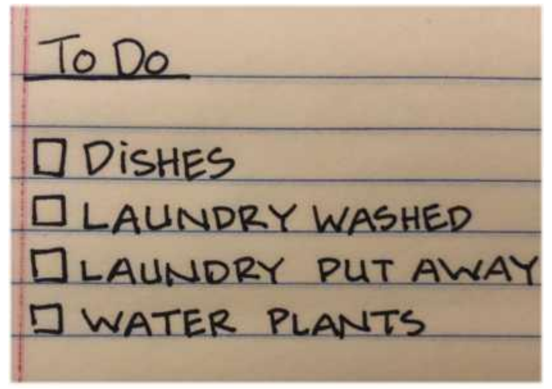
What is DataBase

DataBase is a collection of related information

DataBase can be stored in different ways



To Do

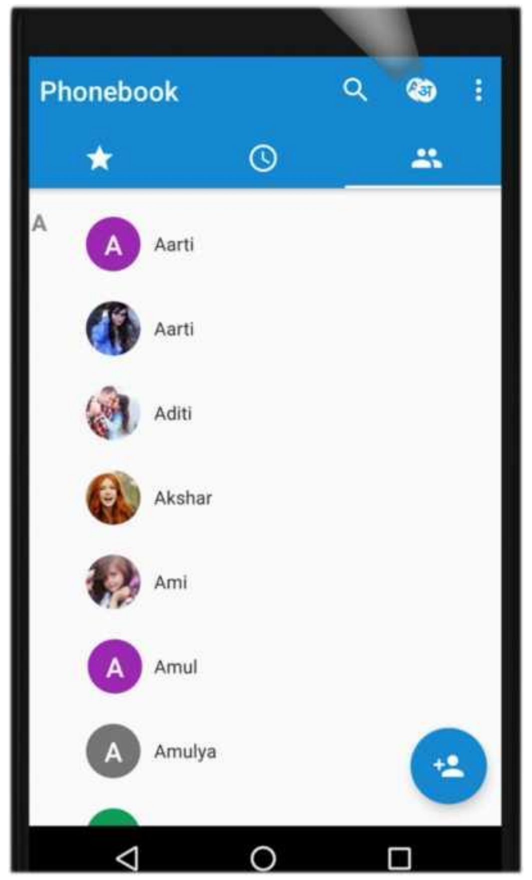
☐ DISHES

☐ LAUNDRY WASHED

☐ LAUNDRY PUT AWAY

☐ WATER PLANTS

Todo List



Phonebook

A

A Aarti

Aarti

Aditi

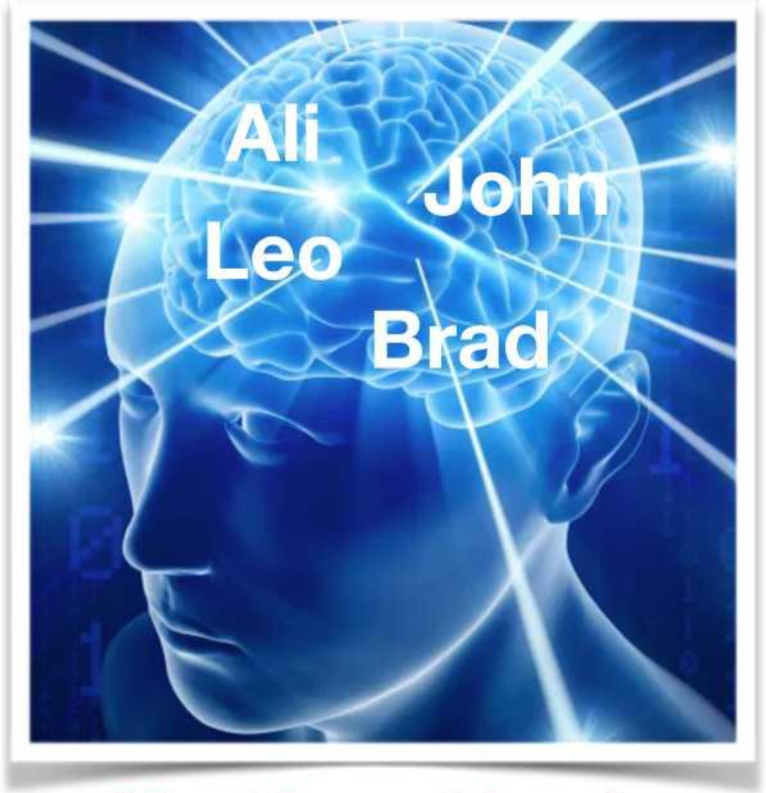
Akshar

Ami

A Amul

A Amulya

Phone book



My 4 best friends

Names of Facebook users

Names of Students in a School

Advantages of Storing Data in Computer’s Memory or Cloud

1) Huge amount of data can be stored

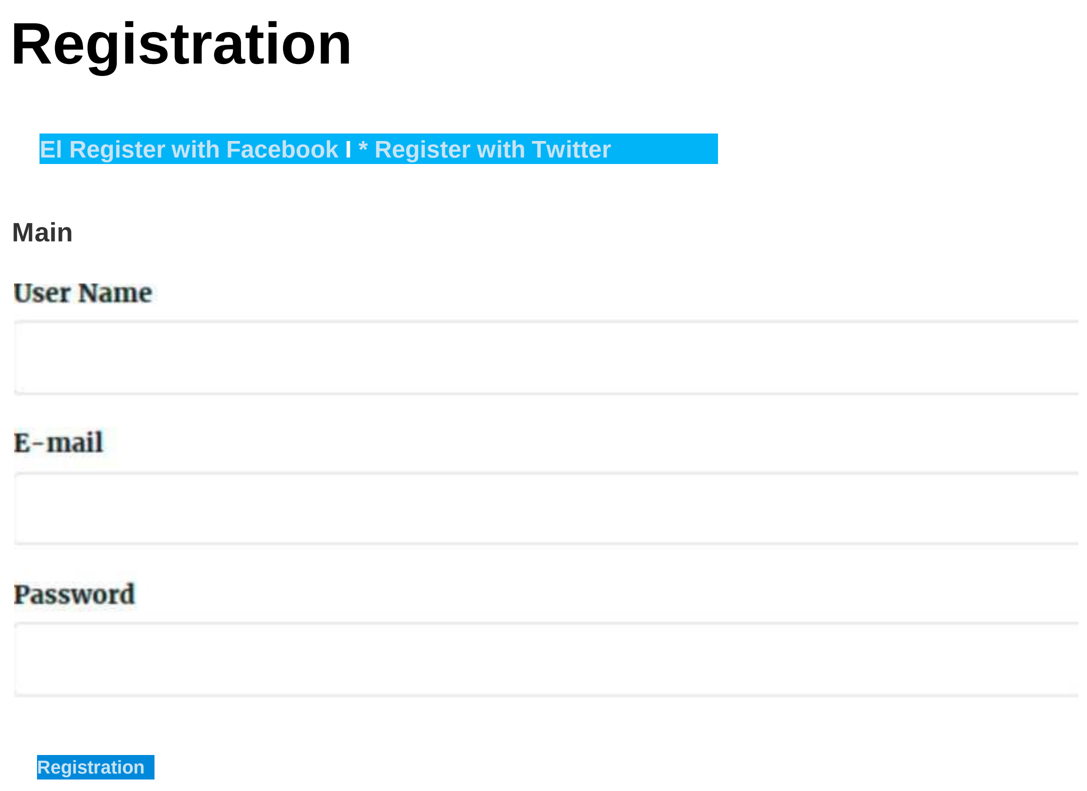
2) Easy to Create, Read, Update, Delete *\r*

3) Easy to access *yf*

4) Quick access *yr*

5) Security

Database Validation Test



Registration

EI Register with Facebook I * Register with Twitter

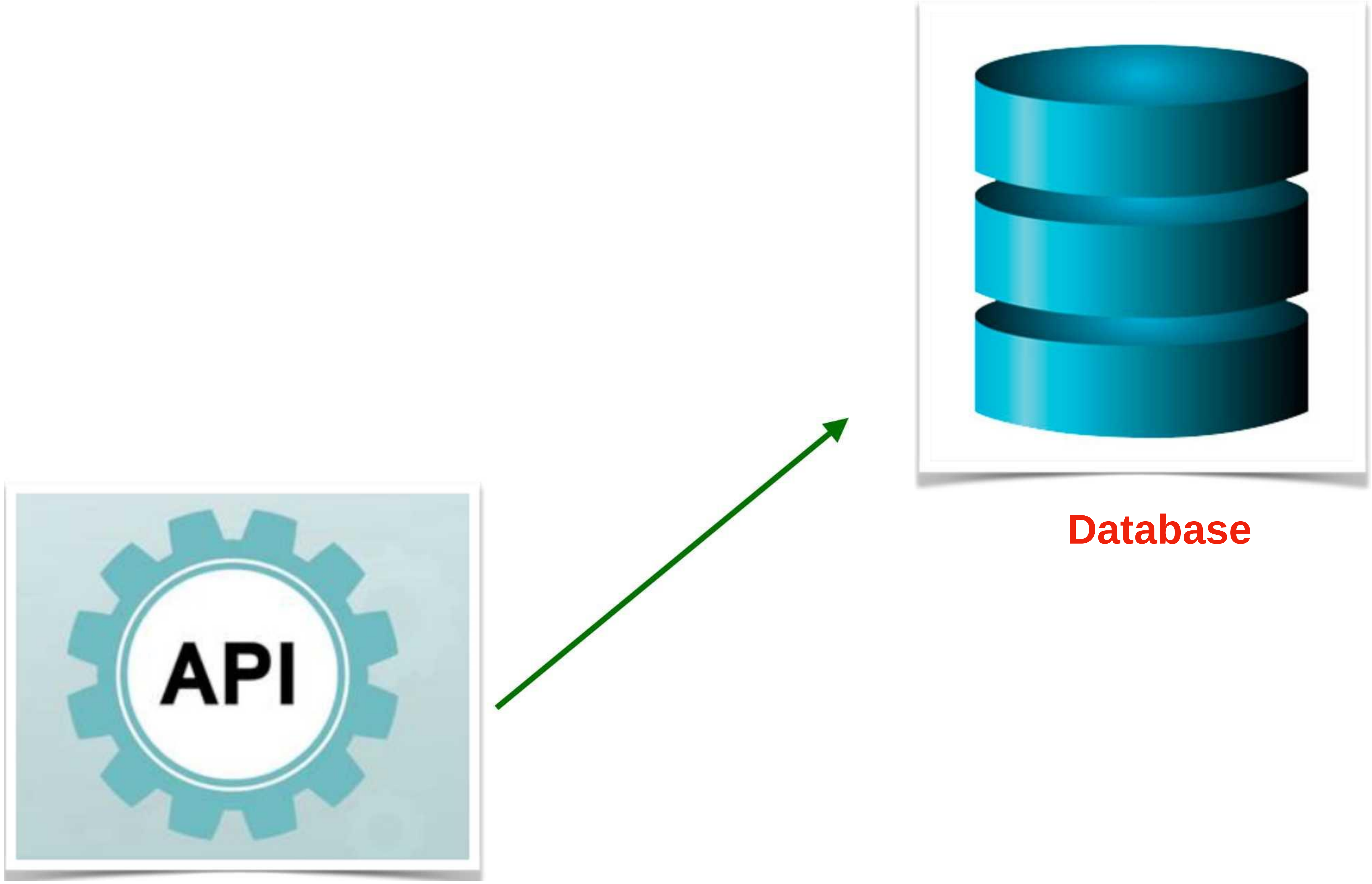
Main

User Name

E-mail

Password

User Interface



Database

API

END To END (E2E) Testing

1) If you send data ta database by using UI
    - A) Validate data from UI by using search functionality (Selenium)
    - B) Validate data by using SQL Codes (SQL + Selenium)
    - C) Validate data by using API Codes (API + Selenium)

2) If you send data to database by using SQL codes
    - A) Validate data from UI by using search functionality (Selenium)
    - B) Validate data by using SQL Codes (SQL + Selenium)
    - C) Validate data by using API Codes (API + Selenium)

3) If you send data to database by using API codes
    - A) Validate data from UI by using search functionality (Selenium)
    - B) Validate data by using SQL Codes (SQL + Selenium)
    - C) Validate data by using API Codes (API + Selenium)

Data Base Management System (DBMS)

DBMS is a special software program which enables its users

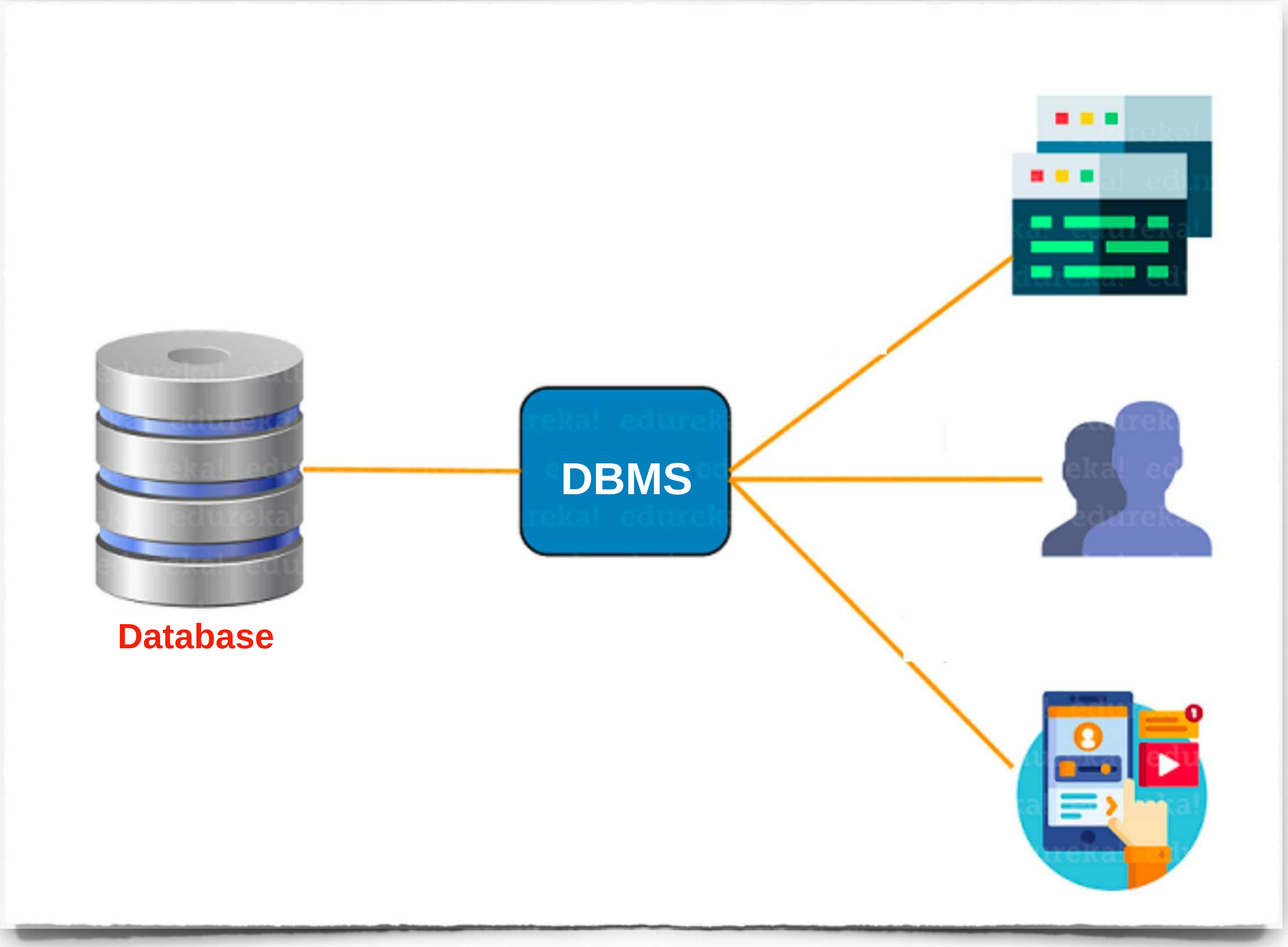
1) To access database,

2) To Create, Read, Update, Delete, (CRUD)

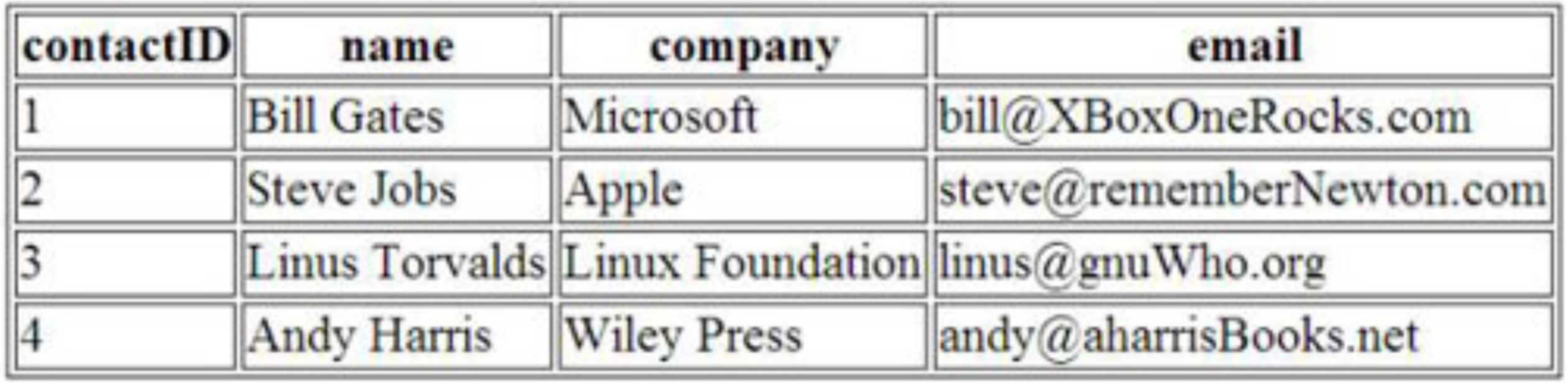
3) To get reports form database,

4) To control access to the database, (Security)

5) To interact with other applications



Tables in SQL



contactID	name	company	email
1	Bill Gates	Microsoft	bill@XBoxOneRocks.com
2	Steve Jobs	Apple	steve@rememberNewton.com
3	Linus Torvalds	Linux Foundation	linus@gnuWho.org
4	Andy Harris	Wiley Press	andy@aharrisBooks.net

Row *(Record)* ====>

Row *(Record)* ====>

Row *(Record)* ====>

Row *(Record)* ====>

Column *(Field)* ====>

Column *(Field)* ====>

Relational Databases *(SQL Databases)*

1) A relational database stores data in tables.

