VIPS
Technical Campus
योग: कर्मसु कौशलम्
IN PURSUIT OF PERFECTION

SCHOOL OF ENGINEERING AND TECHNOLOGY

VIPS

योगः कर्मसु कौशलम्

IN PURSUIT OF PERFECTION

Database Management Systems

What is SQL?

- **Structured Querying Language** – computer language for relational database management and data manipulation

DDL - Data Definition Language:

CREATE Command

DROP Command

ALTER Command

TRUNCATE Command

RENAME Command

What is SQL?

- **Structured Querying Language** – computer language for relational database management and data manipulation

Following are the four main DML commands in SQL:

SELECT Command

INSERT Command

UPDATE Command

DELETE Command

SQL Commands

CREATE DATABASE

Syntax:- CREATE DATABASE DatabaseName;

CREATE DATABASE testDB;

SHOW DATABASE

Syntax:-SHOW DATABASES;

```
SQL> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| AMROOD             |
| TUTORIALSPOINT     |
| mysql              |
| orig               |
| test               |
| testDB             |
+--------------------+
```

SQL Commands

DROP DATABASE

Syntax:- DROP DATABASE DatabaseName;;

DROP DATABASE testDB;

SHOW DATABASE

Syntax:-SHOW DATABASES;

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| AMROOD             |
| TUTORIALSPOINT     |
| mysql              |
| orig               |
| test               |
+--------------------+
```

SQL Commands

USE DATABASE

Syntax:- USE DatabaseName;

SHOW DATABASE

Syntax:-SHOW DATABASES;

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| AMROOD             |
| TUTORIALSPOINT     |
| mysql              |
| orig               |
| test               |
+--------------------+
```

if you want to work with AMROOD database

USE AMROOD;

SQL Commands

CREATE TABLE DATABASE

Syntax:- CREATE TABLE table_name(
column1 datatype,
column2 datatype,
column3 datatype,
.....
columnN datatype,
PRIMARY KEY(one or more columns)
);

```
SQL> CREATE TABLE CUSTOMERS(
 ID INT NOT NULL,
 NAME VARCHAR (20) NOT NULL,
 AGE INT NOT NULL,
 ADDRESS CHAR (25) ,
 SALARY DECIMAL (18, 2),
 PRIMARY KEY (ID)
);
```

SQL Commands


```
SQL> DESC CUSTOMERS;
+---------+--------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+--------------+------+-----+---------+-------+
| ID      | int(11)      | NO   | PRI |         |       |
| NAME    | varchar(20)  | NO   |     |         |       |
| AGE     | int(11)      | NO   |     |         |       |
| ADDRESS | char(25)     | YES  |     | NULL    |       |
| SALARY  | decimal(18,2)| YES  |     | NULL    |       |
+---------+--------------+------+-----+---------+-------+
```

```
SQL> CREATE TABLE CUSTOMERS(
 ID INT NOT NULL,
 NAME VARCHAR (20) NOT NULL,
 AGE INT NOT NULL,
 ADDRESS CHAR (25) ,
 SALARY DECIMAL (18, 2),
 PRIMARY KEY (ID)
);
```

SQL Commands

Following are commonly used constraints available in SQL:

- NOT NULL Constraint: Ensures that a column cannot have NULL value.
- DEFAULT Constraint: Provides a default value for a column when none is specified.
- UNIQUE Constraint: Ensures that all values in a column are different.
- PRIMARY Key: Uniquely identified each rows/records in a database table.
- FOREIGN Key: Uniquely identified a row/record in any other database table.
- CHECK Constraint: The CHECK constraint ensures that all values in a column satisfy certain conditions.
- INDEX: Use to create and retrieve data from the database very quickly

SQL Commands


```
CREATE TABLE CUSTOMERS(
 ID INT NOT NULL,
 NAME VARCHAR (20)          NOT NULL,
 AGE INT                    NOT NULL CHECK (AGE >= 18),
 ADDRESS CHAR (25) ,
 SALARY                DECIMAL (18, 2),  DEFAULT
5000.00,
 PRIMARY KEY (ID)
);
```

Foreign Key Constraints

```
CREATE TABLE ORDERS (
 O_ID INT NOT NULL,
 DATE DATETIME,
 CUSTOMER_ID INT references CUSTOMERS(ID),
AMOUNT double,
 PRIMARY KEY (O_ID)
);
```

SQL Commands

create a table SALARY using CUSTOMERS table and having fields customer ID and customer SALARY:

```
SQL> CREATE TABLE SALARY AS
 SELECT ID, SALARY
 FROM CUSTOMERS;
```

ID	SALARY
1	2000.00
2	1500.00
3	2000.00
4	6500.00
5	8500.00
6	4500.00
7	10000.00

SQL Commands

DROP TABLE

Syntax:DROP TABLE table_name;

DROP TABLE CUSTOMERS;

DESC CUSTOMERS;

CUSTOMERS' doesn't exist

SQL Commands

INSERT INTO TABLE

Syntax: I. INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)]

VALUES (value1, value2, value3,...valueN);;

II. INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);

INSERT INTO CUSTOMERS(ID,NAME,AGE,ADDRESS,SALARY)

VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00);

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
```

SQL Commands

INSERT INTO one table using another table

Syntax: INSERT INTO first_table_name
[(column1, column2, ... columnN)]

SELECT column1, column2, ...columnN

FROM second_table_name

[WHERE condition];

SQL Commands

SELECT Statement

Syntax :SELECT column1, column2, columnN
FROM table_name;

SELECT ID, NAME, SALARY FROM CUSTOMERS;

ID	NAME	SALARY
1	Ramesh	2000.00
2	Khilan	1500.00
3	kaushik	2000.00

SELECT * FROM CUSTOMERS;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

SQL Commands

WHERE clause- For condition

SELECT column1, column2, columnN

FROM table_name

WHERE [condition]

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000;

ID	NAME	SALARY
4	Chaitali	6500.00
5	Hardik	8500.00
6	Komal	4500.00
7	Muffy	10000.00

comparison or logical operators like >, <, =, LIKE, NOT etc.

SQL Commands


```
SQL> SELECT * FROM CUSTOMERS WHERE AGE >= 25 OR SALARY >= 6500;
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
5 rows in set (0.00 sec)

SQL>  SELECT * FROM CUSTOMERS WHERE AGE IS NOT NULL;
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Base table

SQL Commands


```
SQL> SELECT * FROM CUSTOMERS WHERE NAME LIKE 'Ko%';
+----+--------+-----+---------+----------+
| ID | NAME   | AGE | ADDRESS | SALARY   |
+----+--------+-----+---------+----------+
|  6 | Komal  |  22 | MP      | 4500.00  |
+----+--------+-----+---------+----------+
1 row in set (0.00 sec)

SQL> SELECT * FROM CUSTOMERS WHERE AGE IN ( 25, 27 )
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  2 | Khilan   |  25 | Delhi   | 1500.00  |
|  4 | Chaitali |  25 | Mumbai  | 6500.00  |
|  5 | Hardik   |  27 | Bhopal  | 8500.00  |
+----+----------+-----+---------+----------+
3 rows in set (0.00 sec)

SQL> SELECT * FROM CUSTOMERS WHERE AGE BETWEEN 25 AND 27;
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  2 | Khilan   |  25 | Delhi   | 1500.00  |
|  4 | Chaitali |  25 | Mumbai  | 6500.00  |
|  5 | Hardik   |  27 | Bhopal  | 8500.00  |
+----+----------+-----+---------+----------+
```

Base table

SQL Commands


```
SQL> SELECT * FROM CUSTOMERS
WHERE AGE > ALL (SELECT AGE FROM CUSTOMERS WHERE SALARY > 6500);
+----+---------+-----+-----------+----------+
| ID | NAME    | AGE | ADDRESS   | SALARY   |
+----+---------+-----+-----------+----------+
|  1 | Ramesh  |  32 | Ahmedabad | 2000.00  |
+----+---------+-----+-----------+----------+
1 row in set (0.02 sec)

SQL> SELECT * FROM CUSTOMERS
WHERE AGE > ANY (SELECT AGE FROM CUSTOMERS WHERE SALARY > 6500);
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad | 2000.00  |
|  2 | Khilan   |  25 | Delhi     | 1500.00  |
|  4 | Chaitali |  25 | Mumbai    | 6500.00  |
|  5 | Hardik   |  27 | Bhopal    | 8500.00  |
+----+----------+-----+-----------+----------+
4 rows in set (0.00 sec)
```

