON(Emp.Dept_No=Dept.Dept.No)

E_No	E-Name	Dept_No
1	Ram	D1
2	Ravi	D2
3	Raju	D3

Dept_no	D_Name	Loc
D1	HR	Delhi
D2	IT	Hyd
D3	Finance	Chennai
D4	Testing	Ban

E-No	E-Name	D_No	D_No	D_Name	Loc
1	Ram	D1	D1	HR	Delhi
1	Ram	D1	D2	IT	Hyd
1	Ram	D1	D3	Finance	Chennai
Null	Null	Null	D4	Testing	Ban
2	Ravi	D2	D1	HR	Delhi
2	Ravi	D2	D2	IT	Hyd
2	Ravi	D2	D3	Finance	Chennai
Null	Null	Null	D4	Testing	Ban
3	Raju	D2	D1	HR	Delhi
3	Raju	D3	D2	IT	Hyd
3	Raju	D3	D3	Finance	Chennai
Null	Null	Null	D4	Testing	Ban

Relational Calculus

- Relational calculus is a non-procedural query language
  - Tuple Relational Calculus (TRC)
  - Domain Relational Calculus (DRC)

Relational Calculus

- Tuple Relational Calculus (TRC)
  - Used for selecting those tuples that satisfy the given condition
    1. *Query to display the last name of those students where age is greater than 30*
    2. *Query to display all the details of students where Last name is 'Singh'*

First_Name	Last_Name	Age
Ajeet	Singh	30
Chaitanya	Singh	31
Rajeev	Bhatia	27
Carl	Pratap	28

```
{ t.Last_Name | Student(t) AND t.age > 30 }
```

Last_Name
Singh

```
{ t | Student(t) AND t.Last_Name = 'Singh' }
```

First_Name	Last_Name	Age
Ajeet	Singh	30
Chaitanya	Singh	31

Relational Calculus

- Domain Relational Calculus (DRC)
  - Records are filtered based on the domains
    - *Find the first name and age of students where student age is greater than 27*

First_Name	Last_Name	Age
Ajeet	Singh	30
Chaitanya	Singh	31
Rajeev	Bhatia	27
Carl	Pratap	28

$$\{< First\_Name, Age > \mid \in Student \wedge Age > 27\}$$

First_Name	Age
Ajeet	30
Chaitanya	31
Carl	28