2) The relationship between each data point is clear and searching through those relationships is easy.

3) The relationship between tables and field types is called a schema.

4) Relational Databases are also called SQL Databases. (Structured Query Language)

Fido	Dry	
Rex	Wet	N
Bubbles	Dry	
Cujo	Wet	N

1573	15	21
2684	9	7
3795	27	130
4806	6	5

TAG *tt*	NARIA	**Breed**	**Color**	AQQ
1573	Fido	Beagle	Brown/White	1.5
2684	Rex	Pekingese	White	9
3795	Bubbles	Rottweiler	Black	5
4806	Cujo	Chihuahua	Gold	4

Schema

Popular Relational Databases(SQL Database)

SQL Server : Developed by Microsoft

Cons: It can be expensive - with the Enterprise level costing thousands of dollars.
Pros: It has rich user interface and can handle large quantities of data.

MySQL Server : Created by a Swedish Company

Cons: Tends to stop working when it's given too many operations at a given time.
Pros: It's free and open-source. There's also a lot of documentation and online support.

PostgreSQL Server : Created by a computer science professor Michael Stonebraker.

Cons: Installation and configuration can be difficult.

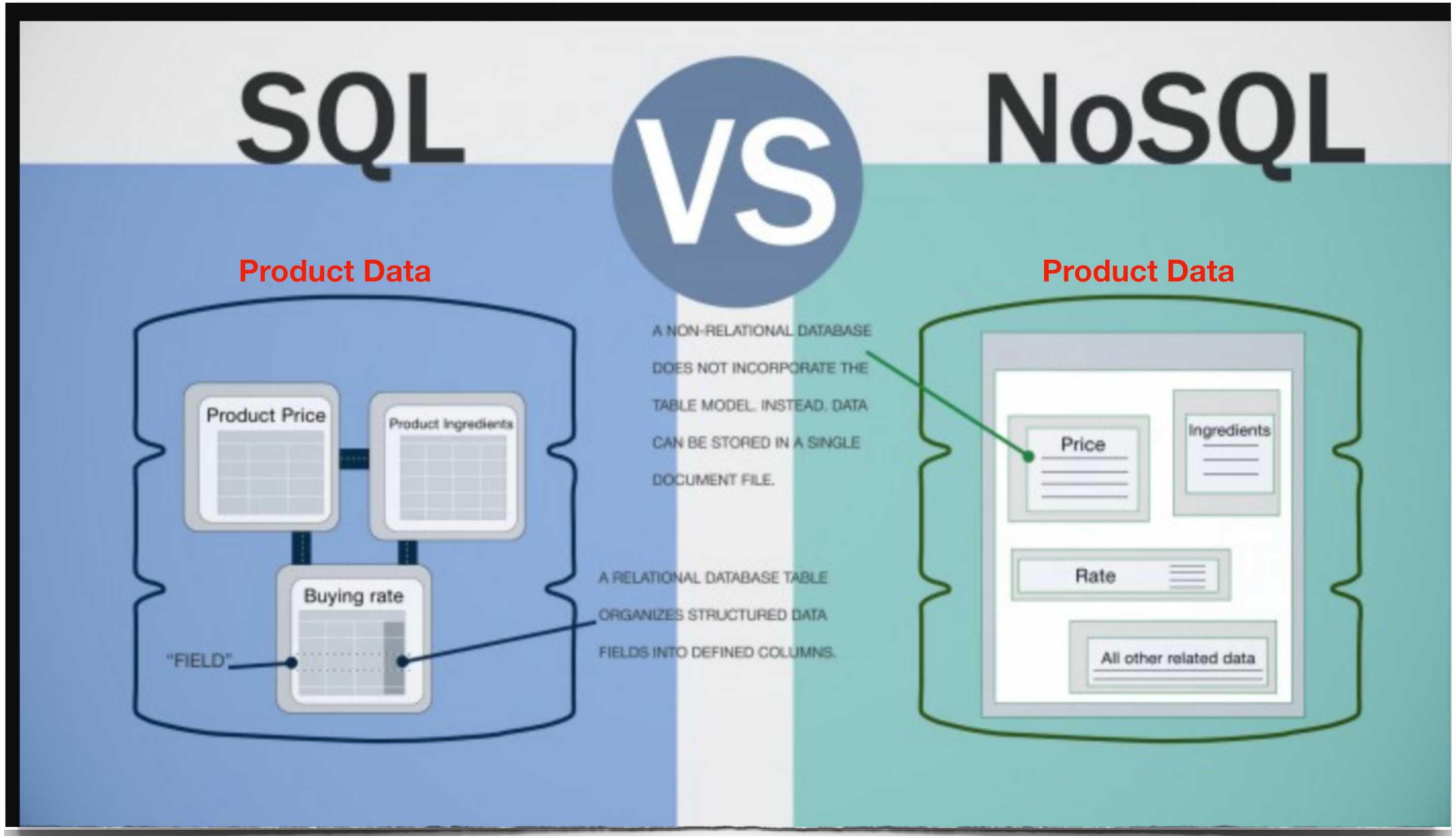
Pros: If you need additional features in PostgreSQL, you can add it yourself - a difficult task in most databases.

PL/SQL is a procedural language designed specifically to embrace SQL statements within its syntax.
PL/SQL program units are compiled by the Oracle Database server and stored inside the database.

Pros: PL/SQL provides high security level.
PL/SQL provides support for Object-Oriented Programming.

Non Relational Databases *(non-SQL Databases)*

A **non-relational database does not use the tabular schema of rows and columns like in relational databases**



A NON-RELATIONAL DATABASE DOES NOT INCORPORATE THE TABLE MODEL. INSTEAD, DATA CAN BE STORED IN A SINGLE DOCUMENT FILE.

A RELATIONAL DATABASE TABLE ORGANIZES STRUCTURED DATA FIELDS INTO DEFINED COLUMNS.

Primary Key

**Primary Key : Primary keys must contain UNIQUE values, and cannot contain NULL values.
For data whose all attributes are same, we need primary key to differentiate between them A table can have only one primary key; and in the table, this primary key can consist of multiple columns**

Note: Primary key can be anything, a number, String, character etc.

**Note: If you use real values as a primary key like SSN or email address, it is called “Natural Key” If you use any values like 1, 2, 3, 4.....it is called “Surrogate Key.”
Surrogate key values are just numbers.**

StudentID	**FirstName**	**LastName**
10 ◄----1	| John	Walker |
11	Tom	Hanks
12 .	Kevin	Star
13*---1	| Carl	Wall |
14	Andrei	Apazniak
15	Mark	High
16	Clara	Star
17 .	John	Ocean
18 ◄----1	| John	Walker |
19	Pamela	Star
20 ◄---	| Carl	Wall |

Email	**FirstName**	**LastName**
JWalker@gmail.com	**|john**	**Walker**
THanks@gmail.com	Tom	Hanks
KStar@gmail.com	Kevin	Star
CWall@gmail.com	**Carl**	**Wall |**
AApazniak@gmail.com	Andrei	Apazniak
MHigh@gmail.com	Mark	High
CStar@gmail.com	Clara	Star
JOcean@gmail.com	John	Ocean
JWalkerOI @gmail.com	**|john**	**Walker**
PStar@gmail.com	Pamela	Star
CWallO1@gmail.com	**|Carl**	**Wall** **|**

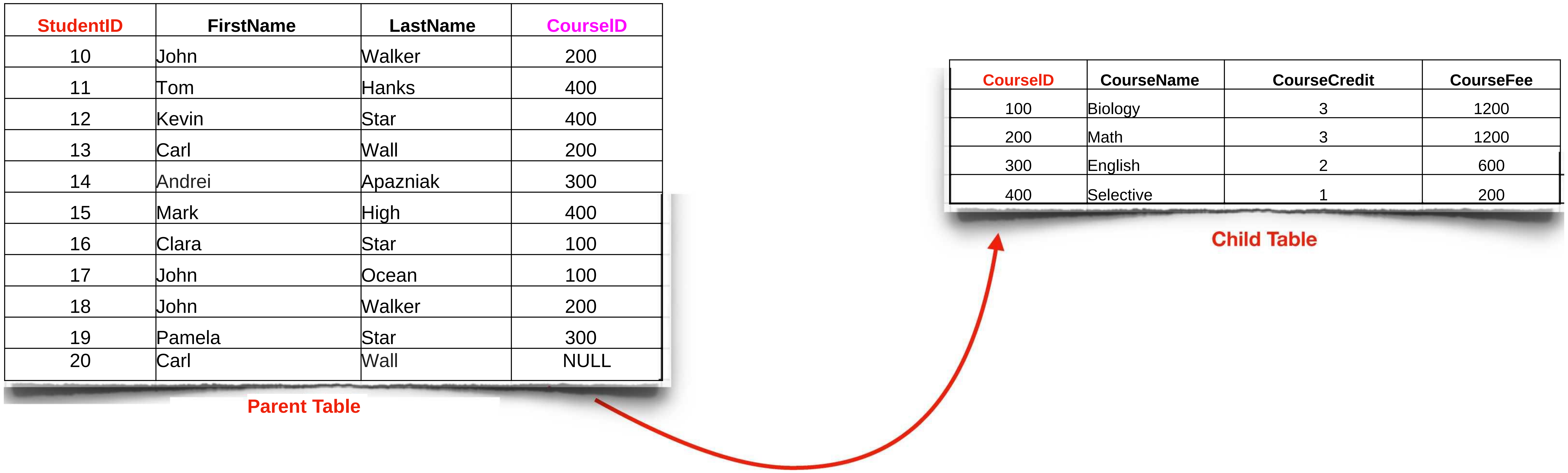
Foreign Key

A **Foreign Key** is a key used to create **link between two tables.**

A **Foreign Key** is a column ***(or collestion of column)*** in one table that **refers to the Primary Key in another table**.

A table can have many **Foreign Keys**

Foreign Key can have NULL values and repeated values



StudentID	FirstName	LastName	CourseID
10	John	Walker	200
11	Tom	Hanks	400
12	Kevin	Star	400
13	Carl	Wall	200
14	Andrei	Apazniak	300
15	Mark	High	400
16	Clara	Star	100
17	John	Ocean	100
18	John	Walker	200
19	Pamela	Star	300
20	Carl	Wall	NULL

Parent Table

CourseID	CourseName	CourseCredit	CourseFee
100	Biology	3	1200
200	Math	3	1200
300	English	2	600
400	Selective	1	200

Child Table

The "CourseID" column in the "Child Table" table is the primary key.
The"CourseID"columninthe “Parent Table" table is a foreign key.

Foreign and Primary Key

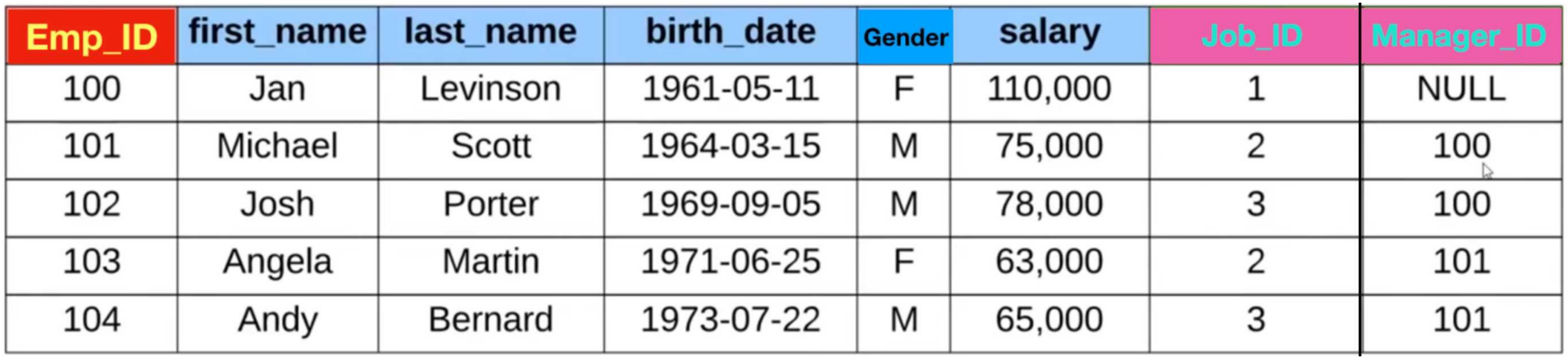
Note: Foreign key can create a relation between the table and the table itself.

1) Who is the Manager of Michael Scott ?

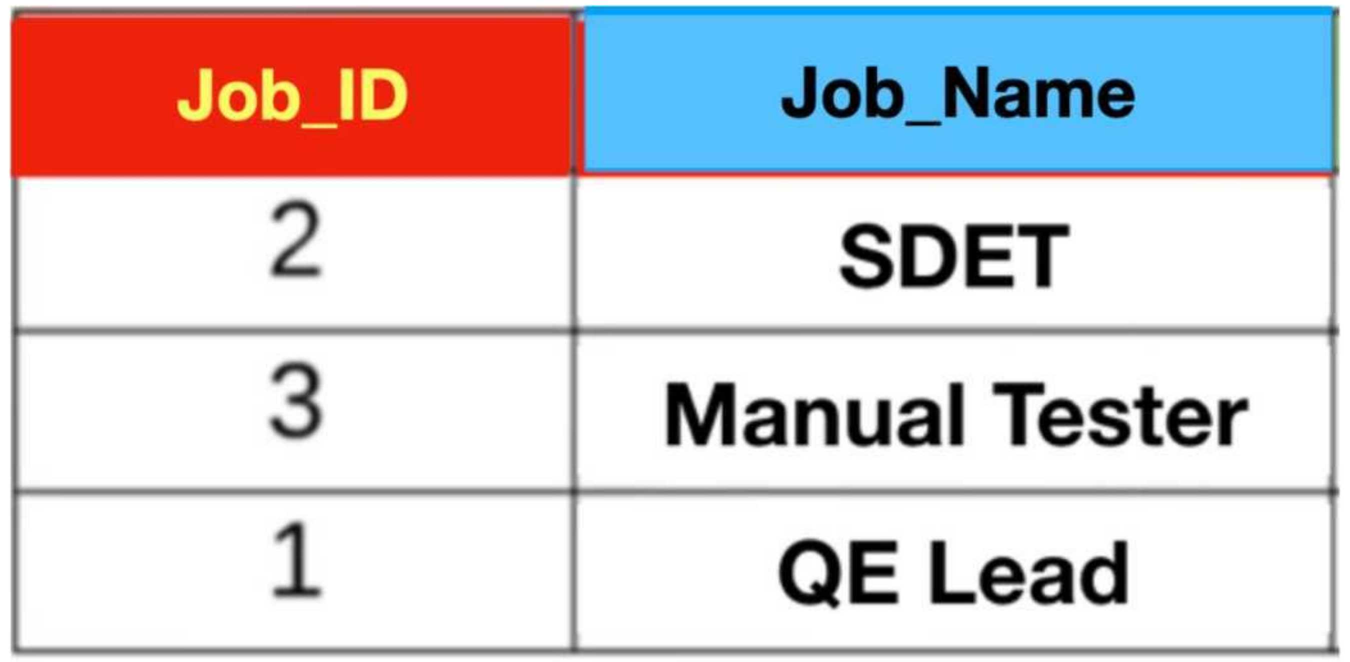
2) What is the job name of Angela Martin ?