Base table

SQL Commands

```
SQL> SELECT AGE FROM CUSTOMERS
WHERE EXISTS (SELECT AGE FROM CUSTOMERS WHERE SALARY > 650
+-----+
| AGE |
+-----+
|  32 |
|  25 |
|  23 |
|  25 |
|  27 |
|  22 |
|  24 |
+-----+
7 rows in set (0.02 sec)
```

Base table

Logical Operators

Operator	Description
ALL	The ALL operator is used to compare a value to all values in another value set.
AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
ANY	The ANY operator is used to compare a value to any applicable value in the list according to the condition.
BETWEEN	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
EXISTS	The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.
IN	The IN operator is used to compare a value to a list of literal values that have been specified.
LIKE	The LIKE operator is used to compare a value to similar values using wildcard operators.
NOT	The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**
OR	The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
IS NULL	The NULL operator is used to compare a value with a NULL value.
UNIQUE	The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

SQL Commands

UPDATE Query

UPDATE table_name

SET column1 = value1, column2 = value2...., columnN = valueN

WHERE [condition];

```
SQL> UPDATE CUSTOMERS
SET ADDRESS = 'Pune'
WHERE ID = 6;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	Pune	4500.00
7	Muffy	24	Indore	10000.00

Base table

SQL Commands

UPDATE Query

UPDATE table_name

SET column1 = value1, column2 = value2...., columnN = valueN

WHERE [condition];

SQL> UPDATE CUSTOMERS

SET ADDRESS = 'Pune', SALARY = 1000.00;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Pune	1000.00
2	Khilan	25	Pune	1000.00
3	kaushik	23	Pune	1000.00
4	Chaitali	25	Pune	1000.00
5	Hardik	27	Pune	1000.00
6	Komal	22	Pune	1000.00
7	Muffy	24	Pune	1000.00

Base table

SQL Commands

DELETE Query

DELETE FROM table_name

WHERE [condition];

DELETE query to delete selected rows, otherwise all the records would be deleted.

DELETE FROM CUSTOMERS

WHERE ID = 6;

1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
7	Muffy	24	Indore	10000.00

Base table

SQL Commands

DELETE Query

DELETE FROM table_name

WHERE [condition];

DELETE query to delete selected rows, otherwise all the records would be deleted.

you want to DELETE all the records from CUSTOMERS table, you do not need to use WHERE clause

SQL> DELETE FROM CUSTOMERS;

Now, CUSTOMERS table would not have any record.

Base table

SQL Commands

LIKE Clause

- The percent sign (%)
- The underscore (_)

SQL> SELECT * FROM CUSTOMERS
WHERE SALARY LIKE '200%';

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
+----+----------+-----+-----------+----------+
```

[Base table]

SQL Commands

ORDER BY Clause

SELECT column-list

FROM table_name

[WHERE condition]

[ORDER BY column1, column2, .. columnN] [ASC | DESC];

SQL> SELECT * FROM CUSTOMERS

ORDER BY NAME, SALARY;

// Here ascending order by name and where name is same arrange ascending order by salary.

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
3	kaushik	23	Kota	2000.00
2	Khilan	25	Delhi	1500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00
1	Ramesh	32	Ahmedabad	2000.00

Base table

SQL Commands

ORDER BY Clause

SELECT column-list

FROM table_name

[WHERE condition]

[ORDER BY column1, column2, .. columnN] [ASC | DESC];

SQL> SELECT * FROM CUSTOMERS

ORDER BY NAME DESC;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
7	Muffy	24	Indore	10000.00
6	Komal	22	MP	4500.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
5	Hardik	27	Bhopal	8500.00
4	Chaitali	25	Mumbai	6500.00

Base table

SQL Commands

Group By Clause

SELECT column1, column2

FROM table_name

WHERE [conditions]

GROUP BY column1, column2

CUSTOMERS table has the following records with duplicate names:

SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS

GROUP BY NAME;

NAME	SUM(SALARY)
Hardik	8500.00
kaushik	8500.00
Komal	4500.00
Muffy	10000.00
Ramesh	3500.00

[Base table1]

SQL Commands

Group By Clause

SELECT column1, column2

FROM table_name

WHERE [conditions]

GROUP BY column1, column2

CUSTOMERS table has the following records with duplicate names:

SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS

GROUP BY NAME;

NAME	SUM(SALARY)
Hardik	8500.00
kaushik	8500.00
Komal	4500.00
Muffy	10000.00
Ramesh	3500.00

Base table1

SQL Commands

Distinct Keyword

SELECT DISTINCT column1, column2,.....columnN

FROM table_name

WHERE [condition]

SQL> SELECT DISTINCT SALARY FROM CUSTOMERS

ORDER BY SALARY;

```
+----------+
| SALARY   |
+----------+
|  1500.00 |
|  2000.00 |
|  4500.00 |
|  6500.00 |
|  8500.00 |
| 10000.00 |
+----------+
```

[Base table]

SQL Commands

RENAME Cars To Car_2021_Details ;

RENAME statement

Car Name	Car Color	Car Cost
Hyundai Creta	White	10,85,000
Hyundai Venue	White	9,50,000
Hyundai i20	Red	9,00,000
Kia Sonet	White	10,00,000
Kia Seltos	Black	8,00,000
Swift Dezire	Red	7,95,000

Table: Cars

SQL Commands

ALTER TABLE - ADD Column

ALTER TABLE table_name

ADD column_name datatype;

ALTER TABLE Customers

ADD Email varchar(255);

ALTER TABLE - DROP COLUMN

ALTER TABLE table_name

DROP COLUMN column_name;

ALTER TABLE Customers

DROP COLUMN Email;

ALTER TABLE - RENAME COLUMN

ALTER TABLE table_name

RENAME COLUMN old_name to new_name;

ALTER TABLE - MODIFY DATATYPE

ALTER TABLE table_name

MODIFY COLUMN column_name datatype;

SQL Commands

ALTER TABLE - ADD Constraints

ALTER TABLE table_name

ADD constraint constraintname(column_name) ;

ALTER TABLE - Drop Constraints

ALTER TABLE table_name

drop constraint constraintname;

SQL Commands

ALTER TABLE Command

ALTER TABLE CUSTOMERS ADD S char(1);

ID	NAME	AGE	ADDRESS	SALARY	S
1	Ramesh	32	Ahmedabad	2000.00	NULL
2	Ramesh	25	Delhi	1500.00	NULL
3	kaushik	23	Kota	2000.00	NULL
4	kaushik	25	Mumbai	6500.00	NULL
5	Hardik	27	Bhopal	8500.00	NULL
6	Komal	22	MP	4500.00	NULL
7	Muffy	24	Indore	10000.00	NULL

ALTER TABLE CUSTOMERS DROP column S;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Ramesh	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	kaushik	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Base table

SQL Commands

TRUNCATE TABLE

Syntax: TRUNCATE TABLE table_name;

The DROP command is used to remove table structure and its contents. Whereas the TRUNCATE command is used to delete all the rows/records from the table,it will not remove the table structure.

SQL > TRUNCATE TABLE CUSTOMERS;

SQL> SELECT * FROM CUSTOMERS;

Empty set (0.00 sec)

Base table

SQL Commands

Alias Syntax

ORDERS

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

SQL> SELECT C.ID, C.NAME, C.AGE, O.AMOUNT FROM CUSTOMERS AS C, ORDERS AS O WHERE C.ID = O.CUSTOMER_ID;

ID	NAME	AGE	AMOUNT
3	kaushik	23	3000
3	kaushik	23	1500
2	Khilan	25	1560
4	Chaitali	25	2060

Base table

SQL Commands

Alias Syntax

column alias:

SQL> SELECT ID AS CUSTOMER_ID, NAME AS CUSTOMER_NAME FROM CUSTOMERS WHERE SALARY IS NOT NULL;

CUSTOMER_ID	CUSTOMER_NAME
1	Ramesh
2	Khilan
3	kaushik
4	Chaitali
5	Hardik
6	Komal
7	Muffy

Base table

SQL Commands

HAVING CLAUSE

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

Syntax:

The following is the position of the HAVING clause in a query:

SELECT

FROM

WHERE

GROUP BY

HAVING

ORDER BY

HAVING CLAUSE

Emp_Id	Emp_Name	Emp_Salary	Emp_City
201	Abhay	2000	Goa
202	Ankit	4000	Delhi
203	Bheem	8000	Jaipur
204 Ram	2000	Goa	
205	Sumit	5000	Delhi

If you want to add the salary of employees for each city, you have to write the following query:

- **SELECT SUM(Emp_Salary), Emp_City FROM Employee GROUP BY Emp_City;**

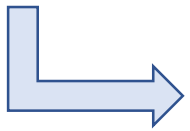
SUM(Emp_Salary)	Emp_City
4000	Goa
9000	Delhi
8000	Jaipur

Q-1 you want to show those cities whose total salary of employees is more than 5000.
SUM(Emp_Salary), Emp_City FROM Employee GROUP BY Emp_City HAVING SUM(Emp_Salary)>5000;

Aggregate Functions

Aggregate functions are used to summarize data, and use descriptive statistics measures.

Function	Description
AVG ( )	Averages a column of values
COUNT ( )	Counts the number of values
MIN ( )	Finds the minimum value in a range
MAX ( )	Find the maximum value in a range
SUM ( )	Sums the column values
DISTNCT	Can be used with some aggregate functions



```
SELECT COUNT (DISTINCT city)
FROM games;
```

SQL JOIN

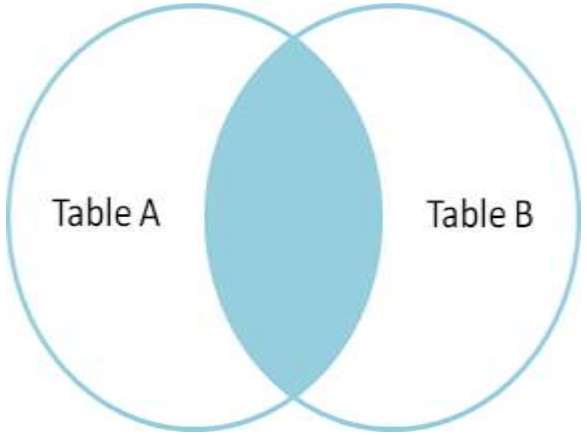
SQL, JOIN means "to combine two or more tables".

The SQL JOIN clause takes records from two or more tables in a database and combines it together.

ANSI standard SQL defines five types of JOIN :

INNER/EQUI JOIN, NATURAL JOIN, CROSS JOIN, SELF JOIN,OUTER JOIN

INNER JOIN/EQUI JOIN



Table_Name : Customers

CustID	Name	Phone_Number
1	Raj Mehta	98540XXXXX
2	Sanjay Mishra	88888XXXXX
3	Aditi Gupta	67809XXXXX
4	Manish Chopra	12345XXXXX

Table_Name : Shopping_Details

ItemID	CustID	Item_Name	Quantity
1	2	Chips	2
2	3	Chocolate	5
3	5	Dress	8

Primary key

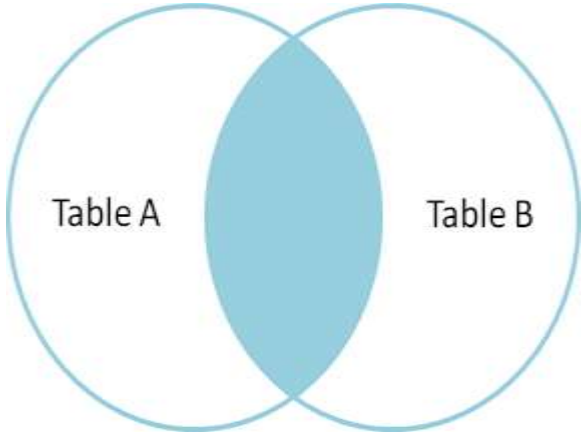
Foreign key

Temporary Table

Customers CustID	Name	Phone_Number	Item_ID	Shopping_details.CustID	Item_Name	Quantity
2	Sanjay Mishra	88888XXXXX	1	2	Chips	2
3	Aditi Gupta	67809XXXXX	2	3	Chocolate	5

The INNER JOIN keyword selects records that have matching values in both tables.

INNER JOIN/EQUI JOIN

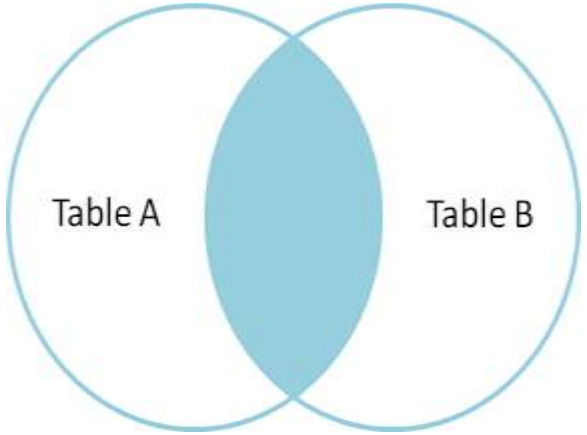



```
SELECT Customers.Name, Shopping_Details.Item_Name,
Shopping_Details.Quantity
FROM Customers INNER JOIN Shopping_Details
ON Customers.CustID==Shopping_Details.CustID;
```

Resultant Table

Name	Item_Name	Quantity
Sanjay Mishra	Chips	2
Aditi Gupta	Chocolate	5

NATURAL JOIN



SQL Natural Join is a type of Inner join based on the condition that columns having the same name and datatype are present in both the tables to be joined.

Table_Name : Customers

CustID	Name	Phone_Number
1	Raj Mehta	98540XXXXX
2	Sanjay Mishra	88888XXXXX
3	Aditi Gupta	67809XXXXX
4	Manish Chopra	12345XXXXX

Table_Name : Shopping_Details

ItemID	CustID	Item_Name	Quantity
1	2	Chips	2
2	3	Chocolate	5
3	5	Dress	8

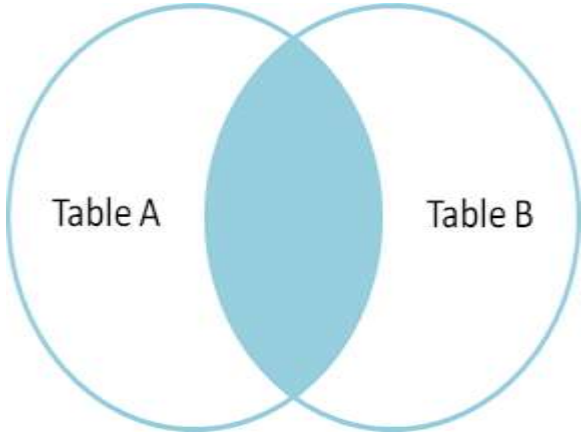
Primary key

Foreign key

Temporary Table

Customers CustID	Name	Phone_Number	Item_ID	Shopping_details.CustID	Item_Name	Quantity
2	Sanjay Mishra	88888XXXXX	1	2	Chips	2
3	Aditi Gupta	67809XXXXX	2	3	Chocolate	5

NATURAL JOIN



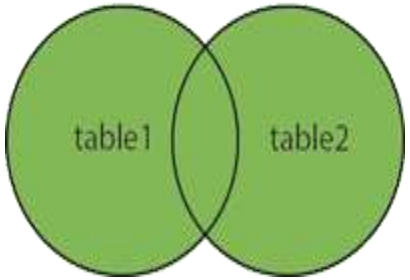

```
SELECT *
FROM Customers NATURAL JOIN Shopping_Details;
```

Resultant Table

CustID	Name	Phone_Number	ItemID	Item_Name	Quantity
2	Sanjay Mishra	88888XXXXX	1	Chips	2
3	Aditi Gupta	67809XXXXX	2	Chocolate	5

OUTER JOI

FULL OUTER JOIN



Table_Name: Customers

CustID	Name	Phone_Number
1	Raj Mehta	98540XXXXX
2	Sanjay Mishra	88888XXXXX
3	Aditi Gupta	67809XXXXX
4	Manish Chopra	12345XXXXX

Table_Name: Shopping_Details

ItemID	CustID	Item_Name	Quantity
1	2	Chips	2
2	3	Chocolate	5
3	5	Dress	8

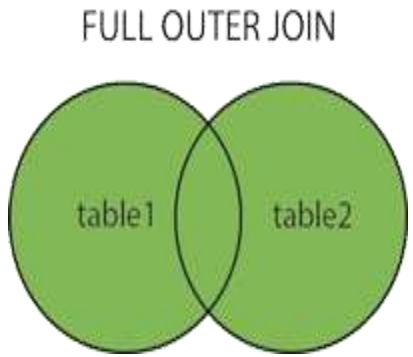
Primary Key

Foreign Key

Temporary table

Customers CustID	Name	Phone_Number	ItemID	Shopping_Details.CustID	Item_Name	Quantity
1	Raj Mehta	98540XXXXX	NULL	NULL	NULL	NULL
2	Sanjay Mishra	88888XXXXX	1	2	Chips	2
3	Aditi Gupta	67809XXXXX	2	3	Chocolate	5
4	Manish Chopra	12345XXXXX	NULL	NULL	NULL	NULL
NULL	NULL	NULL	3	5	Dress	8