3) What is the average salary of Manual Testers ?

4) What is the job name of the highest salary ?



Emp_ID	first_name	last_name	birth_date	Gender	salary	Job_ID	Manager_ID
100	Jan	Levinson	1961-05-11	F	110,000	1	NULL
101	Michael	Scott	1964-03-15	M	75,000	2	100
102	Josh	Porter	1969-09-05	M	78,000	3	100
103	Angela	Martin	1971-06-25	F	63,000	2	101
104	Andy	Bernard	1973-07-22	M	65,000	3	101

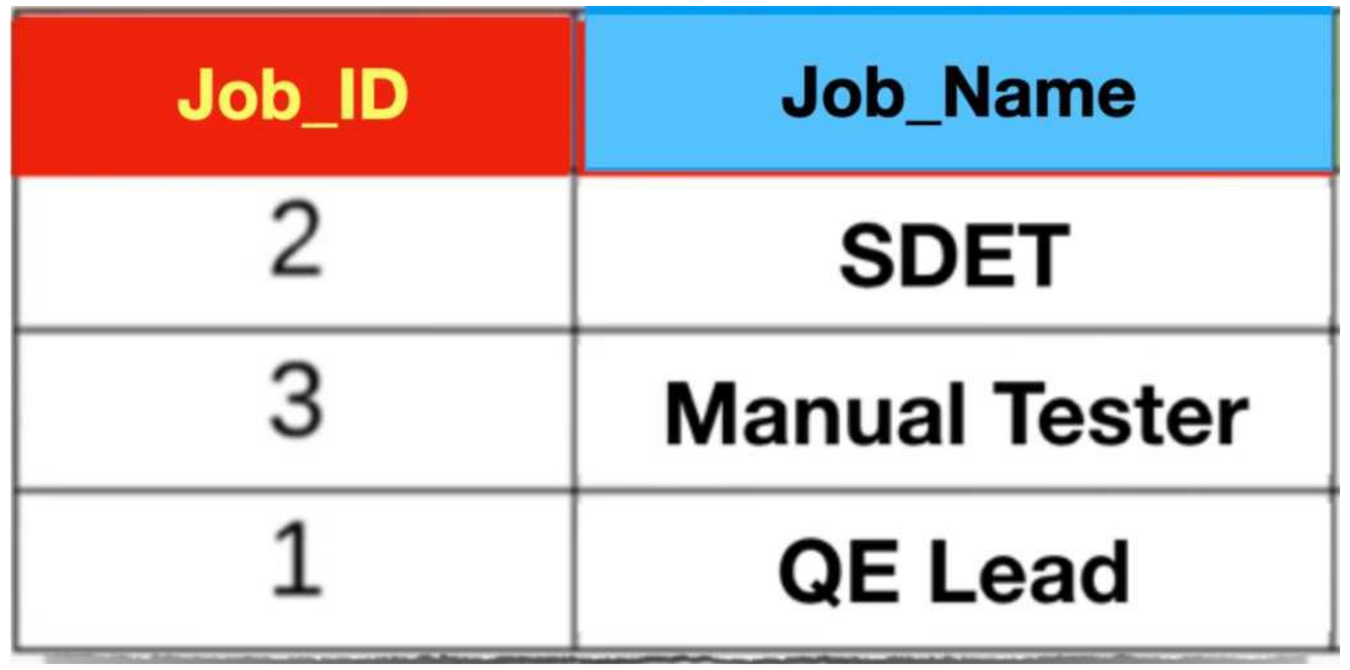


Job_ID	Job_Name
2	SDET
3	Manual Tester
1	QE Lead

SQL Composite Key

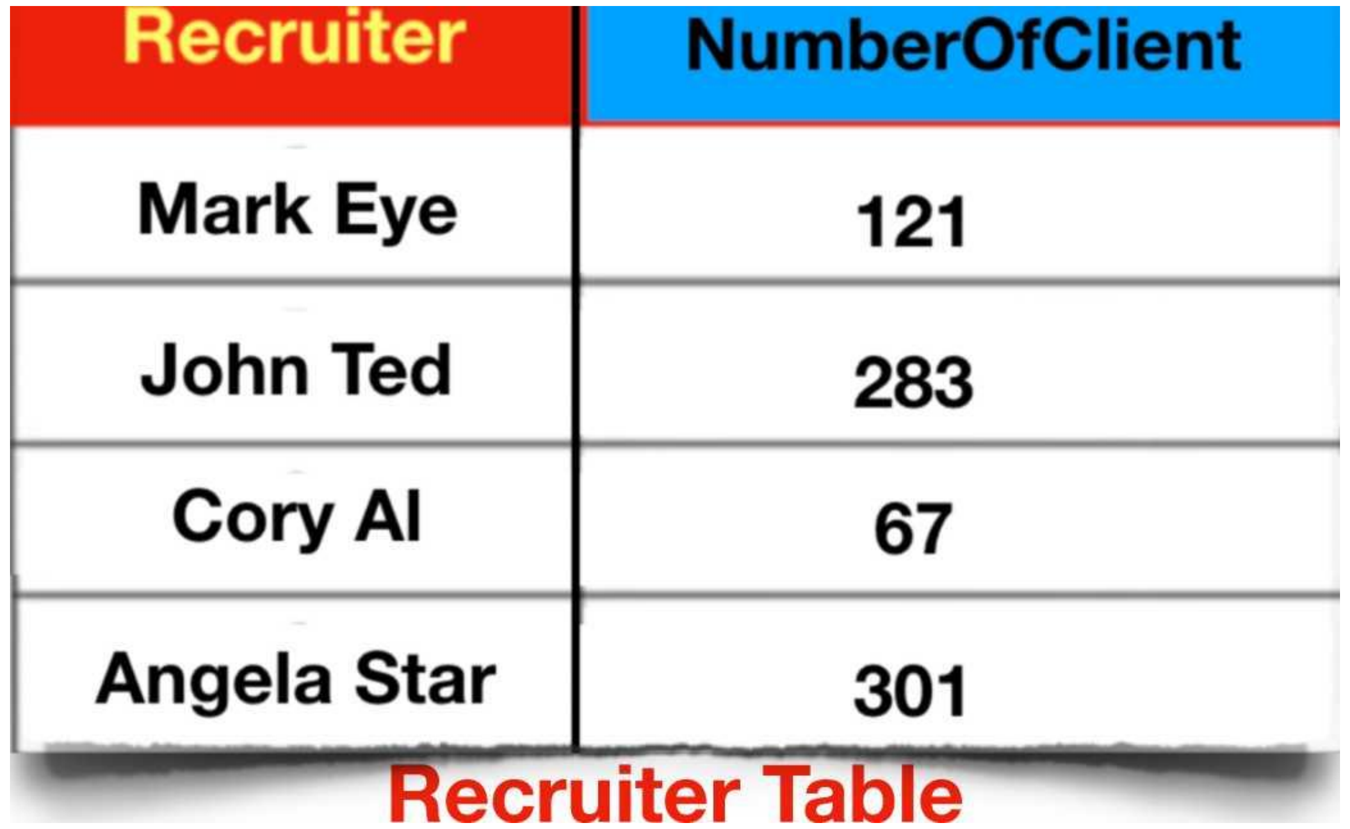
A composite key is a combination of two or more columns in a table that can be used to uniquely identify each row in the table when the columns are combined uniqueness is guaranteed, but when it taken individually it does not guarantee uniqueness.

Note: BranchJD and Recruiter are the primary keys for the Job and Recruiter tables; in addition, they are foreign key for the Company table.
The combination of JobJD and Recruiter foreign keys in Company table is primary key for Company table.



Job_ID	Job_Name
2	SDET
3	Manual Tester
1	QE Lead

Job Table



Recruiter	NumberOfClient
Mark Eye	121
John Ted	283
Cory Al	67
Angela Star	301

Recruiter Table

JobJD	Recruiter 1	
2 ▶	Mark Eye	RCG
3	John Ted	RCG
1 1	Mark Eye	Signature 1
i	John Ted	Info Log
1	Cory Al	Info Log
2	Angela Star	Signature

Company Table

Difference between “UNIQUE KEY” and “PRIMARY KEY”

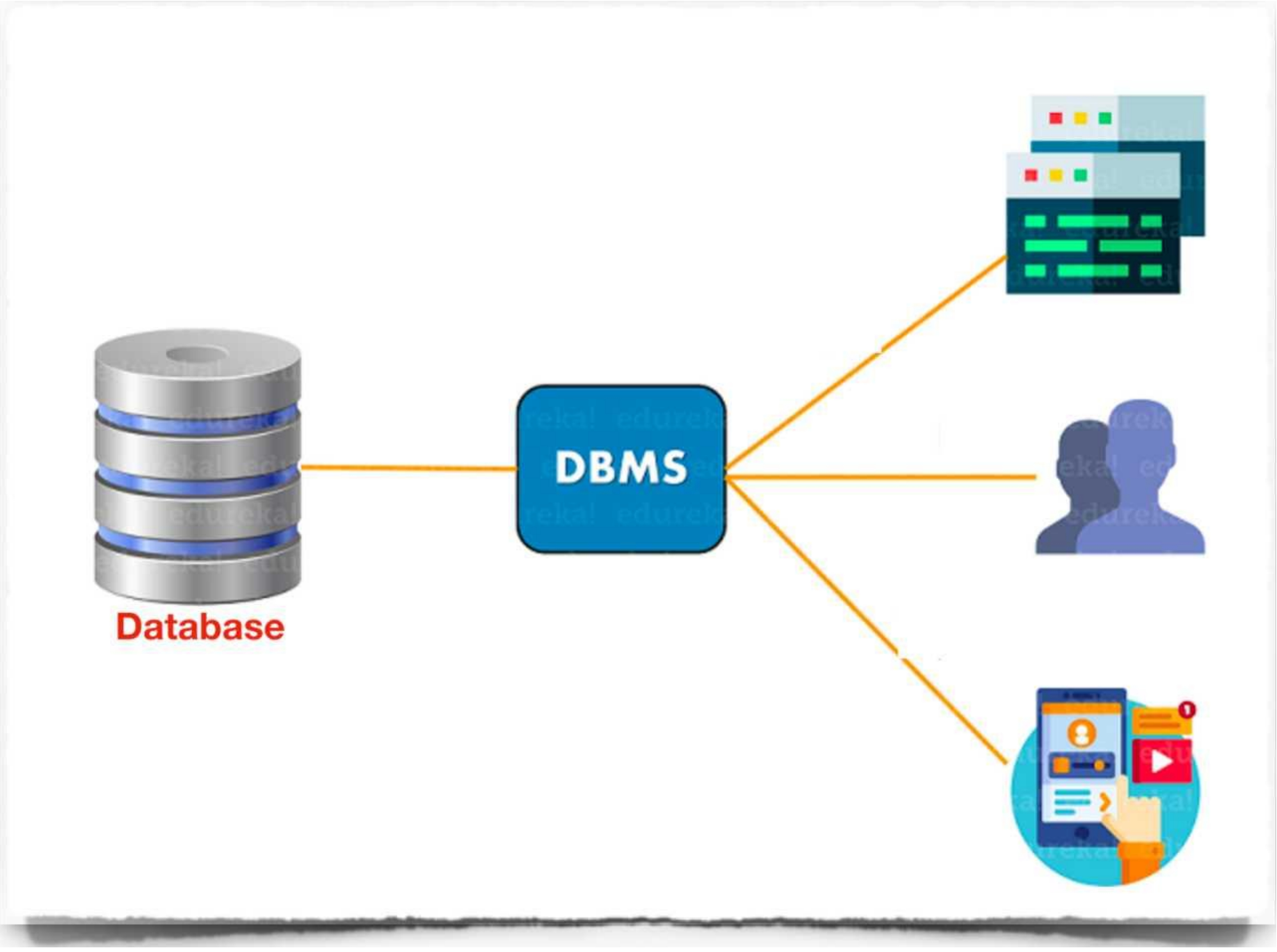
Primary Key	Unique Key
Only one primary key is allowed to use in a table.	A table can have more than one unique key.
Primary key does not accept NULL values.	Unique key constraints can accept just one NULL value for column.

Common features of “UNIQUE KEY” and “PRIMARY KEY”

Primary Key	Unique Key
A primary key of one table can be referenced by the foreign key of another table.	Unique keys are also referenced by the foreign key of another table.
Primary key does not allow duplication	Unique key also does not allow duplication

What is SQL

SQL stands for Structured Query Language



SQL is a language used for interacting with Relational Data Base Management Systems (RDBMS)

By using SQL we can;

1) Create and Manage databases
2) Create and Design database tables
3) Create, Read, Update, and Delete data (CRUD)

4) Perform administration tasks like security, user management etc.

We can use SQL for all RDBMS (MySQL, Microsoft SQL, PostgreSQL, Oracle SQL) The concepts are same but implementation can be slightly different.

More about SQL

SQL is the combination of 4 different languages;

1) Data Control Language (DCL)

DCL is used to control privileges in Database. To perform any operation in the database, such as for creating tables, sequences or views, a user needs privileges.

DCL manages users and permissions

2) Data Definition Language (DDL)

DDL deals with descriptions of the database schema (tables, columns, rows) **and is used to create and modify the structure of database objects**

3) Data Manipulation Language (DML)

DML deals with the manipulation of data present in the database. For example, insert, update, and delete data

4) Data Query Language (DQL)

DQL is used to query the database for information

DQL is used to get information that is already stored in database

Working with Related Tables

<==== One to One Relation ====>

1) Find the address of the Tom Hanks
2) Find the address of the John Walker
3) Find the address of the student whose ID is 17

	StudentID	FirstName	LastName				StudentID	Street	ZipCode	City	State
	10	John	Walker				10	1234 W 23th Street	33018	Hialeah	Florida
	11	Tom	Hanks				11	1235 N 3th Street	22145	Austwell	Texas
	12	Kevin	Star				12	1236 SE 12th Street	54234	Orange	California
	13	Carl	Wall				13	1237 N 5th Street	33018	Hialeah	Florida
	14	Andrei	Apazniak				14	1238 SW 53th Street	33026	Miami	Florida
	15	Mark	High				15	1239 S 123th Street	22314	Avery	Texas
	16	Clara	Star				16	1240 N 1 st Street	12345	Arlington	Virginia
	17	John	Ocean				17	1241 NW 2nd Street	65432	Pittsburgh	Pensylvania
	18	John	Walker				18	1242 W 5th Street	22133	Baytown	Texas
	19	Pamela	Star				19	1243 SE 55th Street	74352	Beachwood	Ohio
	20	Carl	Wall				20	1244 SW 17th Street	22314	Avery	Texas

<==== one to Many Relation ====>

1) Find the names of the students who take Biology class

2) Find the names of the students who take Selective class

3) Find the names of the students who take the class whose course fee is 600

CourseID	CourseName	CourseCredit	CourseFee	InstructorID			StudentID	FirstName	LastName	CourseID
100	Biology	3	1200	1			10	John	Walker	200
200	Math	3	1200	2			11	Tom	Hanks	400
300	English	2	600	3			12	Kevin	Star	400
400	Selective	1	200	1			13	Carl	Wall	200
							14	Andrei	Apazniak	300
							15	Mark	High	400
							16	Clara	Star	100
							17	John	Ocean	100
							18	John	Walker	200
							19	Pamela	Star	300
							20	Carl	Wall	400

<==== Many to Many Relation ====>

To resolve Many to Many relation we need Linking Table

1) Find the names of the students whose instructor is Mark Adam

2) Find the names of the instuctors of Kevin Star

3) Find the names of the instuctors of Pamela Star

StudentID	FirstName	LastName			StudentID	InstructorID		InstructorID	FirstName	LastName	Phone	Department
10	John	Walker			12	1		1	Mark	Adam	1234567891	Science
11	Tom	Hanks			11	2		2	Eve	Sky	1239876543	Engineering
12	Kevin	Star			12	2		3	Leo	Ocean	1237845691	Language
13	Carl	Wall			13	1		4	Andy	Mark	1232134567	Health
14	Andrei	Apazniak			15	1						
15	Mark	High			17	3						
16	Clara	Star			15	4						
17	John	Ocean										
18	John	Walker										
19	Pamela	Star										
20	Carl	Wall										

SQL Data Types

String Data Types

Data Type ***Description***

char(size)

Maximum size of **2000 bytes**.
1 character uses **1** byte. **"size"** is the **number of characters** to store. "char" is used to store character data.
Fixed length Strings.
The "char" is useful for expressions where the length of characters is always fix like SSN or ZipCode or State Abbreviations (FL, CA, ...)

nchar(size)

Maximum size of **2000 bytes**.
1 character uses **2** bytes.
"size" is the **number of characters** to store.
"**nchar**" is used to **store Unicode** Data.
It is often used to store data in **different language**s.
Fixed length Strings.

varchar2(size)

Maximum size of **4000 bytes**.
1 character uses **1** byte.
"size" is the **number of characters** to store.
Variable length string.