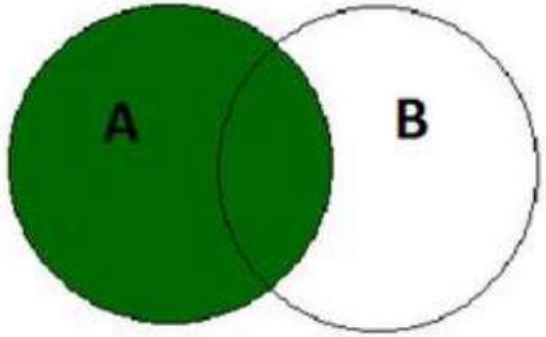
OUTER JOIN




```
SELECT Customers.Name, Shopping_Details.Item_Name
FROM Customers FULL OUTER JOIN Shopping_Details
ON Customer.CustID = Shopping_Details.ID;
```

Name	Item_Details
Sanjay Mishra	Chips
Aditi Gupta	Chocolate
Raj Mehta	NULL
Manish Chopra	NULL
NULL	Dress

LEFT OUTER JOIN



Table_Name : Customers **Left**

CustID	Name	Phone_Number
1	Raj Mehta	98540XXXXX
2	Sanjay Mishra	88888XXXXX
3	Aditi Gupta	67809XXXXX
4	Manish Chopra	12345XXXXX

Table_Name : Shopping_Details **Right**

ItemID	CustID	Item_Name	Quantity
1	2	Chips	2
2	3	Chocolate	5
3	5	Dress	8

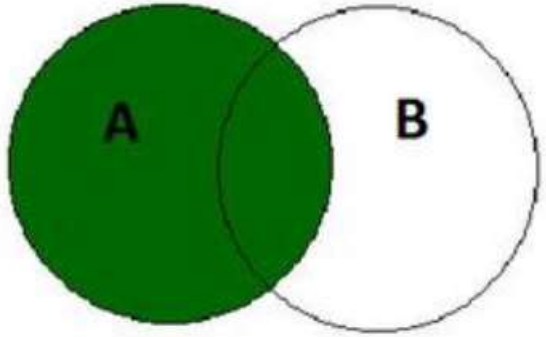
Primary key

Foreign key

Temporary Table

Customers CustID	Name	Phone_Number	Item_ID	Shopping_details.CustID	Item_Name	Quantity
1	Raj Mehta	98540XXXXX	NULL	NULL	NULL	NULL
2	Sanjay Mishra	88888XXXXX	1	2	Chips	2
3	Aditi Gupta	67809XXXXX	2	3	Chocolate	5
4	Manish Chopra	12345XXXXX	NULL	NULL	NULL	NULL

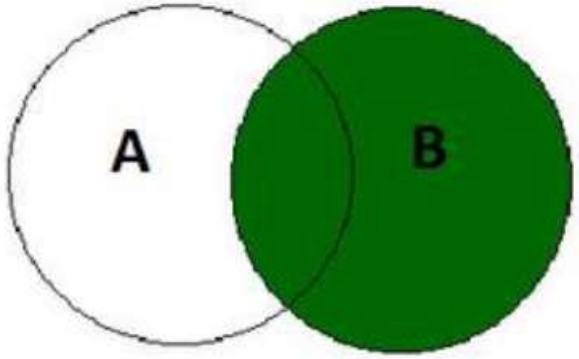
LEFT OUTER JOIN




```
SELECT Customers. Name, Shopping_Details.Item_Name
FROM Customers LEFT OUTER JOIN Shopping_Details
ON Customer.CustID = Shopping_Details.CustID;
```

Name	Item_Details
Sanjay Mishra	Chips
Aditi Gupta	Chocolate
Raj Mehta	NULL
Manish Chopra	NULL

RIGHT OUTER JOIN



Table_Name : Customers **Left**

CustID	Name	Phone_Number
1	Raj Mehta	98540XXXXX
2	Sanjay Mishra	88888XXXXX
3	Aditi Gupta	67809XXXXX
4	Manish Chopra	12345XXXXX

Table_Name : Shopping_Details **Right**

ItemID	CustID	Item_Name	Quantity
1	2	Chips	2
2	3	Chocolate	5
3	5	Dress	8

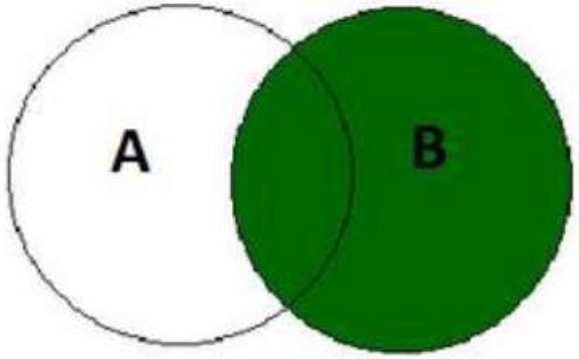
Primary key

Foreign key

Temporary Table

Customers CustID	Name	Phone_Number	Item_ID	Shopping_details.CustID	Item_Name	Quantity
2	Sanjay Mishra	88888XXXXX	1	2	Chips	2
3	Aditi Gupta	67809XXXXX	2	3	Chocolate	5
NULL	NULL	NULL	3	5	Dress	8

RIGHT OUTER JOIN




```
SELECT Customers.Name, Shopping_Details.Item_Name
FROM Customers RIGHT OUTER JOIN Shopping_Details
ON Customer.CustID = Shopping_Details.CustID;
```

Resultant Table

Name	Item_Details
Sanjay Mishra	Chips
Aditi Gupta	Chocolate
NULL	Dress

CROSS JOINS

Cross Join, is the cartesian product of all the rows of the first table with all the rows of the second table.

Table : Customers

ID	Name
1	Raj Mehta
2	Sanjay Mishra
3	Aditi Gupta

Table_Name : Shopping_Details

ID	Item_Name
2	Chips
3	Chocolate

Table : Result

3 x 2 = 6 rows

Customers CustID	Name	Shopping_details.CustID	Item_Name
1	Raj Mehta	2	Chips
1	Raj Mehta	3	Chocolate
2	Sanjay Mishra	2	Chips
2	Sanjay Mishra	3	Chocolate
3	Aditi Gupta	2	Chips
3	Aditi Gupta	3	Chocolate

```
SELECT *
FROM Customers CROSS JOIN
Shopping_Details;
```

SELF JOIN

Self Join, a table is joined to itself.

ID	Name	Phone_Number	Supervisor_ID
1	Raj Mehta	98540XXXXX	4
2	Sanjay Mishra	88888XXXXX	3
3	Aditi Gupta	67809XXXXX	4
4	Manish Chopra	12345XXXXX	7

Primary key

Foreign key

```
SELECT a.Name AS Supervisors
FROM Employees a, Employees b
WHERE a.ID = b.supervisor_ID;
```

Result Table :

Supervisors
Aditi Gupta
Manish Chopra

F

salesman

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5003	Lauson Hen		0.12
5007	Paul Adam	Rome	0.13

customer

customer_id	customer_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3005	Graham Zusi	California	200	5002
3001	Brad Guzan	London		
3004	Fabian Johns	Paris	300	5006
3007	Brad Davis	New York	200	5001
3009	Geoff Camero	Berlin	100	
3008	Julian Green	London	300	5002
3003	Jozy Altidor	Moncow	200	5007

order

order_no	*purch_amt*	*order_date*	*customer_id*	*salesman_id*
70001	*150.5*	*2016-10-05*	*3005*	*5002*
70009	*270.65*	*2016-09-10*	*3001*	
70002	*65.26*	*2016-10-05*	*3002*	*5001*
70004	*110.5*	*2016-08-17*	*3009*	
70007	*948.5*	*2016-09-10*	*3005*	*5002*
70005	*2400.6*	*2016-07-27*	*3007*	*5001*
70008	*5760*	*2016-09-10*	*3002*	*5001*
70010	*1983.43*	*2016-10-10*	*3004*	*5006*
70003	*2480.4*	*2016-10-10*	*3009*	
70012	*250.45*	*2016-06-27*	*3008*	*5002*
70011	*75.29*	*2016-08-17*	*3003*	*5007*

String Functions

- Concatenation

```
SELECT CONCAT(firstName, ' ', lastName) AS
fullName
FROM bookshop_customer;
```

	FIRSTNAME	LASTNAME
1	Ada	Lovelace
2	Grace	Hopper
3	Elizabeth	Feinler
4	Maya	Angelou