Value	CHAR(4)	Storage Required	VARCHAR (4)	Storage Required
1 !	| |	4 bytes	1 1	1 byte
'ab'	' ab '	4 bytes	' ab '	3 bytes
'abed'	'abed'	4 bytes	'abed'	5 bytes
'abcdefgh'	'abed'	4 bytes	'abed'	5 bytes

nvarchar2( )

Maximum size of **8000 bytes**.
1 character uses **2** byte.
"size" is the **number of characters** to store.
"**varchar**" uses **Non-Unicode** data while "**nvarchar**" uses **Unicode Data**
Variable length string.

Numeric Data Types

Data Type

Description

The “**Precision**” is a **number of digits** in a number.
The “**Scale**” is the **number of digits** to the **right of the decimal** point in a number.
For example, for **1234,56** ==> Precision is 6, and Scale is 2.

Precision (p) can range from **1 to 38**
Scale (s) can range from **-84 to 127**

number(p, s)

1) **“number(5, 2)”** is a number that has **3 digits before** the decimal and **2 digits after** the decimal. **==>** ***123,45*** *is stored as* ***123,45***

2) “**number”** defines a number that can store numeric values with the maximum range. **==>** ***12345,678*** *is stored as* ***12345,678***

3) **“number(7)”** defines a 7 digits number with scale zero. **==>** ***12345,67*** *is stored as* ***12345***
   Note: “number(7)” and **“number(7, 0)”** are **same**.

4) **“number(7, -2)”** rounds the numeric value to hundreds. **==>** ***1234567,89*** **==>** ***1234600***

5) **“number(4, 2)”** **==>** ***123,45*** **==>** Exceeds precision ***error***
   Note: **If the precision is exceeded, SQL will give error**

Date Data Types

Data Type	***Description***
DATE	"**DATE**" stores values that **include** both **date and time** with a precision of one second "**DATE**" stores the **year**, the **month**, the **day**, the **hours**, the **minutes**, and the **seconds**. The standart **"Date Format**" for input and output is "**dd - MMM - yy"** like 13 - Apr - 20 We can change the format by using "**ALTER SESSION SET NLS_DATE_FORMAT = "YYYY-MM-DD**" The new date format is 2020 - 04 - 13

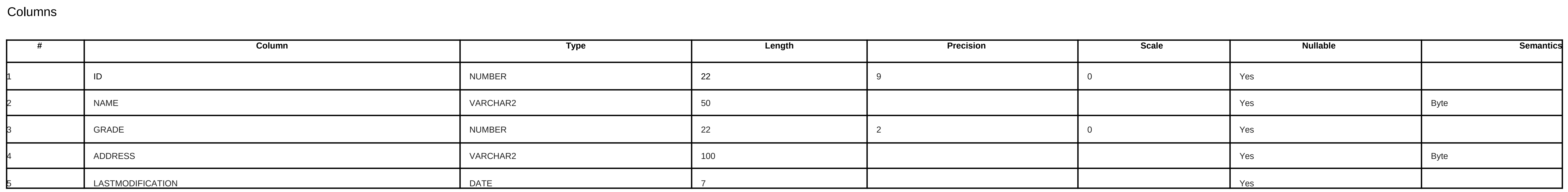
BLOB Data Types

Data Type	***Description***
BLOB	“**BLOB**” stands for “**B**inary **L**arge **OB**jects” “**BLOB**” is good to store digitized information like images, audios, and videos.

How to Create a Table

1) Create from Scratch

```
CREATE TABLE students

id number(9), name
varchar2(50), grade
number(Z), address
varchar2(100),
last_modification date );
```

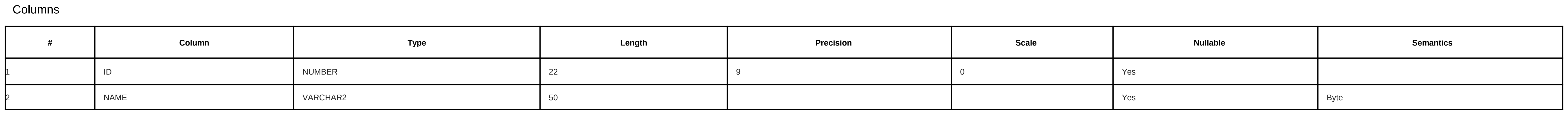


Columns

#	Column	Type	Length	Precision	Scale	Nullable	Semantics
1	ID	NUMBER	22	9	0	Yes	
2	NAME	VARCHAR2	50			Yes	Byte
3	GRADE	NUMBER	22	2	0	Yes	
4	ADDRESS	VARCHAR2	100			Yes	Byte
5	LASTMODIFICATION	DATE	7			Yes	

2) Create from an Existing Table

```
CREATE TABLE studentsIdName
AS
SELECT id, name
FROM students:
```



Columns

#	Column	Type	Length	Precision	Scale	Nullable	Semantics
1	ID	NUMBER	22	9	0	Yes	
2	NAME	VARCHAR2	50			Yes	Byte

Practice Exercise 1:

Create a table called “ *suppliers”* that stores “suppherJD”, “name”, address information which has “street”, “city”, “state”, and “zip_code” columns separately.

Practice Exercise 2:

Create a table called “ *suppliers_id_name”* that stores “supplier ID”, “name” by using “suppliers” table

How to Enforce a Column not to Accept “repeated” Values

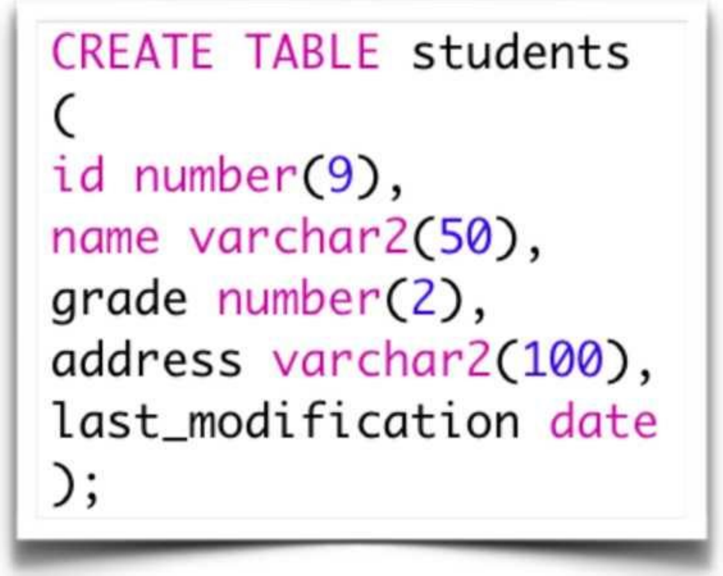
To make “id” column “not repeated”, we need to type “UNIQUE” after the id column data type

```
(CREATE TABLE students

iid char(ll), name
warchar2(50), grade
rnumber(3), address
warchar2(80), update_date
cdate ) J
```

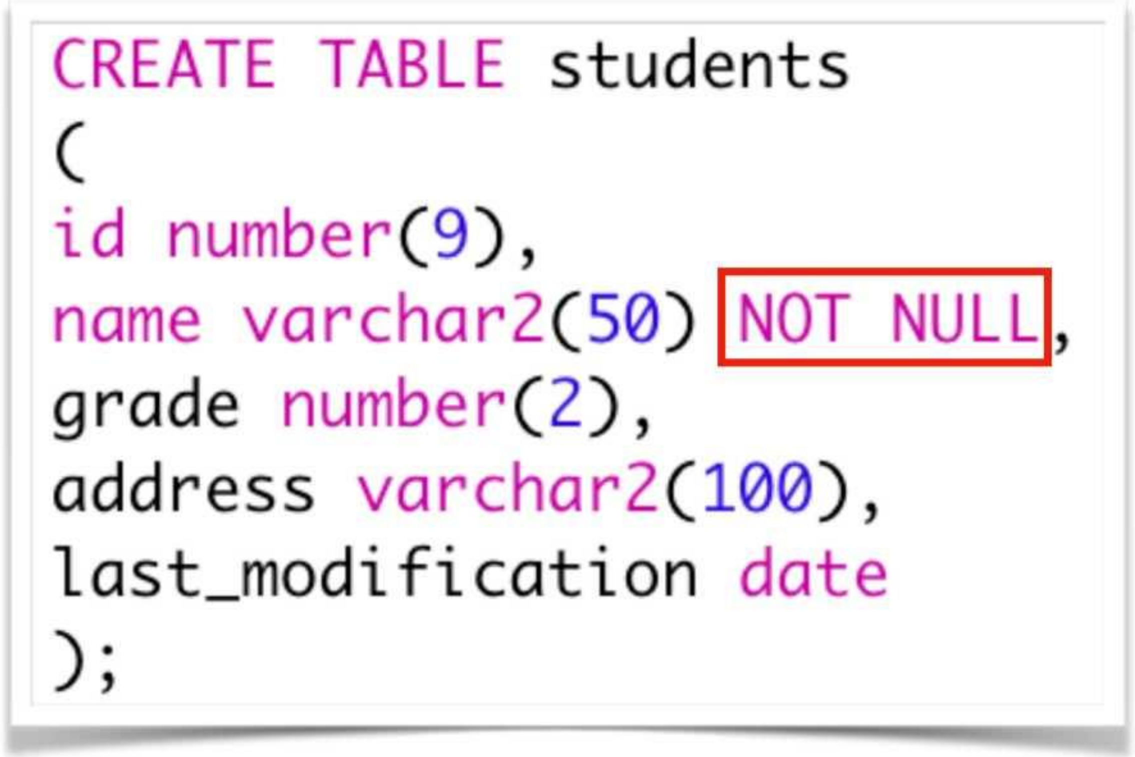
```
CREATE TABLE students

iid char(ll)| UNIQUE), name
warchar2(50), grade
rnumber(3), address
warchar2(80) update.date
cdate
```

How to Enforce a Column not to Accept “null” Values

To make “id” column “not null”, we need to type “not null” after the id column data type



```
CREATE TABLE students
(
id number(9),
name varchar2(50),
grade number(2),
address varchar2(100),
last_modification date
);
```



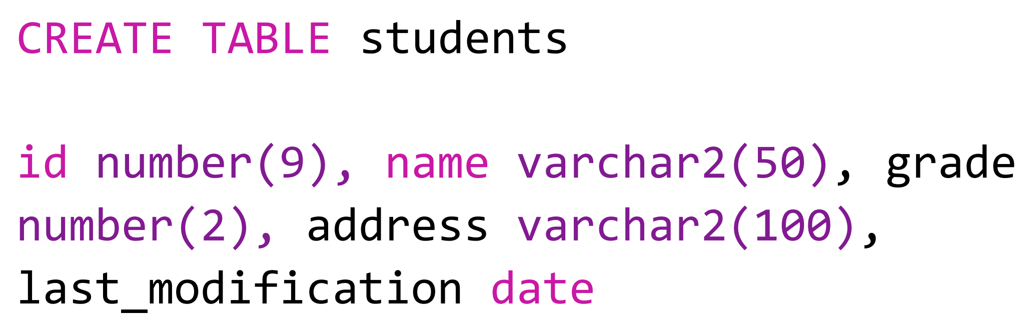
```
CREATE TABLE students
(
id number(9),
name varchar2(50) NOT NULL,
grade number(2),
address varchar2(100),
last_modification date
);
```

How to Add a “Primary Key” for a Table

1) A primary key is a single field or combination of fields that uniquely defines a record.
2) A table can have only one primary key.
3) None of the fields that are part of the primary key can contain a null value.

To Make “id” Column “primary key”

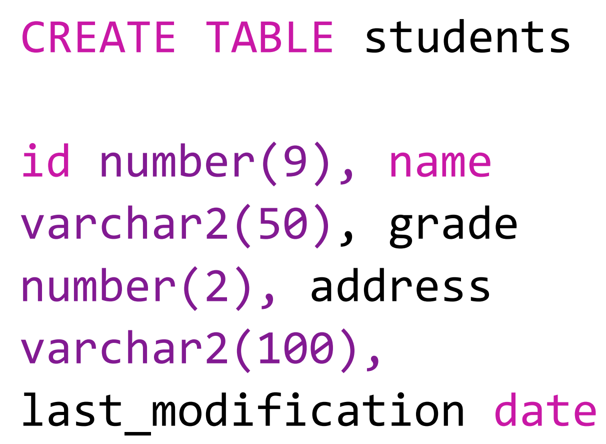
1) We can type “primary key” after the id column data type.
If you want to give a name to constraint you can type “CONSTRAINT constraintName PRIMARY KEY”



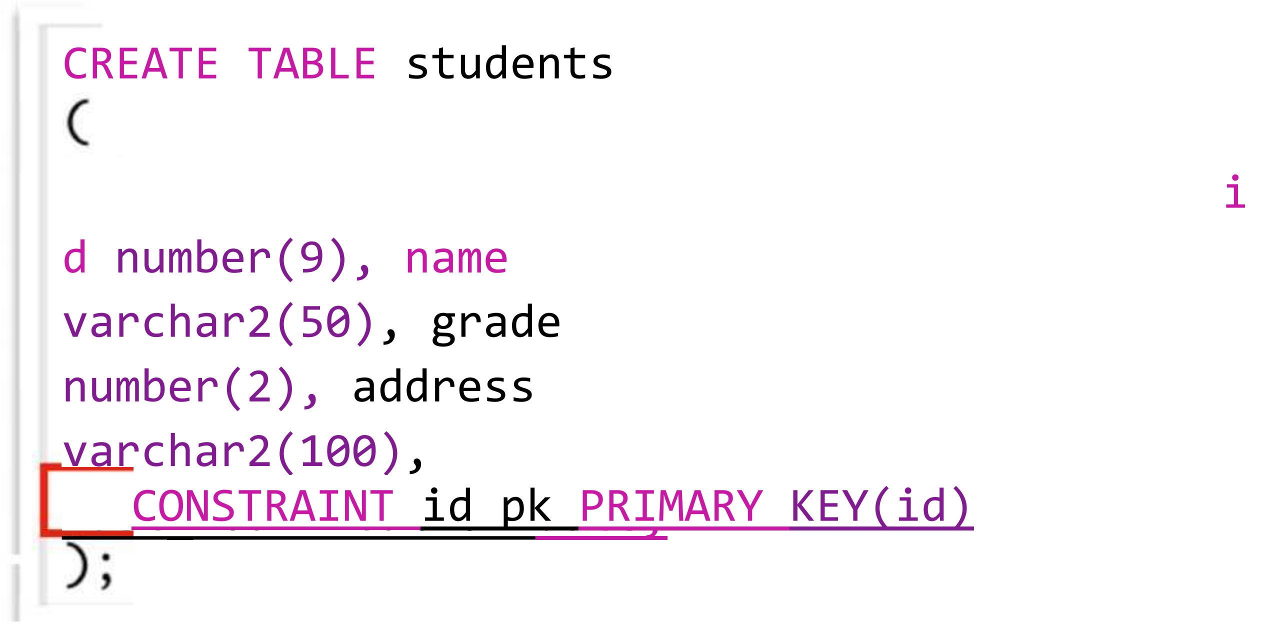
```
CREATE TABLE students

id number(9), name varchar2(50), grade
number(2), address varchar2(100),
last_modification date
```

```
CREATE TABLE students

 id number(9) primary key ,
 name varchar2(50), grade
number(Z), address
varchar2(100), last_modification
date
```

2) We can use “CONSTRAINT constraintName PRIMARY KEY(column1, column2, ... column_n)”



```
CREATE TABLE students

id number(9), name
varchar2(50), grade
number(2), address
varchar2(100),
last_modification date
```



```
CREATE TABLE students
(
                                                                  i

d number(9), name
varchar2(50), grade
number(2), address
varchar2(100),
   CONSTRAINT id pk PRIMARY KEY(id)
);
```

Practice Exercise 3:

**Create a table called “ *cities”* that stores “area code”, “name”, “population”, “state”
The “area code” will be “primary key”
Add “primary key” by using first method.**

Practice Exercise 4:

**Create a table called “ *teachers”* that stores “SSN”, “name”, “subject”, “gender”
The “SSN” will be “primary key”
Add “primary key” by using second method.**

How to Add “foreign key” to a Table

A Foreign Key is a key used to create link between two tables.
A Foreign Key is a column *(or collection of column)* **in one table that refers to the Primary Key in another table.**
The referenced table is called the *parent table* while the table with the foreign key is called the *child table*.
A table can have many Foreign Keys
Foreign Key can have NULL value

Syntax: CONSTRAINT constraintName FOREIGN KEY(column1, column2, ...) REFERENCES parentTableName(column1, column2, ...)

```
CREATE TABLE students

id number(9), name
varcharZ(50), grade
number(Z), address
varcharZ(100),
last_modification date
);
```

“ Child Table "

```
CREATE TABLE studentPhoneNumber

studentld number(9),
                    PhoneNumber varchar2C10),
CONSTRAINT studentld.fk FOREIGN KEY(studentld) REFERENCES students(id)
```

Parent Table

Note 1: If “Parent Table” does not have same student id with the “Child Table” you cannot insert data

Note 2: You cannot drop “Parent Table” without dropping the “Child Table” You need to drop “Child Table” first, then you can drop “Parent Table”

Practice Exercise 5:

Create a table called “*supplier*” that stores “supplier_id”, “supplier_name”, “contact_name” and make “supplier_id” as primary key.