	FULL_NAME
1	Ada Lovelace
2	Grace Hopper
3	Elizabeth Feinler
4	Maya Angelou

String Functions

SUBSTRING

It takes three arguments: the string to be removed from, the substring's starting position, and the substring's length.

```
SELECT SUBSTRING(lastName, 1, 3) AS firstName
FROM bookshop_customer;
```

firstName
Lov
Hop
Fei
Ang

	FIRSTNAME	LASTNAME
1	Ada	Lovelace
2	Grace	Hopper
3	Elizabeth	Feinler
4	Maya	Angelou

String Functions

LOWER / UPPER

	CUSTOMER_NAME	EMAIL
1	Ada Lovelace	ada@coginiti.co
2	Grace Hopper	grace@coginiti.co
3	Elizabeth Feinler	elizabeth@company.co
4	Maya Angelou	maya@company.co
5	Annie Easley	annie@coginiti.co
6	Karen Sparck Jones	karen@coginiti.co

```
SELECT UPPER(title) AS uppercase_name
FROM bookshop;
```

Resulting table:

	UPPERCASE_NAME
1	ADA LOVELACE
2	GRACE HOPPER
3	ELIZABETH FEINLER
4	MAYA ANGELOU
5	ANNIE EASLEY
6	KAREN SPARCK JONES

Q- SELECT LOWER(email) AS lowercase_email
FROM bookshop_customer;

String Functions

TRIM

The TRIM SQL string function removes leading or trailing whitespace (spaces, tabs, or newlines) from a string. TRIM function only removes whitespace characters from the beginning and end of the input strin g. If there are whitespace characters in the middle of the string, the function will not affect them. Also, if the input string is null, the TRIM function returns a null value. Additionally, there are variations of TRIM, such as LTRIM (removes only leading spaces) and RTRIM (removes only trailing spaces), that can be used depending on the requirements.

```
SELECT TRIM(customer_name) AS trimmed_name
FROM bookshop_orders;
```

Resulting table:

	UPPERCASE_NAME
1	ADA LOVELACE
2	GRACE HOPPER
3	ELIZABETH FEINLER
4	MAYA ANGELOU
5	ANNIE EASLEY
6	KAREN SPARCK JONES

Q- SELECT LOWER(email) AS lowercase_email
FROM bookshop_customer;

String Functions

TRIM

```
SQL> SELECT TRIM('   bar    ');
+-------------------------------------------------------------+
| TRIM('   bar    ')                                          |
+-------------------------------------------------------------+
| bar                                                         |
+-------------------------------------------------------------+
1 row in set (0.00 sec)

SQL> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
+-------------------------------------------------------------+
| TRIM(LEADING 'x' FROM 'xxxbarxxx')                          |
+-------------------------------------------------------------+
| barxxx                                                      |
+-------------------------------------------------------------+
1 row in set (0.00 sec)

SQL> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
+-------------------------------------------------------------+
| TRIM(BOTH 'x' FROM 'xxxbarxxx')                             |
+-------------------------------------------------------------+
| bar                                                         |
+-------------------------------------------------------------+
1 row in set (0.00 sec)

SQL> SELECT TRIM(TRAILING 'xyz' FROM 'barxxyz');
+-------------------------------------------------------------+
| TRIM(TRAILING 'xyz' FROM 'barxxyz')                         |
+-------------------------------------------------------------+
| barx                                                        |
```

String Functions

LTRIM(str)

Returns the string str with leading space characters removed.

```
SQL> SELECT LTRIM('  barbar');
+-------------------------------------------------------+
| LTRIM('  barbar')                                     |
+-------------------------------------------------------+
| barbar                                                |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

RTRIM(str)

Returns the string str with trailing space characters removed.

```
SQL> SELECT RTRIM('barbar   ');
+-------------------------------------------------------+
| RTRIM('barbar   ')                                    |
+-------------------------------------------------------+
| barbar                                                |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

String Functions

REPLACE

```
SELECT REPLACE(description, 'unnew',
'second_hand') AS new_description
FROM books;
```

	PRODUCT_ID	NAME	PRICE	DESCRIPTIONS
1	101	Jane Eyre	30	unnew
2	102	The Great Gatsby	25	unnew
3	103	1984	10	unnew

	NEW_DESCRIPTIONS
1	second_hand
2	second_hand
3	second_hand

String Functions

INSTR(str,substr)

Returns the position of the first occurrence substring substr in string str.

```
SQL> SELECT INSTR('foobarbar', 'bar');
+--------------------------------------------------+
| INSTR('foobarbar', 'bar')                        |
+--------------------------------------------------+
| 4                                                |
+--------------------------------------------------+
1 row in set (0.00 sec)
```

String Functions

- Trimming spaces

```
SELECT CONCAT(name,', ', SUBSTR(continent,1,3))
   FROM world;
```

CONCAT(name,'..
Algeria, Afr
Angola, Afr
Benin, Afr
Botswana, Afr
Burkina Faso, Afr
Burundi, Afr

SET Operations in SQL

The SQL Set operation is used to combine the two or more SQL SELECT statements.

Types of Set Operation

Union

UnionAll

Intersect

Minus/EXCEPT

SET Operations in SQL

Union

- The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
- In the union operation, all the number of datatype and columns must be same in both the tables
- The union operation eliminates the duplicate rows from its resultset.

Syntax

```
SELECT column_name FROM table1
UNION
SELECT column_name FROM table2;
```

SET Operations in SQL

Union

```
SELECT * FROM First
UNION
SELECT * FROM Second;
```

The resultset table will look like:

ID	NAME
1	Jack
2	Harry
3	Jackson
4	Stephan
5	David

The First table

ID	NAME
1	Jack
2	Harry
3	Jackson

The Second table

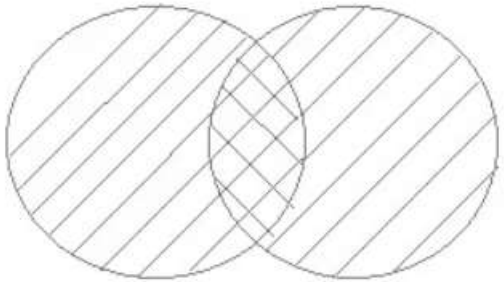
ID	NAME
3	Jackson
4	Stephan
5	David

SET Operations in SQL

Union All

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

```
SELECT column_name FROM table1
UNION ALL
SELECT column_name FROM table2;
```



SET Operations in SQL

Union All

```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```

The resultset table will look like:

ID	NAME
1	Jack
2	Harry
3	Jackson
3	Jackson
4	Stephan
5	David

The First table

ID	NAME
1	Jack
2	Harry
3	Jackson

The Second table

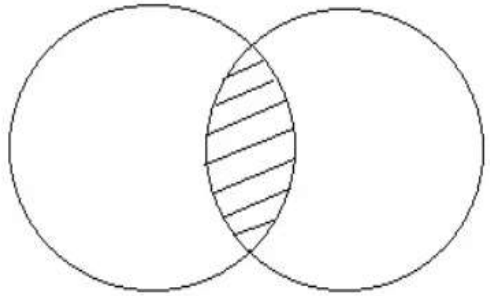
ID	NAME
3	Jackson
4	Stephan
5	David

SET Operations in SQL

Intersect

- It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
- In the Intersect operation, the number of datatype and columns must be the same.
- It has no duplicates

SELECT column_name FROM table1
INTERSECT
SELECT column_name FROM table2;



SET Operations in SQL

Intersect

```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

ID	NAME
3	Jackson

The First table

ID	NAME
1	Jack
2	Harry
3	Jackson

The Second table

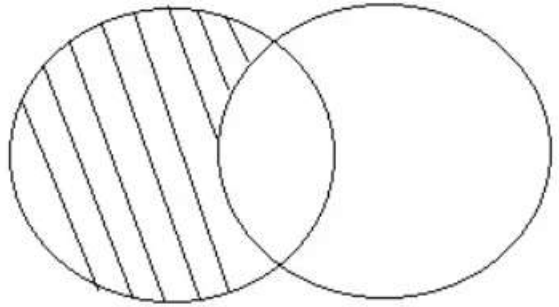
ID	NAME
3	Jackson
4	Stephan
5	David

SET Operations in SQL

Minus/Except

- It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates

SELECT column_name FROM table1
Except
SELECT column_name FROM table2;