Create another table called “*products*” that stores “supplier_id” and “product_id” and make “supplier_id” as foreign key.

```
CREATE TABLE supplier
C
  supplier_id number(10) not null, supplier_name
  varchar2(50) not null, contact_name varchar2(50),
   CONSTRAINT supplier.pk PRIMARY KEY (supplier_id) );
```

```
CREATE TABLE products

supplier_id number(10),
product_id number(10),
   CONSTRAINT fk_supplier FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id) );
```

Practice Exercise 6:

Create a table called “*supplier*” that stores “supplier_id”, “supplier_name”, “contact_name” and make the combination of “supplier_id” and “supplier_name” as primary key.

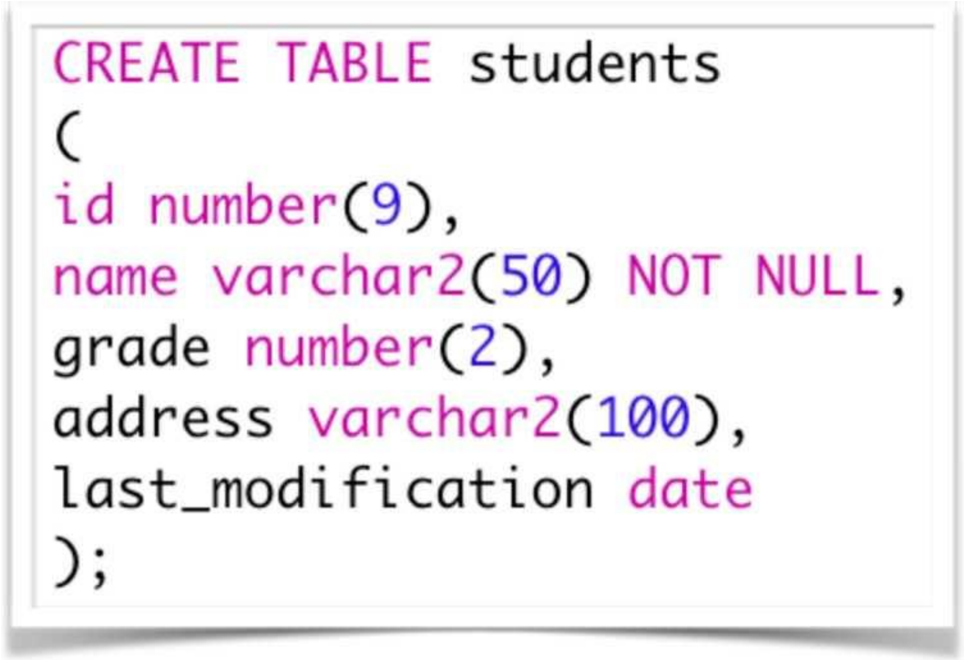
Create another table called “*products*” that stores “supplier_id” and “product_id” and make the combination of “supplier_id” and “supplier_name” as foreign key.

```
CREATE TABLE supplier

  supplier_id number(10) not null,
  supplier_name varchar2(50) not null,
  contact_name varchar2(50),
  CONSTRAINT supplier_pk PRIMARY KEY (supplier_id, supplier_name) );
```

```
CREATE TABLE products
C
product_id number(10),
supplier_id number(10),
supplier_name varchar2(50) not null,
  CONSTRAINT fk_supplier FOREIGN KEY (supplier_id, supplier_name) REFERENCES supplier(supplier_id, supplier_name) );
```

How to Insert Data Into a Table

The Oracle INSERT INTO statement is used to insert a single record or multiple records into a table in Oracle.



```
CREATE TABLE students
(
id number(9),
name varchar2(50) NOT NULL,
grade number(2),
address varchar2(100),
last_modification date
);
```

1) To insert values for all columns

```
INSERT INTO students VALUES(123456789, ’John Walker’, 11, ’1234 W 12th TER Addison Texas 75001', '14-Apr-2020' );
```

```
    ID        NAME GRADE            ADDRESS             LAST-MODIFICATION

123456789 John Walker 11     1234 W 12th TER Addison Texas 75001   14-APR-20
```

2) To insert values for some selected columns

```
INSERT INTO students(id, name) VALUES(234567890, ’John Walker’);
```

```
  ID       NAME GRADE ADDRESS LAST-MODIFICATION

234567890 John Walker -     -      -
```

Note: When inserting records into a table using the INSERT INTO statement, you must provide a value for every NOT NULL column.

```
CREATE TABLE students

 id number(9),
  name varcharZ(50) NOT NULL,
 grade number(Z), address
 varcharZ(100), last_modification date
 ) ;
```

```
INSERT INTO studentsfid, grade) VALUES(1Z3456789, 11);
```

```
ORA-01400: cannot insert NULL into ("SQL_LFGUHVRSOWWDACLEMHRMQGCJQ"."STUDENTS"."NAME") ORA-06512: at "SYS.DBMS_SQL", Line 1721
```

Practice Exercise 7:

Create an Oracle table called “ *teachers*” that stores “SSN”, “name”, “subject”, “gender”
Based on the *“teachers”* table, insert a contact record whose *SSN* is 234 43 1223, *name* is Jane Smith, subject is Mathematics, and gender is female.

Practice Exercise 8:

Based on the *“teachers”* table, insert a contact record whose *SSN* is 567 59 7624, *name* is Leo Mark

How to use UPDATE SET

The UPDATE SET statement is used to update existing records in a table.

1)

```
CREATE TABLE supplier
C
  supplier_id number(10),
  supplier_name varchar2(50),
  contact_name varchar2(50),
  CONSTRAINT supplier_pk PRIMARY KEY Csupplier_id, supplier_name)
);
```

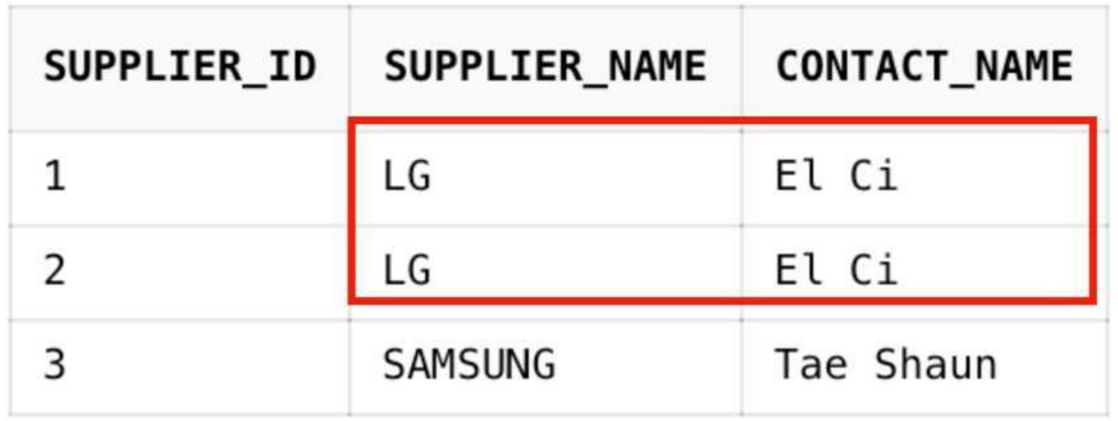
```
INSERT INTO supplier VALUES(1, 'IBM', 'John Walker');
  INSERT INTO supplier VALUES(2, 'APPLE', 'Steve Max');
INSERT INTO supplier VALUES(3, 'SAMSUNG', 'Tae Shaun');
```

SUPPLIER_ID	SUPPLIER-NAME	CONTACT-NAME
1	IBM	John Walker
2	APPLE	Steve Max
3	SAMSUNG	Tae Shaun

```
UPDATE supplier
SET supplier_name = ’LINUX',
      contact_name = 'Alex Leo'
WHERE supplier_id=l;
```

SUPPLIER_ID	SUPPLIER-NAME	CONTACT-NAME
1	LINUX	Alex Leo
2	APPLE	Steve Max
3	SAMSUNG	Tae Shaun

```
     UPDATE supplier
SET supplier_name = ’LG’,
      contact_name = ’El Ci’
     WHERE supplier_id<3;
```



SUPPLIER_ID	SUPPLIER_NAME	CONTACT_NAME
1	LG	El Ci
2	LG	El Ci
3	SAMSUNG	Tae Shaun

2)

```
CREATE TABLE supplier
C
  supplier_id numberC10),
  supplier.name varchar2(50),
  contact_name varchar2C50),
  CONSTRAINT supplier.pk PRIMARY KEY Csupplier_id, supplier_name)
);
```

```
INSERT INTO supplier VALUES(1, 'IBM', 'John Walker');
INSERT INTO supplier VALUES(2, 'APPLE', 'Steve Max');
INSERT INTO supplier VALUES(3, 'SAMSUNG', 'Tae Shaun');
```

SUPPLIER_ID	SUPPLIER_NAME	CONTACT_NAME
1	IBM	John Walker
2	APPLE	Steve Max
3	SAMSUNG	Tae Shaun

```
CREATE TABLE products
C
  supplier_id number(10),
  product-id number(10),
  product_name varchar2C50),
  customer_name varchar2(50),
  CONSTRAINT fk_supplier FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id) );
```

```
INSERT INTO products VALUESfl, 11, ’Laptop’, ’John Walker’);
INSERT INTO products VALUES(2, 22, ’Ipad’, ’Eddie Murphy’);
INSERT INTO products VALUES(3, 33, ’Galaxy 10’, ’Adam Eve’);
```

SUPPLIER_ID	PRODUCT_ID	PRODUCT_NAME	CUSTOMER_NAME
1	11	Laptop	John Walker
2	22	Ipad	Eddie Murphy
3	33	Galaxy 10	Adam Eve

```
UPDATE supplier
SET supplier_name = (SELECT product_name
                     FROM products
                     WHERE supplier.contact_name = products.customer_name)
WHERE supplier_id < 3;
```

SUPPLIER_ID	SUPPLIER_NAME	CONTACT_NAME
1	Laptop	John Walker
2	—	Steve Max
3	SAMSUNG	Tae Shaun

Note: This UPDATE SET example updates only the *“supplier”* table for all records where the *supplier_id* is less than 3. When the *contact_name* from the *suppliers* table matches the *customer_name* from the *products* table.

Practice Exercise 9:

a) Create a table called “ *students”* that stores “student_id”, “student_name”, “student_grade”, “student_gpa”, “school_name” b) Insert 5 different data with 2.6, 1.9, 3.2, 3.8, 3.5 GPA scores.

c) Update the student names whose GPAs are more than 3.0 to “Gifted Student”.

a) Create a table called “ *students”* that stores “student_id”, “student_name”, “student_grade”, “school_name” b) Insert 5 different data with 2.6, 1.9, 3.2, 3.8, 3.5 GPA scores.

c) Create a table called “*parents”* that stores “student_id”, “parent_name”, “school_name”

d) Insert 5 different data with at least 2 same school names with the students table.

e) Update the *student names* in the *students* table with the *parent name* in the *parents* table when the *school name* in the *students* table matches the school *name* in the *parents* table.

```
CREATE TABLE suppliers
(
supplier_id number(11) PRIMARY KEY,
supplier_name varchar2(50), contact_name
varchar2(50)
);
```

```
INSERT INTO suppliers VALUES(100, 'IBM', 'Ali Can');
INSERT INTO suppliers VALUES(101, 'APPLE', 'Merve Temiz');
INSERT INTO suppliers VALUES(102, 'SAMSUNG', 'Kemal Can');
INSERT INTO suppliers VALUES(103, 'LG', 'Ali Can');
```

```
CREATE TABLE products (
supplier_id number(11), product_id
number(11), product_name
varchar2(50), costumer_name
varchar2(50)
costumer_name varchar2(50),
CONSTRAINT supplier_id_fk FOREIGN KEY(supplier_id) REFERENCES suppliers(supplier_id );
```

```
INSERT INTO products VALUES(100, 1001,'Laptop', 'Suleyman');
INSERT INTO products VALUES(101, 1002,'iPad', 'Fatma');
INSERT INTO products VALUES(102, 1003,'TV', 'Ramazan');
INSERT INTO products VALUES(103, 1004,'Phone', 'Ali Can');
```

Practice Exercise 11:

According to the given tables do the followings

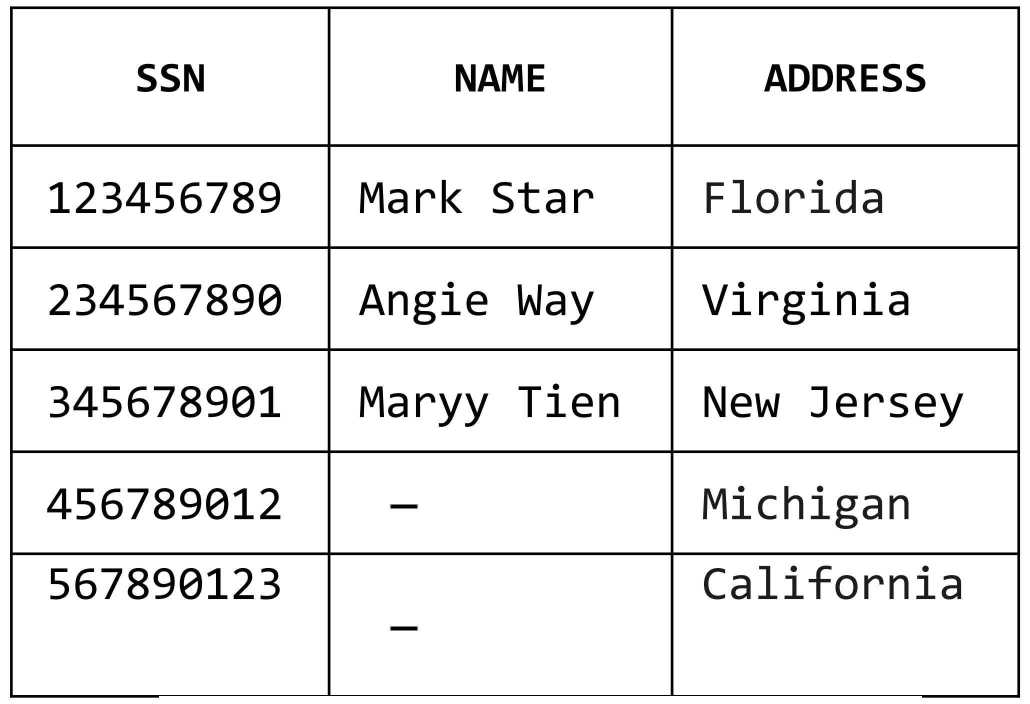
a) Change the product which Ali Can purchased to the supplier name which Merve Temiz is contact person

b) Change the customer name who purchased TV to the contact name of Apple

“IS NULL” Condition

```
CREATE TABLE people
( ssn char(9), name
varchar2(50), address
varchar2(50) );
```

```
INSERT INTO people VALUES(123456789, 'Mark Star', 'Florida');
INSERT INTO people VALUES(234567890, 'Angie Way', 'Virginia');
  ► INSERT INTO people VALUES(345678901, 'Maryy Tien', 'New Jersey');
INSERT INTO people(ssn, address) VALUES(456789012, 'Michigan');
INSERT INTO people(ssn, address) VALUES(567890123, 'California');
```

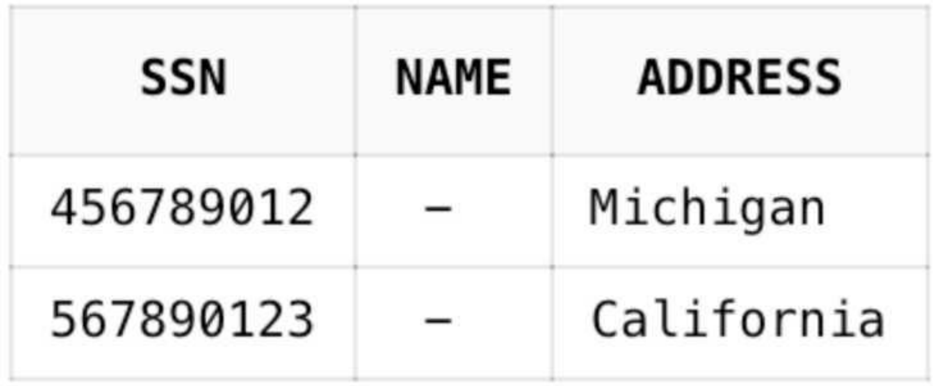


SSN	NAME	ADDRESS
123456789	Mark Star	Florida
234567890	Angie Way	Virginia
345678901	Maryy Tien	New Jersey
456789012	–	Michigan
567890123	–	California

Table name is “people”

Example: Return all records from the *people* table where the *name* contains a null value.