SET Operations in SQL

Minus/Except

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```

ID	NAME
1	Jack
2	Harry

The First table

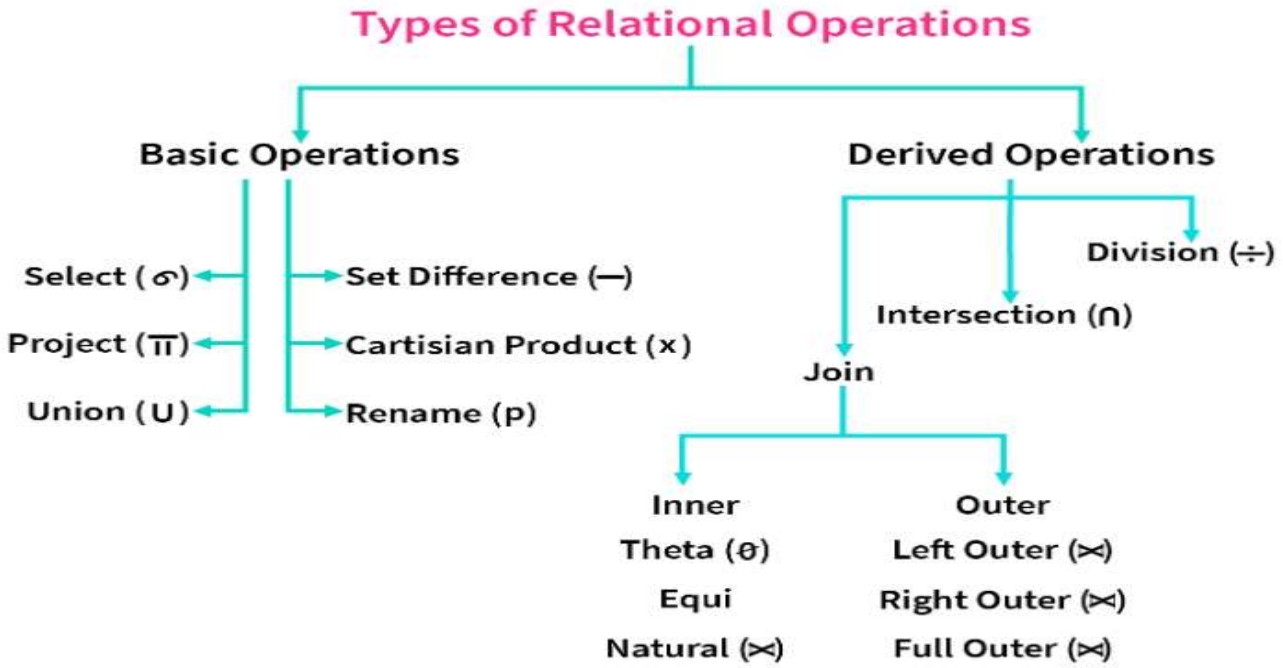
ID	NAME
1	Jack
2	Harry
3	Jackson

The Second table

ID	NAME
3	Jackson
4	Stephan
5	David

Relational Algebra

Relational algebra is a procedural query language. It gives a step by step process to obtain the result of the query. It uses operators to perform queries



Relational Algebra

1. Select Operation:

- ☆ Select tuples that satisfy a given predicate.
- ☆ It is denoted by lowercase Greek letter sigma (σ).
- ☆ Syntax: $\sigma_{\text{<selection\_condition>}}$ (Relation).
- ☆ Example: $\sigma_{D\_ID = 2}$ (Employee).
- ☆ Comparison operators: =, ≠, <, ≤, >, and ≥.
- ☆ Connectives: AND (∧), OR (∨) and NOT (¬).

Relational Algebra

1. Select (σ)

Example 1: Write an RA expression to find all the instructors working in "Finance" department.

Solution:

$\sigma_{Dept\_Name = "Finance"}(INSTRUCTOR)$

Output:

ID	Name	Dept_Name	Salary
26589	Yusuf	Finance	95000
12547	Neil	Finance	80000

INSTRUCTOR

ID	Name	Dept_Name	Salary
10101	John	Biology	65000
12121	Robin	Computer Science	90000
25252	Alya	Electrical	40000
26589	Yusuf	Finance	95000
54789	Ravi	Music	60000
78787	Raj	Physics	87000
87458	Jayant	History	75000
76985	Pratik	Computer Science	89000
12547	Neil	Finance	80000

Example 3: Find all instructors who are working in "Finance" department and drawing the salary greater than $87,000.

Example 2: Find all instructors with salary greater than $87,000.

Relational Algebra

2. Project Operation:

- ☆ In the result, the duplicate rows are eliminated.
- ☆ Syntax: $\Pi_{Attribute1, Attribute2, \ldots}$ (Relation).

Example 1: List all instructors' ID, name, and salary, but do not care about the dept_name.

Solution:

$\Pi_{ID, Name, Salary}$ (INSTRUCTOR)

Example 2: Find the name of all instructors in the Computer Science department.

INSTRUCTOR

ID	Name	Dept_Name	Salary
10101	John	Biology	65000
12121	Robin	Computer Science	90000
25252	Alya	Electrical	40000
26589	Yusuf	Finance	95000
54789	Ravi	Music	60000
78787	Raj	Physics	87000
87458	Jayant	History	75000
76985	Pratik	Computer Science	89000
12547	Neil	Finance	80000

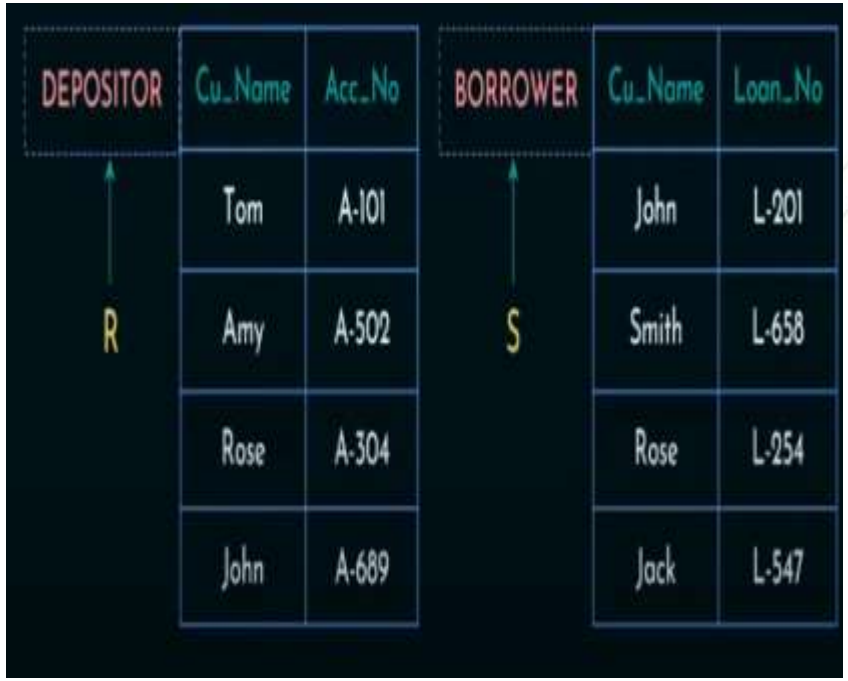
Relational Algebra

3. Union Operation:

- ☆ Like project, the duplicate rows are eliminated.
- ☆ It is denoted by U.
- ☆ Syntax: Π_{Column} (Relation_1) U Π_{Column} (Relation_2)

Example 1: List all customer names associated with the bank either as an account holder or a loan borrower.

Solution: Π_{Cu_Name} (Depositor) U Π_{Cu_Name} (Borrower)



DEPOSITOR (R)

Cu_Name	Acc_No
Tom	A-101
Amy	A-502
Rose	A-304
John	A-689

BORROWER (S)

Cu_Name	Loan_No
John	L-201
Smith	L-658
Rose	L-254
Jack	L-547

Relational Algebra

3. Union Operation:

Instructor (ID, Name, Dept_Name, Salary)
Course (Course_ID, Title, Dept_Name, Credits)
Department (Dept_Name, Building, Budget)
Section (Course_ID, Sec_ID, Semester, Year, Building, Room_No, Time_slot_ID)
Teaches (ID, Course_ID, Sec_ID, Semester, Year)
Student (ID, Name, Dept_Name, Tot_Cred)
Advisor (S_ID, I_ID)
Takes (ID, Course_ID, Sec_ID, Semester, Year, Grade)
Classroom (Building, Room_Number, Capacity)
Time_Slot (Time_Slot_ID, Day, Start_Time, End_Time)

Example 2: Find the set of all courses taught in the Fall 2009 semester, the Spring 2010 semester, or both.

Relational Algebra

4. Set Difference:

Example 1: List all customer names those who have a deposit account but not availed loan.