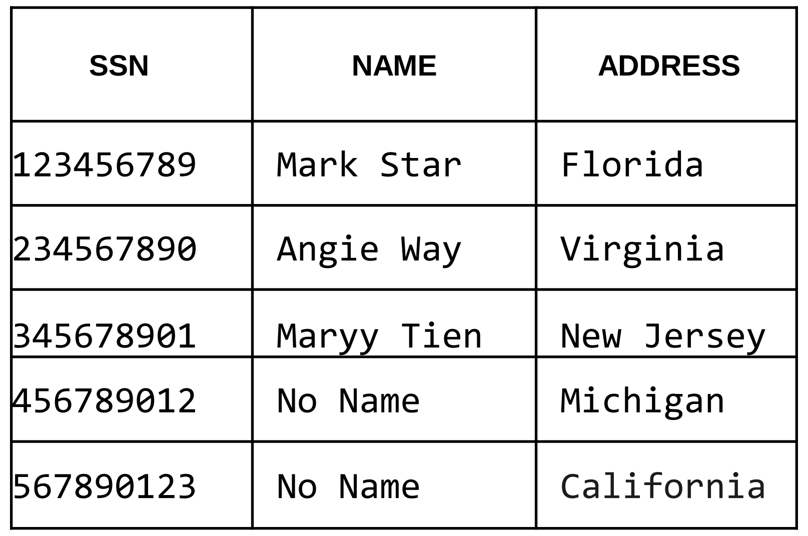
```
SELECT * FROM
people WHERE name IS
NULL;
```



SSN	NAME	ADDRESS
456789012	–	Michigan
567890123	–	California

Example: Update all null names to “No Name” from the *people* table

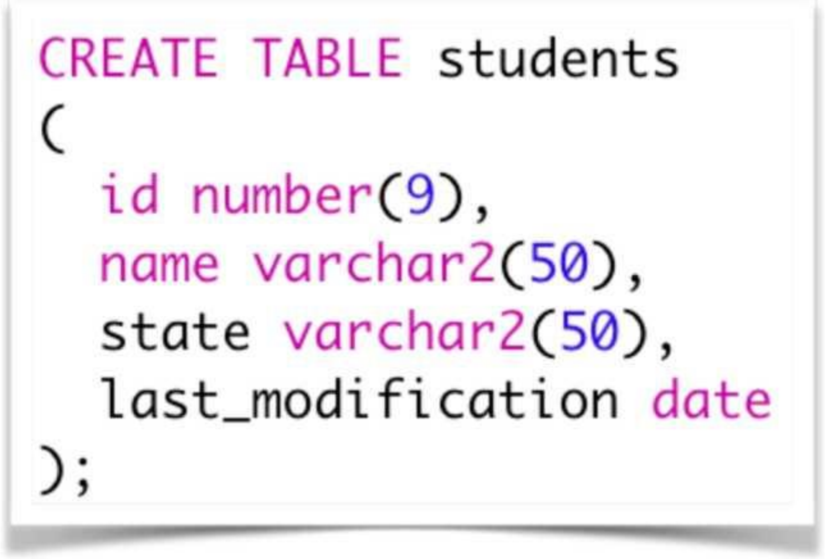
```
UPDATE people
SET name = 'No Name'
WHERE name IS NULL;
```



SSN	NAME	ADDRESS
123456789	Mark Star	Florida
234567890	Angie Way	Virginia
345678901	Maryy Tien	New Jersey
456789012	No Name	Michigan
567890123	No Name	California

How to Delete Data from a Table

1) “DELETE FROM students” deletes all inserted data inside the table, but it does not delete the table. After using “DELETE FROM students”, you will have an empty table.



```
CREATE TABLE students
(
  id number(9),
  name varchar2(50),
  state varchar2(50),
  last_modification date
);
```

```
  INSERT INTO students VALUES(123456789, 'John Walker', 'Texas', '14-Apr-2020');
► INSERT INTO students VALUES(234567890, 'Eddie Murphy', 'Florida', '15-Apr-2020');
  INSERT INTO students VALUES(345678901, 'Adam Eve', 'New York', '16-Apr-2020');
```

ID	NAME	STATE	LAST_MODIFICATION
123456789	John Walker	Texas	14-APR-20
234567890	Eddie Murphy	Florida	15-APR-20
345678901	Adam Eve	New York	16-APR-20

```
DELETE FROM students
```

```
ID          NAME          STATE LAST_MODIFICATION
```

Empty Table

2) “ DELETE FROM students WHERE name = ‘John Walker’ ” deletes the data whose name is John Walker.

CREATE TABLE students

id number(9),
name varchar2(50), state
varchar2(50), lastmodification
date

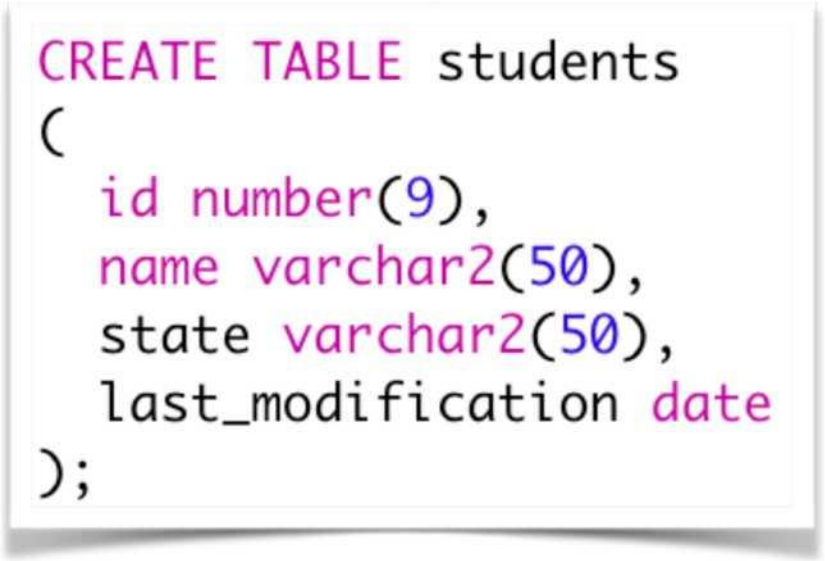
```
  INSERT INTO students VALUES(123456789, 'John Walker', 'Texas', '14-Apr-2020');
► INSERT INTO students VALUES(234567890, 'Eddie Murphy', 'Florida', '15-Apr-2020');
  INSERT INTO students VALUES(345678901, 'Adam Eve', 'New York', '16-Apr-2020');
```

ID	NAME	STATE	LAST_MODIFICATION
123456789	John Walker	Texas	14-APR-20
234567890	Eddie Murphy	Florida	15-APR-20
345678901	Adam Eve	New York	16-APR-20

```
DELETE FROM students WHERE name          'John Walker';
```

ID	NAME	STATE	LAST_MODIFICATION
234567890	Eddie Murphy	Florida	15-APR-20
345678901	Adam Eve	New York	16-APR-20

3) “ DELETE FROM students WHERE name = ‘John Walker’ OR state = ‘New York’ ” deletes the data whose name is John Walker.



```
CREATE TABLE students
(
  id number(9),
  name varchar2(50),
  state varchar2(50),
  last_modification date
);
```

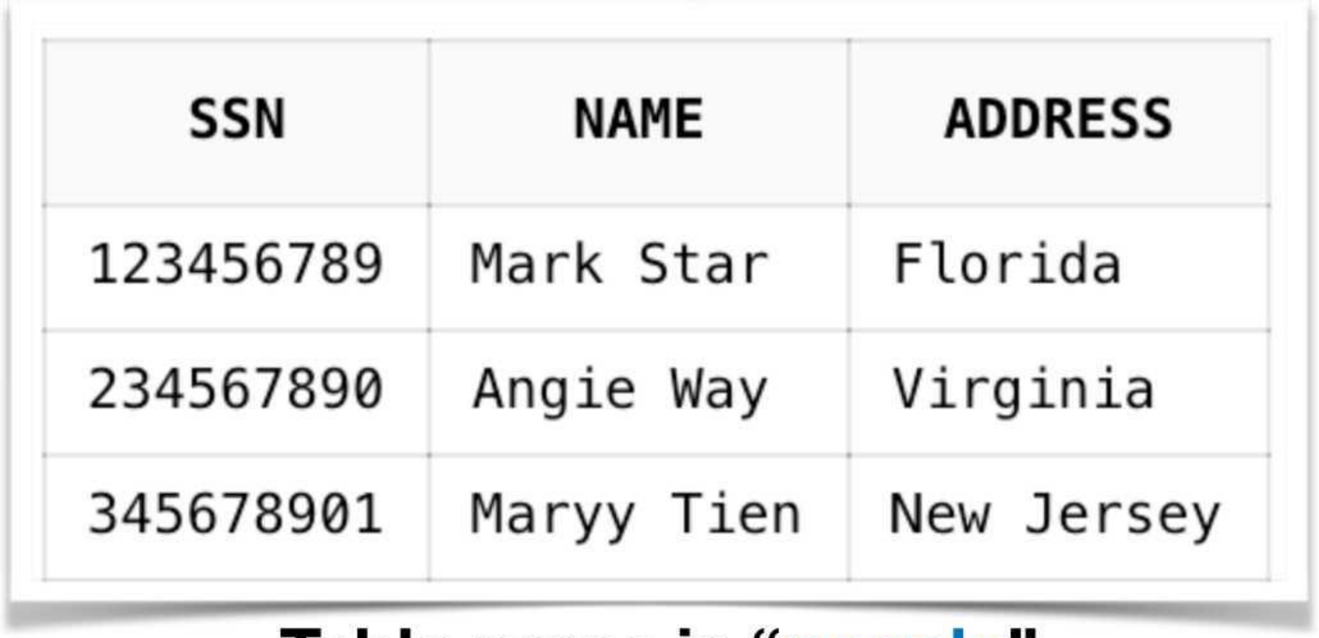
```
INSERT INTO students VALUES(123456789, 'John Walker', 'Texas', '14-Apr-2020'); ► INSERT
INTO students VALUES(234567890, 'Eddie Murphy', 'Florida', '15-Apr-2020');
INSERT INTO students VALUES(345678901, 'Adam Eve', 'New York', '16-Apr-2020');
```

ID	NAME	STATE	LAST_MODIFICATION
123456789	John Walker	Texas	14-APR-20
234567890	Eddie Murphy	Florida	15-APR-20
345678901	Adam Eve	New York	16-APR-20

```
DELETE FROM students WHERE name = ’John Walker' OR state = 'New York'; ----->
```

```
ID        NAME           STATE LAST_MODIFICATION
234567890 Eddie Murphy Florida 15-APR-20
```

Review Question



SSN	NAME	ADDRESS
123456789	Mark Star	Florida
234567890	Angie Way	Virginia
345678901	Maryy Tien	New Jersey

Table name is “people”

1) Create the given table by using SQL Queries
2) Update “Virginia” to “Pennsylvania ”
3) Delete 3rd row from the table
4) Drop the table

Note: Use SELECT * FROM people; to see the table on the console.

“Truncate” Statement

“Truncating” a table is a fast way to clear out records from a table if you don't need to worry about rolling back.

Warning: If you truncate a table, the TRUNCATE TABLE statement can not be rolled back.

```
TRUNCATE TABLE customers;
```

```
DELETE FROM customers;
```

Note: The main difference between the two is that you can roll back the DELETE FROM statement, but you can't roll back the TRUNCATE TABLE statement.

How to Drop (Deletes table contents and table structure) a Table

```
CREATE TABLE students

id number(9), name
varchar2(50), grade number(Z),
address varchar2(100),
last_modification date
```

```
*■ DROP TABLE students
```

Table with all contents and structure moved to the trash

```
CREATE TABLE students

id number(9), name
varchar2(50), grade number(2),
address varchar2(100),
last_modification date
```

```
► DROP TABLE students PURGE
```

The PURGE option will purge the table and its dependent objects so that they do not appear in the recycle bin.

Warning: The risk of specifying the PURGE option is that you will not be able to recover the table.

Benefit of PURGE: You can ensure that sensitive data will not be left sitting in the recycle bin.

“SELECT” Statement

1) Select all fields *(columns)* from one table

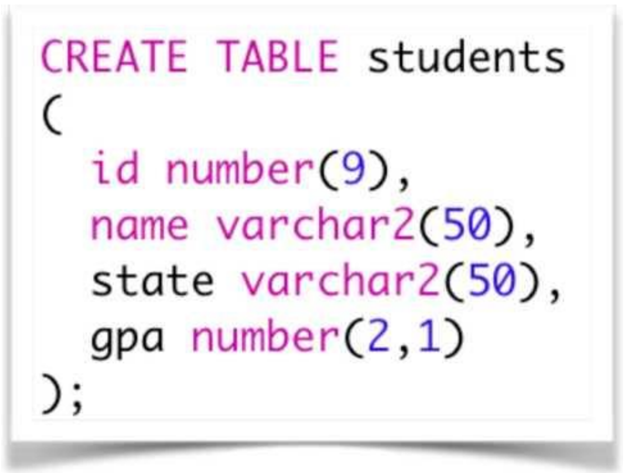
Example 1: Get all data from students table

```
CREATE TABLE students
C
  id number(9),
 name varchar2(50),
 state varchar2(50),
 gpa number(2,1)
);
```

k **SELECT * FROM students;**

ID	NAME	STATE	GPA
123456789	John Walker	Texas	2.8
234567890	Eddie Murphy	Florida	3.2
345678901	Adam Eve	New York	3.5
456789012	Alex Tien	New York	3.8
567890123	Chris Matala	Virginia	4

Example 2: Get all data from students table where GPA>3.2



```
CREATE TABLE students
(
  id number(9),
  name varchar2(50),
  state varchar2(50),
  gpa number(2,1)
);
```

SELECT *
► **FROM students**
WHERE gpa>3.2;

ID	NAME	STATE	GPA
345678901	Adam Eve	New York	3.5
456789012	Alex Tien	New York	3.8
567890123	Chris Matala	Virginia	4

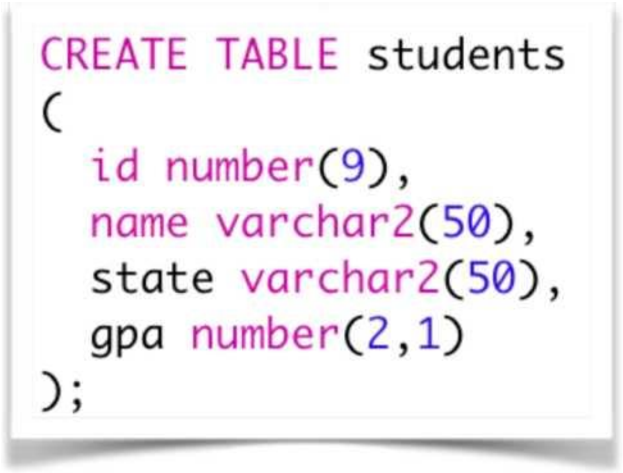
Operators to use in WHERE filter

WHERE clause is used to filter the results from a SELECT, INSERT, UPDATE, or DELETE statement.

“ = “ =	=> Equal to sign
“ > “ =	=> Greater than sign
“ < “ =	=> Less than sign
“ >= “	==> Greater than or equal to sign
“ <= “	==> Less than or equal to sign
“ < > “	==> Not Equal to sign
“AND”	==> And operator
“OR”	:=> Or operator

2) Select individual fields *(columns)* from one table

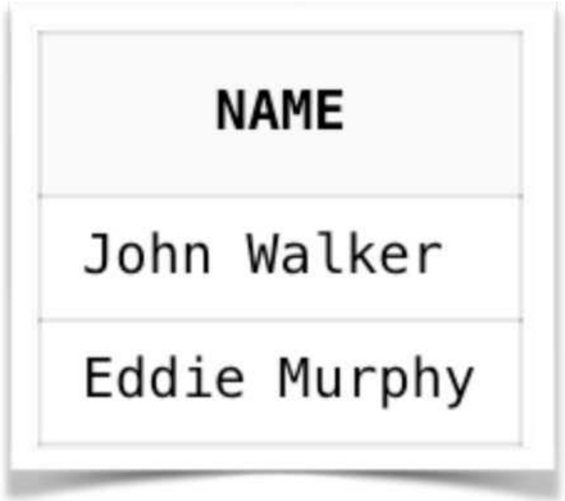
Example 1: Get the names of the students whose gpa is 2.8 OR state is Florida from students table



```
CREATE TABLE students
(
  id number(9),
  name varchar2(50),
  state varchar2(50),
  gpa number(2,1)
);
```

ID	NAME	STATE	GPA
123456789	John Walker	Texas	2.8
234567890	Eddie Murphy	Florida	3.2
345678901	Adam Eve	New York	3.5
456789012	Alex Tien	New York	3.8
567890123	Chris Matala	Virginia	4

SELECT name
▶ FROM students
WHERE gpa = 2.8 OR state ='Florida';



NAME
John Walker
Eddie Murphy

Example 2: Get the names, and id of the students whose state is New York AND gpa is 3.5 from students table

```
CREATE TABLE students
c
 id number(9), name
 varchar2(50), state
 varchar2(50),
 gpa number(2,1) );
```

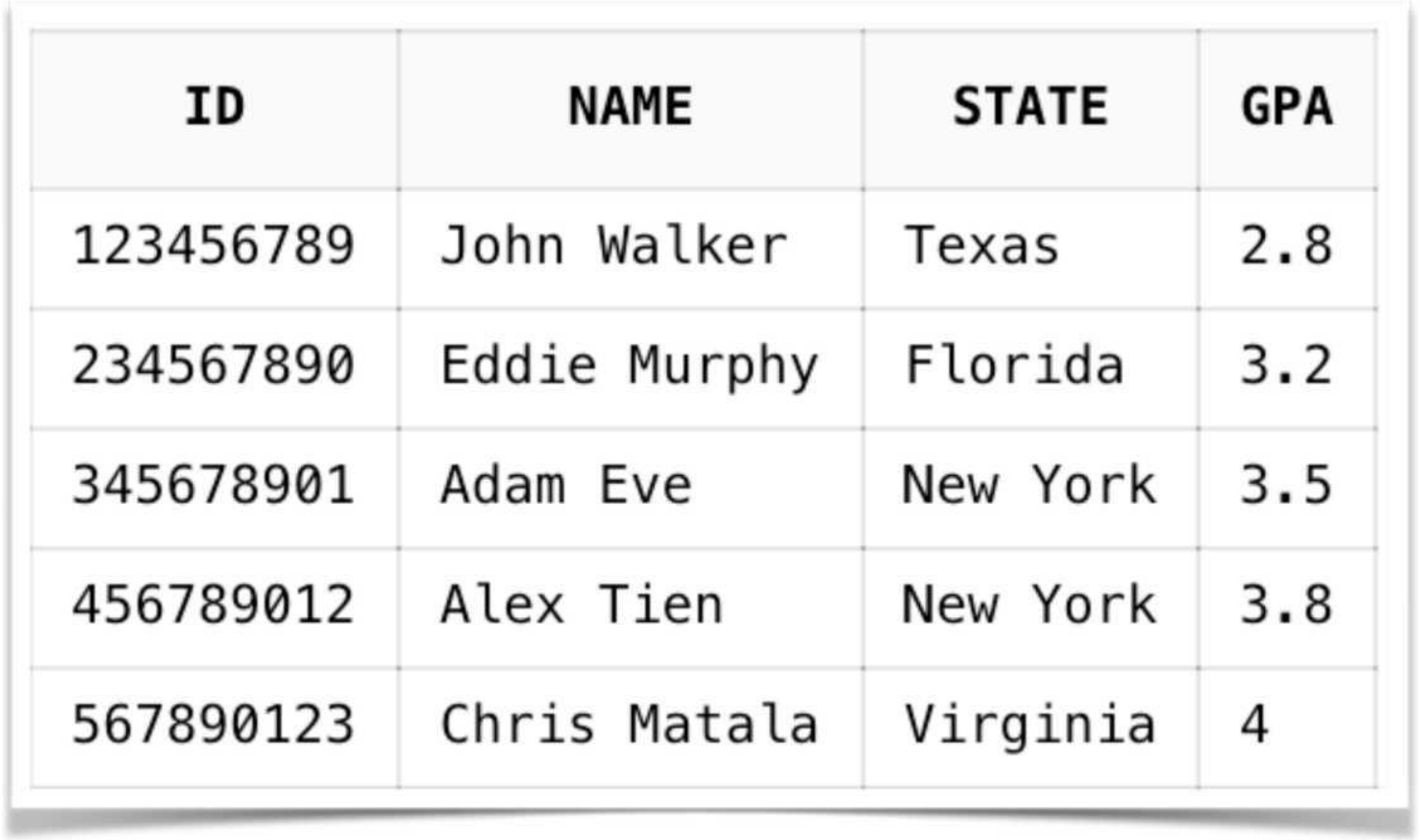
--▶

ID	NAME	STATE	GPA
123456789	John Walker	Texas	2.8
234567890	Eddie Murphy	Florida	3.2
345678901	Adam Eve	New York	3.5
456789012	Alex Tien	New York	3.8
567890123	Chris Matala	Virginia	4

SELECT name, id
—▶ FROM students —▶
WHERE state = 'New York' AND gpa = 3.5 ;

NAME	ID
Adam Eve	345678901

Practice Exercise 11:



ID	NAME	STATE	GPA
123456789	John Walker	Texas	2.8
234567890	Eddie Murphy	Florida	3.2
345678901	Adam Eve	New York	3.5
456789012	Alex Tien	New York	3.8
567890123	Chris Matala	Virginia	4

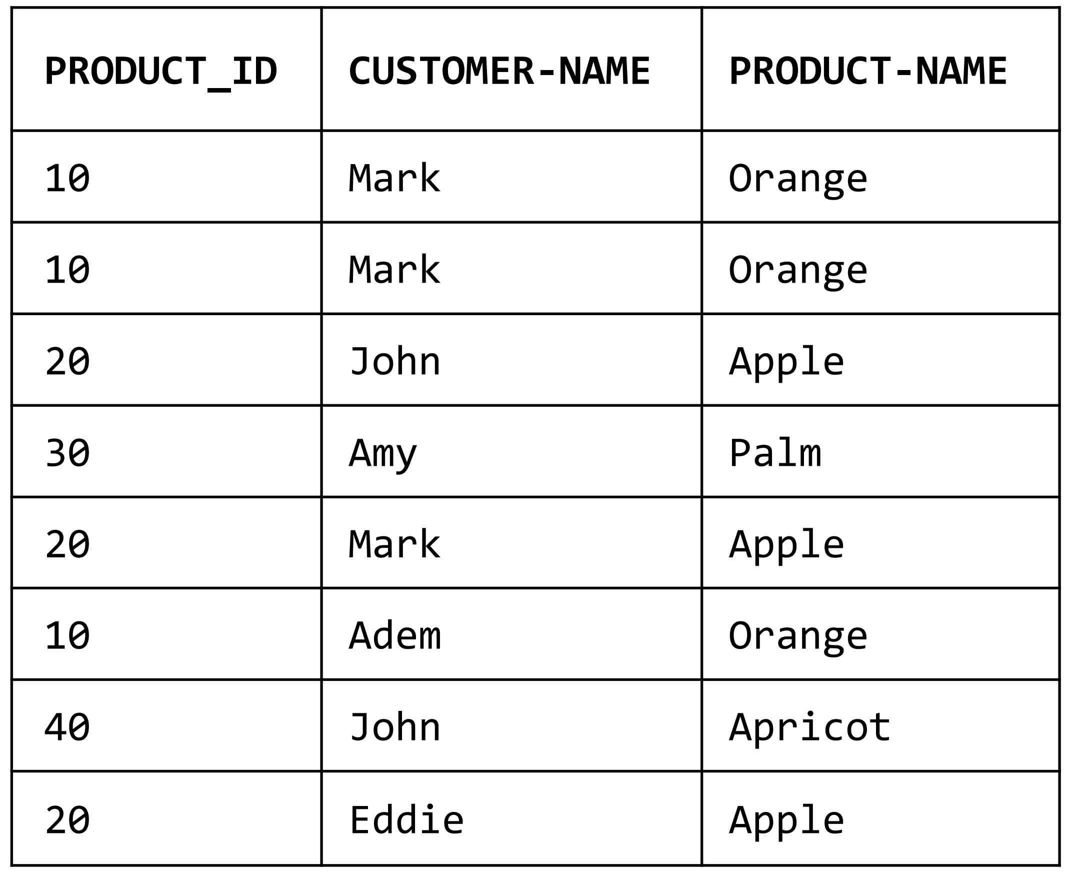
a) Create the given table
b) Select all fields from the *students* table whose *gpa* is greater than 3.1 or school name is “Texas”
c) Select students name from the *students* table whose *gpa* is less than 3.5 and state is “Florida”
d) Select students names and student ids from the *students* table whose *gpa* is between 2.8 and 3.5
e) Select all fields from the *students* table whose *state is “New York”, and gpa is greater than 3.3, and gpa is less than 3.7*
f) Select all fields from the *students* table whose *state is “New York” and gpa is greater than 3.7, or gpa is less than 3.3*

“IN” Condition

IN condition is used to help reduce the need to use multiple OR conditions in a SELECT, INSERT, UPDATE, or DELETE statement.

```
CREATE TABLE customers_products
(
  product_id number(10),
 customer_name varchar2(50),
 product_name varchar2(50)
);
```

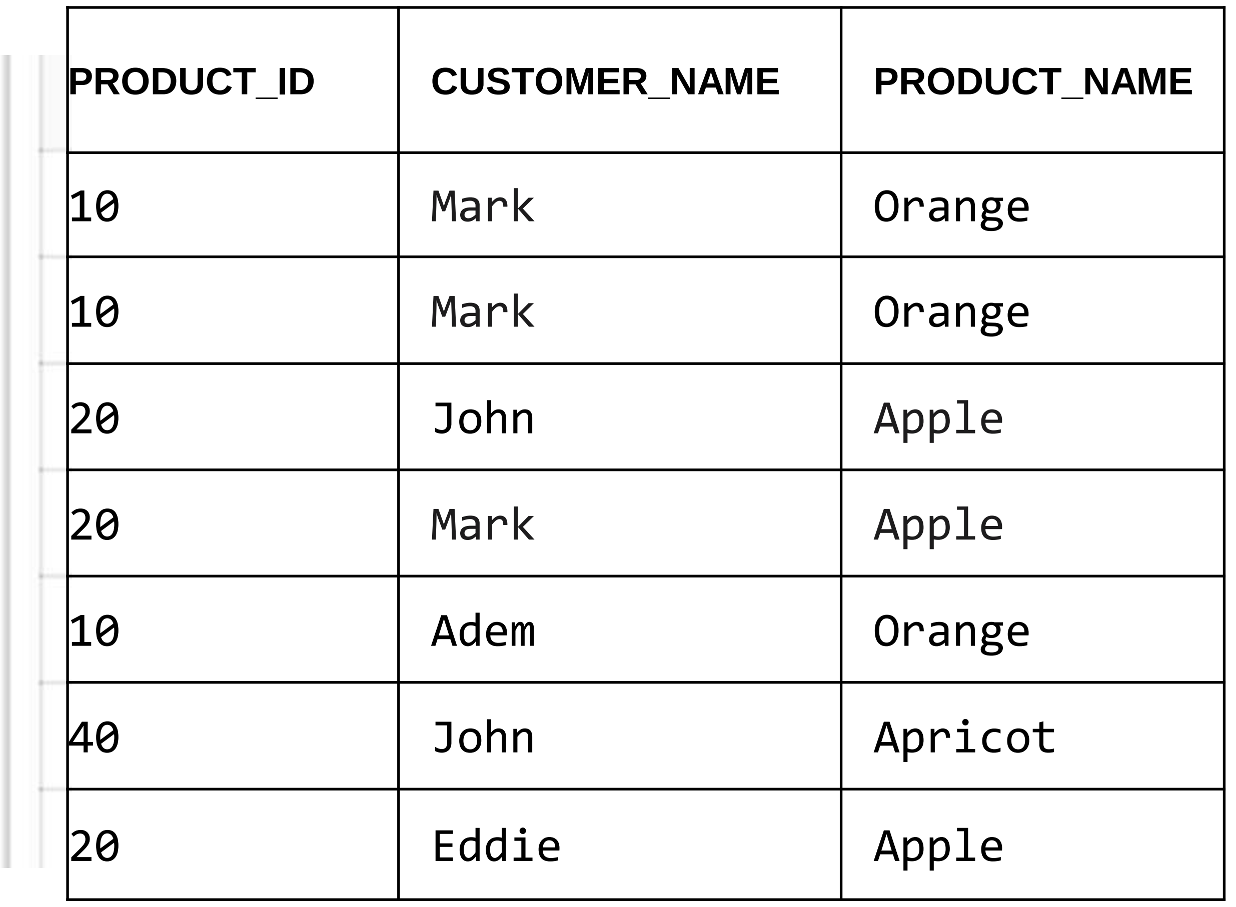
```
INSERT INTO customers_products VALUES (10, 'Mark', 'Orange');
INSERT INTO customers_products VALUES (10, 'Mark', 'Orange');
INSERT INTO customers_products VALUES (20, 'John', 'Apple');
INSERT INTO customers_products VALUES (30, 'Amy', 'Palm');
INSERT INTO customers_products VALUES (20, 'Mark', 'Apple');
INSERT INTO customers_products VALUES (10, 'Adem', 'Orange');
INSERT INTO customers_products VALUES (40, 'John', 'Apricot');
INSERT INTO customers_products VALUES (20, 'Eddie', ‘Apple');
```



PRODUCT_ID	CUSTOMER-NAME	PRODUCT-NAME
10	Mark	Orange
10	Mark	Orange
20	John	Apple
30	Amy	Palm
20	Mark	Apple
10	Adem	Orange
40	John	Apricot
20	Eddie	Apple

```
SELECT *
FROM customers_products
WHERE product_name ='Orange' OR product_name ='Apple' OR product_name ='Apricot';
```

```
SELECT *
FROM customers_products
WHERE product_name IN ('Orange', 'Apple', 'Apricot');
```



PRODUCT_ID	CUSTOMER_NAME	PRODUCT_NAME
10	Mark	Orange
10	Mark	Orange
20	John	Apple
20	Mark	Apple
10	Adem	Orange
40	John	Apricot
20	Eddie	Apple

Review Questions 10 Minutes

1) What is the difference between “DELETE” and “TRUNCATE”

2) What is the difference between “DELETE” and “DROP”

3) What is the difference between “DROP” and “DROP PURGE”

4) Type a query which gives the same result with the following query

```
SELECT *
FROM students
WHERE age>=8 AND age<=17;
```

5) Type a query which gives the same result with the following query

```
SELECT *
FROM students
WHERE age<8 OR age>17;
```

6) Type a query which gives the same result with the following query

```
SELECT *
FROM students
WHERE grade = 6 OR grade = 7 OR grade = 8 OR grade = 9;
```

Answers of Review Questions

1) What is the difference between “DELETE” and “TRUNCATE”

A)TRUNCATE removes all rows from a table. DELETE command is used to remove all or specific rows from a table based on WHERE condition. B)If you use TRUNCATE rollback is not possible. For DELETE rollback is possible.

C)We cannot use WHERE clause with TRUNCATE but we can use WHERE with DELETE.

2) What is the difference between “DELETE” and “DROP”

A) DROP command removes a table from the database while DELETE removes records from a table.

3) What is the difference between “DROP” and “DROP PURGE”

A) The DROP will drop the table and place it into the recycle bin.
DROP with PURGE will drop the table and flush it out from the recycle bin also.