DEPOSITOR (R)

Cu_Name	Acc_No
Tom	A-101
Amy	A-502
Rose	A-304
John	A-689

BORROWER (S)

Cu_Name	Loan_No
John	L-201
Smith	L-658
Rose	L-254
Jack	L-547

Solution: $\Pi_{Cu\_Name}(Depositor) - \Pi_{Cu\_Name}(Borrower)$

Relational Algebra

5. Cartesian product

Notation: E X D

EMPLOYEE

EMP_ID	EMP_NAME	EMP_DEPT
1	Smith	A
2	Harry	C
3	John	B

DEPARTMENT

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales
C	Legal

EMPLOYEE X DEPARTMENT

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal
2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal

Relational Algebra

5. Cartesian product

INSTRUCTOR

ID	Name	Dept_Name	Salary
10101	John	Biology	65000
78787	Raj	Physics	87000

TEACHES

ID	Course_ID	Sec_ID	Semester	Year
10101	BIO-108	1	Summer	2009
20202	CS-103	1	Spring	2010
78787	PHY-101	2	Fall	2011
12345	EE-203	1	Spring	2009

Example : Find the names of all instructors in the Physics department together with the course id of all courses they taught.

Relational Algebra

5. Cartesian product

$$\sigma_{Instructor.ID = Teaches.ID}\ (\sigma_{Dept\_Name = "Physics"}\ (INSTRUCTOR \times TEACHES))$$

INSTRUCTOR.ID	Name	Dept_Name	Salary	TEACHES.ID	Course_ID	Sec_ID	Semester	Year
78787	Raj	Physics	87000	78787	PHY-101	2	Fall	2011

$$\Pi_{name,\ course\_id}\ (\sigma_{Instructor.ID = Teaches.ID}\ (\sigma_{Dept\_Name = "Physics"}\ (INSTRUCTOR \times TEACHES)))$$

Name	Course_ID
Raj	PHY-101

Relational Algebra

6. Rename Operation:

The rename operation is used to rename the output relation. It is denoted by rho (ρ).

Example: We can use the rename operator to rename STUDENT relation to STUDENT1.

ρ(STUDENT1, STUDENT)

Relational Algebra

7. Set Intersection:

DEPOSITOR	Cu_Name	Acc_No
	Tom	A-101
	Amy	A-502
	Rose	A-304
	John	A-689

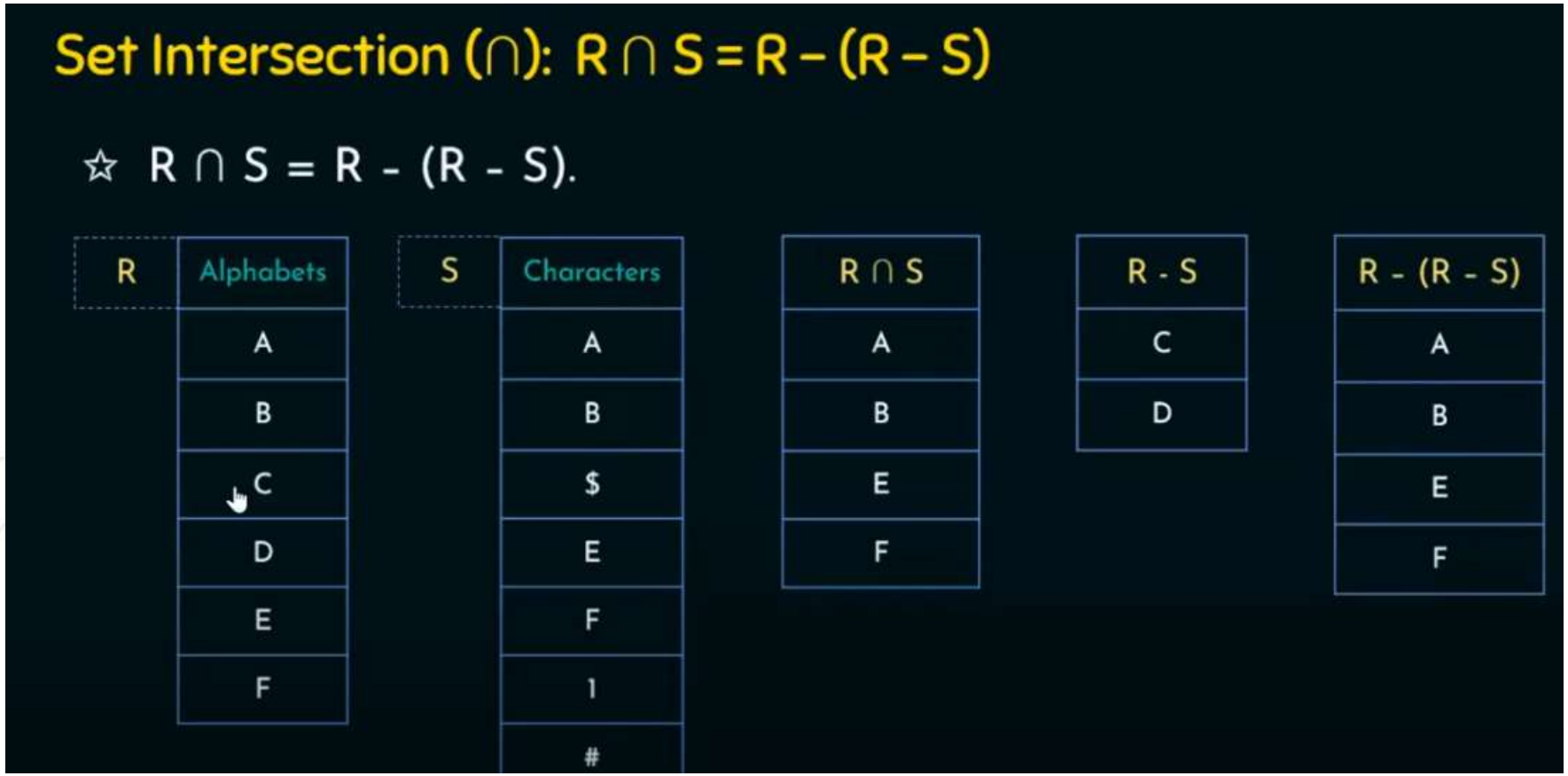
BORROWER	Cu_Name	Loan_No
	John	L-201
	Smith	L-658
	Rose	L-254
	Jack	L-547

Solution: $\Pi_{Cu\_Name}(Depositor) \cap \Pi_{Cu\_Name}(Borrower)$

Example 1: Find the names of all customers who have deposited money and also availed loan.

Relational Algebra

7. Set Intersection:



Relational Algebra

Join:

Join operation denoted by ⋈.

JOIN operation also allows joining variously related tuples from different relations.

Types of JOIN:

Various forms of join operation are:

Inner Joins:

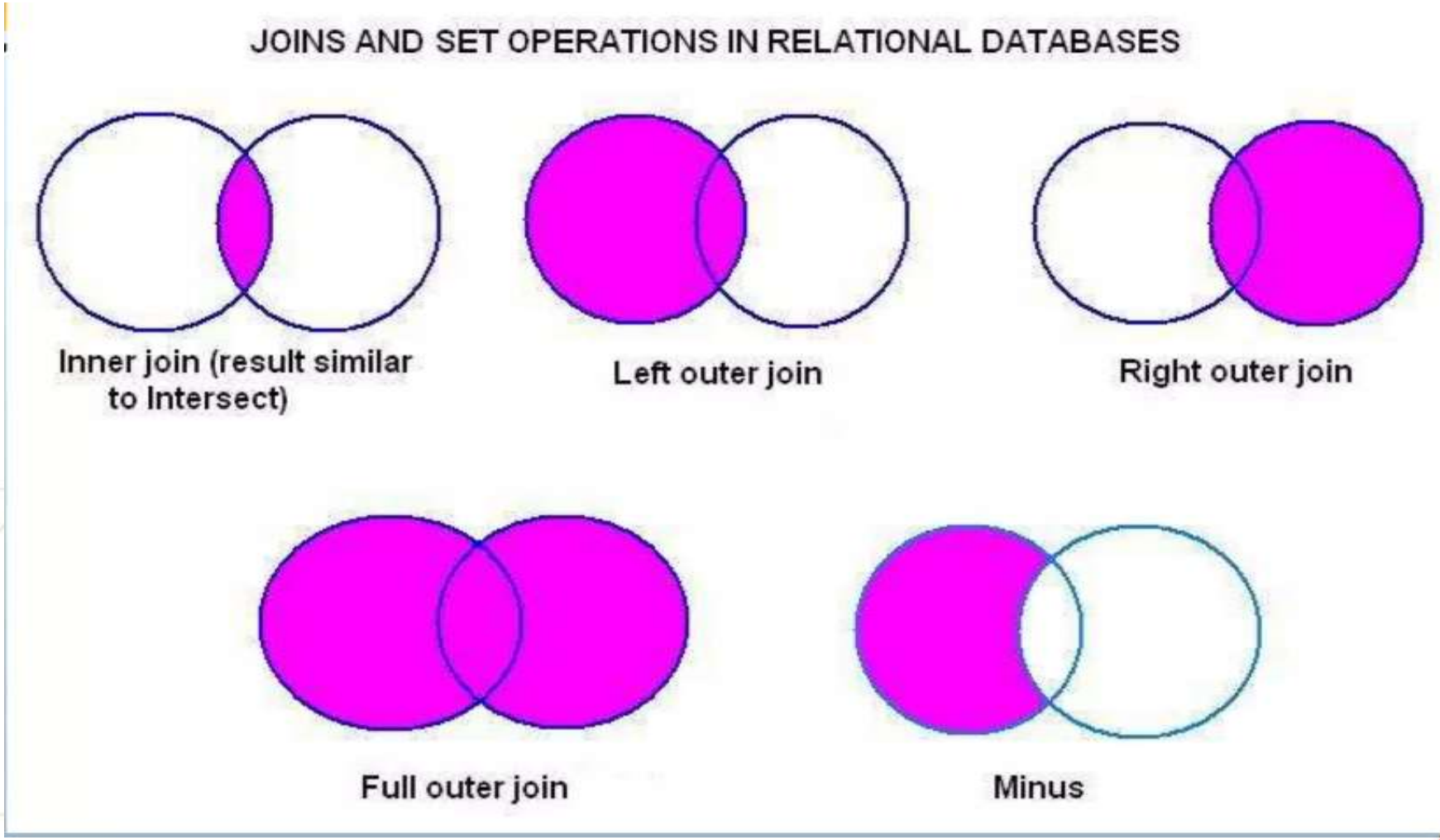
- Theta join
- EQUI join
- Natural join

Outer join:

- Left Outer Join
- Right Outer Join
- Full Outer Join

Relational Algebra

Join:



Relational Algebra

Inner Join

In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded.

Example

1. Theta Join

$A \bowtie \theta B$

Theta join can use any conditions in the selection criteria.

For example:

$A \bowtie$ A.column 2 > B.column 2 (B)

Relational Algebra

Inner Join

Table A		Table B	
column 1	column 2	column 1	column 2
1	1	1	1
1	2	1	3

For example:

$$A \bowtie_{A.column\ 2 > B.column\ 2} (B)$$

A ⋈ A.column 2 > B.column 2 (B)	
column 1	column 2
1	2

Relational Algebra

2. Equi Join:

Syntax: $R \bowtie_{\text{<join condition>}} S$

DEPARTMENT	DName	DNo	Mgr_SSN
	Research	2	553621425
	Finance	5	996856974

EMPLOYEE	SSN	FName	LName	DNo
	123658974	Alex	Smith	2
	553621425	Fred	Scott	5
	996856974	Elsa	David	5
	859689742	Peter	Williams	2

$$Dept\_Mgr \leftarrow DEPARTMENT \bowtie_{Mgr\_SSN = SSN} EMPLOYEE$$

$$Temp \leftarrow DEPARTMENT \times EMPLOYEE$$
$$Dept\_Mgr \leftarrow \sigma_{(MgrSSN = SSN)}(Temp)$$

Relational Algebra

9. Natural Join(*):

PROJECT	PID	PName	DNum
	101	ProjectX	1
	102	ProjectY	2
	103	ProjectZ	2

DEPARTMENT	DNo	Mgr_SSN
	1	553621425
	2	996856974

$$Proj\_Dept \leftarrow PROJECT * \rho_{(DNum,\ Mgr\_SSN)}(DEPARTMENT)$$

$$DEPT \leftarrow \rho_{(DNum,\ Mgr\_SSN)}(DEPARTMENT)$$

$$Proj\_Dept \leftarrow PROJECT * DEPT$$

Relational Algebra

3. Natural Join(*):

PROJECT

PID	PName	DNum
101	ProjectX	1
102	ProjectY	2
103	ProjectZ	2

DEPARTMENT

DNo	Mgr_SSN
1	553621425
2	996856974

$$Proj\_Dept \leftarrow PROJECT * \rho_{(DNum,\ Mgr\_SSN)}(DEPARTMENT)$$

$$DEPT \leftarrow \rho_{(DNum,\ Mgr\_SSN)}(DEPARTMENT)$$

$$Proj\_Dept \leftarrow PROJECT * DEPT$$

Relational Algebra

Outer Join

An Outer Join doesn't require each record in the two join tables to have a matching record. In this type of join, the table retains each record even if no other matching record exists.

Three types of Outer Joins are:

1. Left Outer Join
2. Right Outer Join
3. Full Outer Join

Left Outer Join (A ⟕ B)

Left Outer Join returns all the rows from the table on the left even if no matching rows have been found in the table on the right. When no matching record is found in the table on the right, NULL is returned.

Relational Algebra

1. Left Outer Join

$A \bowtie B$

A ⋈ B

Num	Square	Cube
2	4	8
3	9	18
4	16	–

A

Num	Square
2	4
3	9
4	16

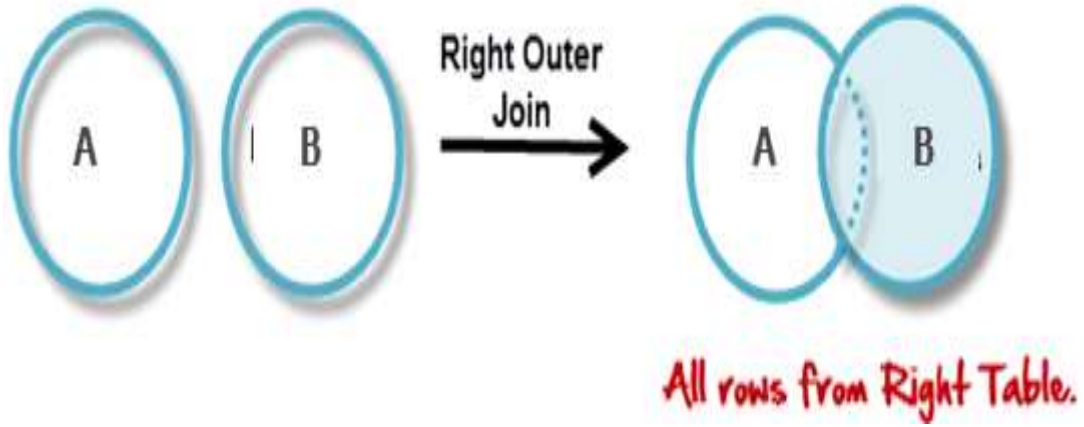
B

Num	Cube
2	8
3	18
5	75

Relational Algebra

2. Right Outer Join (A ⟖ B)

Right Outer Join returns all the columns from the table on the right even if no matching rows have been found in the table on the left. Where no matches have been found in the table on the left, NULL is returned. RIGHT outer JOIN is the opposite of LEFT JOIN



Relational Algebra

2. Right Outer Join (A ⟖ B)

A ⟖ B

A ⋈ B		
Num	Cube	Square
2	8	4
3	18	9
5	75	-

A	
Num	Square
2	4
3	9
4	16

B	
Num	Cube
2	8
3	18
5	75

Relational Algebra

Full Outer Join (A ⟗ B)

Example:

```
A ⟗ B
```

A ⋈ B

Num	Square	Cube
2	4	8
3	9	18
4	16	-
5	-	75

A

Num	Square
2	4
3	9
4	16

B

Num	Cube
2	8
3	18
5	75

Relational Algebra

Summary

- There are mainly two types of joins in DBMS 1) Inner Join 2) Outer Join
- An inner join is the widely used join operation and can be considered as a default join-type.
- Inner Join is further divided into three subtypes: 1) Theta join 2) Natural join 3) EQUI join
- Theta Join allows you to merge two tables based on the condition represented by theta
- When a theta join uses only equivalence condition, it becomes an equi join.
- Natural join does not utilize any of the comparison operators.
- An outer join doesn’t require each record in the two join tables to have a matching record.
- Outer Join is further divided into three subtypes are: 1)Left Outer Join 2) Right Outer Join 3) Full Outer Join
- The LEFT Outer Join returns all the rows from the table on the left, even if no matching rows have been found in the table on the right.
- The RIGHT Outer Join returns all the columns from the table on the right, even if no matching rows have been found in the table on the left.
- In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.

Relational Algebra

Division operation

The division operator is used for queries which involve the 'all'.

R1 ÷ R2 = tuples of R1 associated with all tuples of R2.

Example
Retrieve the name of the subject that is taught in all courses.

Example

Retrieve the name of the subject that is taught in all courses.

Name	Course
System	Btech
Database	Mtech
Database	Btech
Algebra	Btech

÷

Course
Btech
Mtech

=

Name
database

Resources

- https://sqlzoo.net
- https://www.w3schools.com/