4)

```
SELECT * FROM students
WHERE age>=8 AND age<=17;
```

→

```
SELECT *
FROM students
WHERE age BETWEEN 8 AND 17;
```

5)

```
SELECT *
FROM students
WHERE age<8 OR age>17;
```

→

```
SELECT *
FROM students
WHERE age NOT
BETWEEN 8 AND 17;
```

6)

```
SELECT *
FROM students
WHERE grade = 6 OR grade = 7 OR grade = 8 OR grade = 9;
```

```
SELECT *
FROM students
WHERE grade IN (6, 7, 8, 9);
```

“BETWEEN” Condition

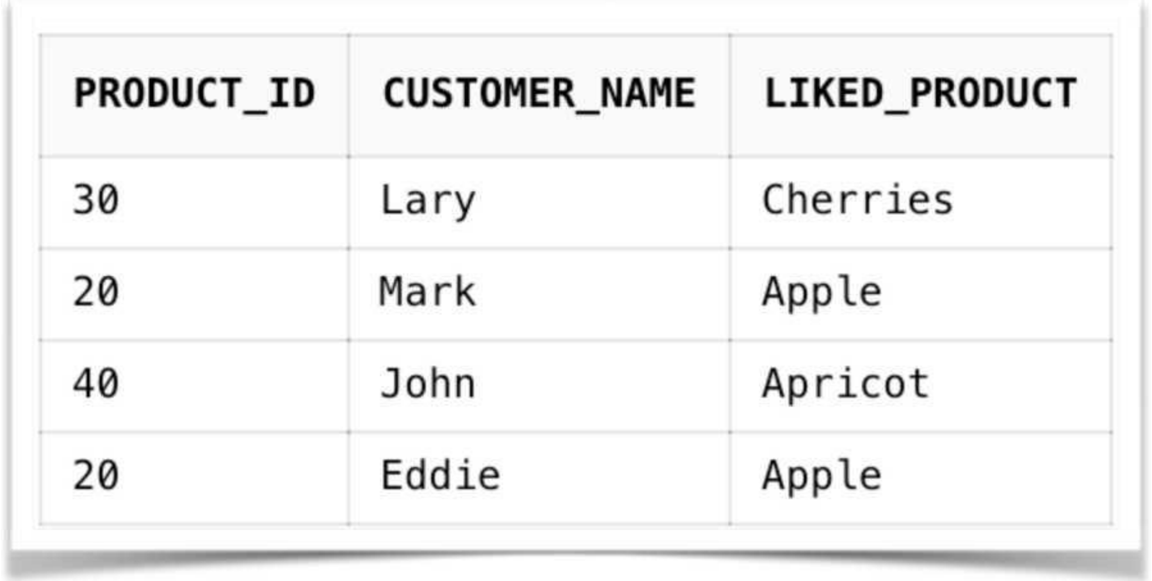
```
CREATE TABLE customers_likes
(
  product_id number(10),
 customer_name varchar2(50),
 liked_product varchar2(50)
);
```

```
INSERT INTO customers_likes VALUES (10, 'Mark', 'Orange');
INSERT INTO customers_likes VALUES (50, 'Mark', 'Pineapple');
INSERT INTO customers_likes VALUES (60, 'John', 'Avocado');
INSERT INTO customers_likes VALUES (30, 'Lary', 'Cherries');
INSERT INTO customers_likes VALUES (20, 'Mark', 'Apple');
INSERT INTO customers_likes VALUES (10, 'Adem', 'Orange');
INSERT INTO customers_likes VALUES (40, 'John', 'Apricot');
INSERT INTO customers_likes VALUES (20, 'Eddie', 'Apple');
```

PRODUCT_ID	CUSTOMER-NAME	LIKED_PRODUCT
10	Mark	Orange
50	Mark	Pineapple
60	John	Avocado
30	Lary	Cherries
20	Mark	Apple
10	Adem	Orange
40	John	Apricot
20	Eddie	Apple

```
SELECT *
FROM customers_likes
WHERE product_id BETWEEN 20 AND 40;
```

```
SELECT *
FROM customers_likes
WHERE product_id >= 20 AND product_id <     40;
```



PRODUCT_ID	CUSTOMER_NAME	LIKED_PRODUCT
30	Lary	Cherries
20	Mark	Apple
40	John	Apricot
20	Eddie	Apple

Note: 20 and 40 are inclusive for BETWEEN condition

“EXISTS” Condition

EXISTS condition is used in combination with a subquery and is considered "to be met" if the subquery returns at least one row. It can be used in a SELECT, INSERT, UPDATE, or DELETE statement.

```
CREATE TABLE customers_products
(
  product_id number(10),
 customer_name varchar2(50),
 product_name varchar2(50)
);
```

```
INSERT INTO customers_products VALUES (10, 'Mark', 'Orange');
INSERT INTO customers_products VALUES (10, 'Mark', 'Orange');
INSERT INTO customers_products VALUES (20, 'John', 'Apple');
INSERT INTO customers_products VALUES (30, 'Amy', 'Palm');
INSERT INTO customers_products VALUES (20, 'Mark', 'Apple');
INSERT INTO customers_products VALUES (10, 'Adem', 'Orange');
INSERT INTO customers_products VALUES (40, 'John', 'Apricot');
INSERT INTO customers_products VALUES (20, 'Eddie', ‘Apple');
```

PRODUCT_ID	CUSTOMER-NAME	PRODUCT-NAME
10	Mark	Orange
10	Mark	Orange
20	John	Apple
30	Amy	Palm
20	Mark	Apple
10	Adem	Orange
40	John	Apricot
20	Eddie	Apple

```
CREATE TABLE customers_likes
(
  product_id number(10),
 customer_name varchar2(50),
 liked_product varchar2(50)
);
```

```
INSERT INTO customers_likes VALUES (10, 'Mark', 'Orange');
INSERT INTO customers_likes VALUES (50, 'Mark', 'Pineapple');
INSERT INTO customers_likes VALUES (60, 'John', 'Avocado');
INSERT INTO customers_likes VALUES (30, 'Lary', 'Cherries');
INSERT INTO customers_likes VALUES (20, 'Mark', 'Apple');
INSERT INTO customers_likes VALUES (10, 'Adem', 'Orange');
INSERT INTO customers_likes VALUES (40, 'John', 'Apricot');
INSERT INTO customers_likes VALUES (20, 'Eddie', 'Apple');
```

PRODUCT_ID	CUSTOMER-NAME	LIKED_PRODUCT
10	Mark	Orange
50	Mark	Pineapple
60	John	Avocado
30	Lary	Cherries
20	Mark	Apple
10	Adem	Orange
40	John	Apricot
20	Eddie	Apple

```
SELECT customer_name
FROM customers_products
WHERE EXISTS (SELECT product_id FROM customers_likes WHERE customers_products.product_id = customers_likes.product_id);
```

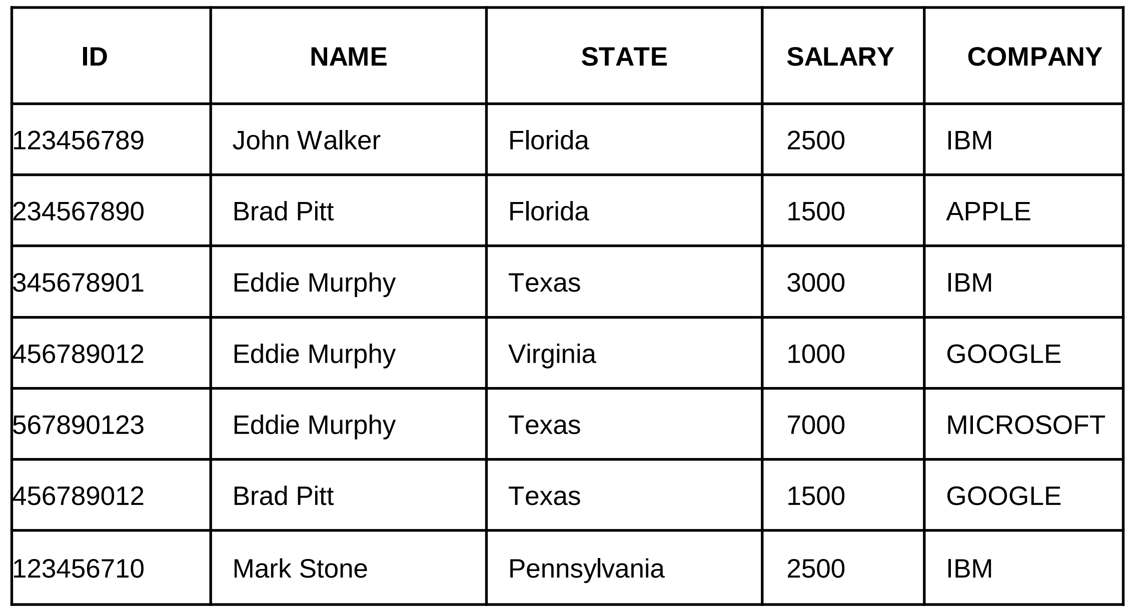
CUSTOMER-NAME
Mark
Mark
Adem
Amy
John
Mark
Eddie
John

“SUBQUERIES”

SUBQUERY is a query within a query

```
CREATE TABLE employees
(
  id number(9), name
  varchar2(50), state
  varchar2(50), salary
  number(20), company
  varchar2(20)
);
```

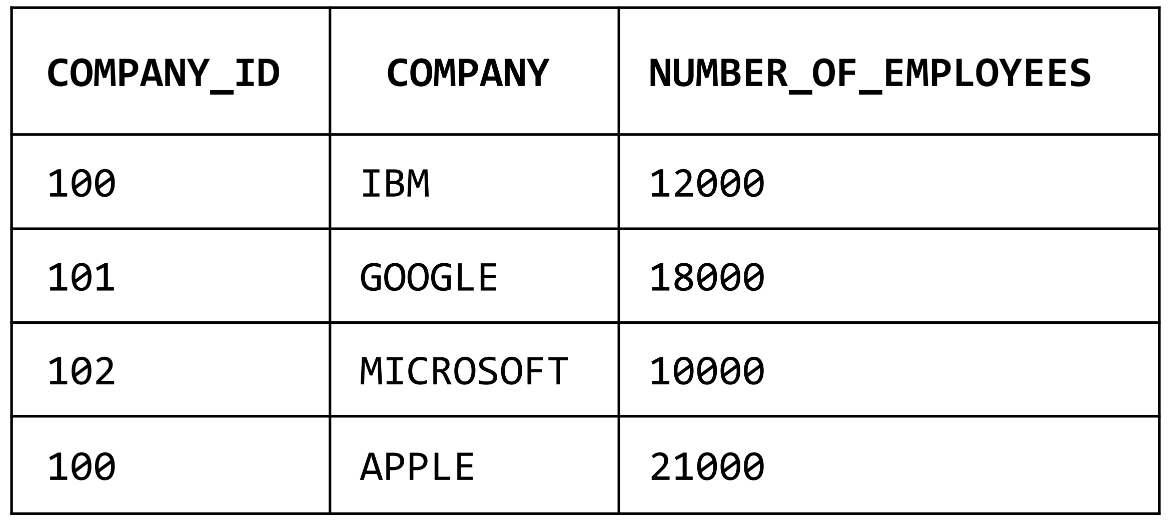
```
INSERT INTO employees VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');
INSERT INTO employees VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');
INSERT INTO employees VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');
INSERT INTO employees VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');
INSERT INTO employees VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');
INSERT INTO employees VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');
INSERT INTO employees VALUES(123456710, 'Mark Stone', 'Pennsylvania', 2500, 'IBM');
```



ID	NAME	STATE	SALARY	COMPANY
123456789	John Walker	Florida	2500	IBM
234567890	Brad Pitt	Florida	1500	APPLE
345678901	Eddie Murphy	Texas	3000	IBM
456789012	Eddie Murphy	Virginia	1000	GOOGLE
567890123	Eddie Murphy	Texas	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	GOOGLE
123456710	Mark Stone	Pennsylvania	2500	IBM

```
CREATE TABLE companies
(
  company_id number(9), company
  varchar2(20),
  number_of_employees number(20)
);
```

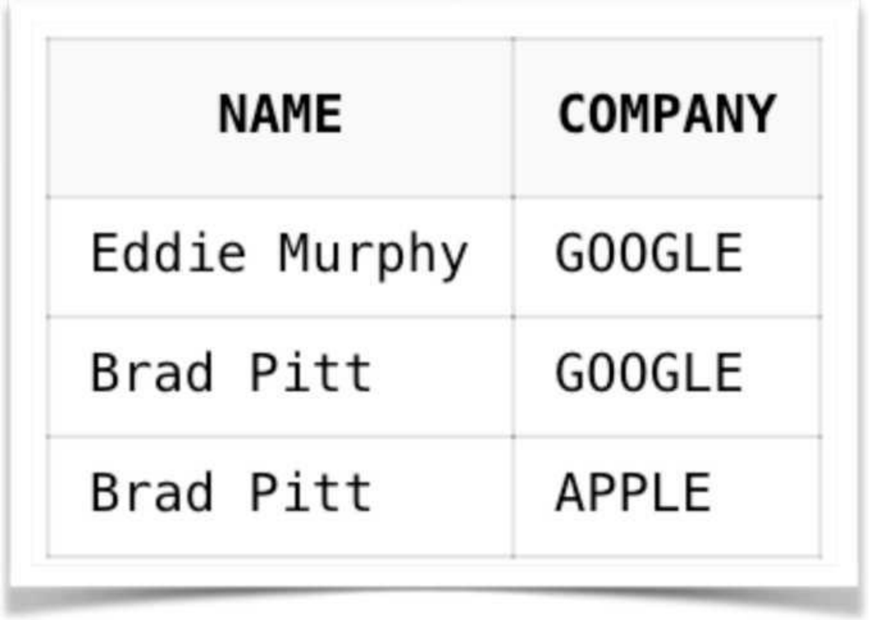
```
INSERT INTO companies VALUES(100, 'IBM', 12000);
INSERT INTO companies VALUES(101, 'GOOGLE', 18000);
INSERT INTO companies VALUES(102, 'MICROSOFT', 10000);
INSERT INTO companies VALUES(100, 'APPLE', 21000);
```



COMPANY_ID	COMPANY	NUMBER_OF_EMPLOYEES
100	IBM	12000
101	GOOGLE	18000
102	MICROSOFT	10000
100	APPLE	21000

Example: Find the employee and company names whose company has more than 15000 employees

```
SELECT name, company
FROM employees
WHERE company IN (SELECT company FROM companies
                  WHERE number_of_employees > 15000);
```



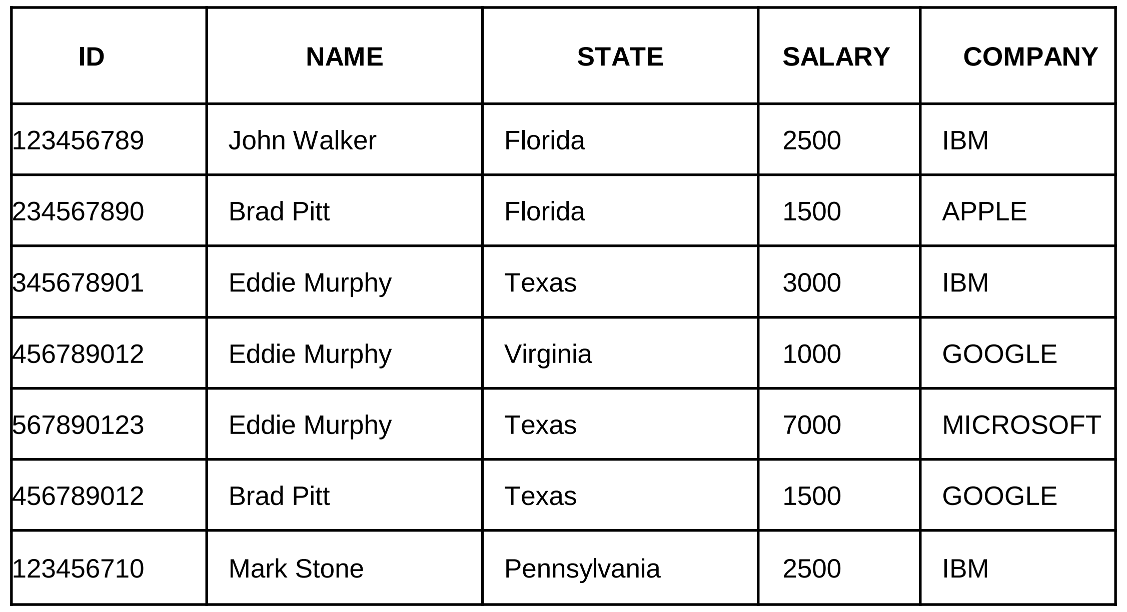
NAME	COMPANY
Eddie Murphy	GOOGLE
Brad Pitt	GOOGLE
Brad Pitt	APPLE

2) SUBQUERY in the SELECT clause

A SUBQUERY in the select clause must return a single value.
Therefore, an aggregate function such as SUM, COUNT, MIN, or MAX is commonly used in the subquery.

```
CREATE TABLE employees
(
  id number(9), name
  varchar2(50), state
  varchar2(50), salary
  number(20), company
  varchar2(20)
);
```

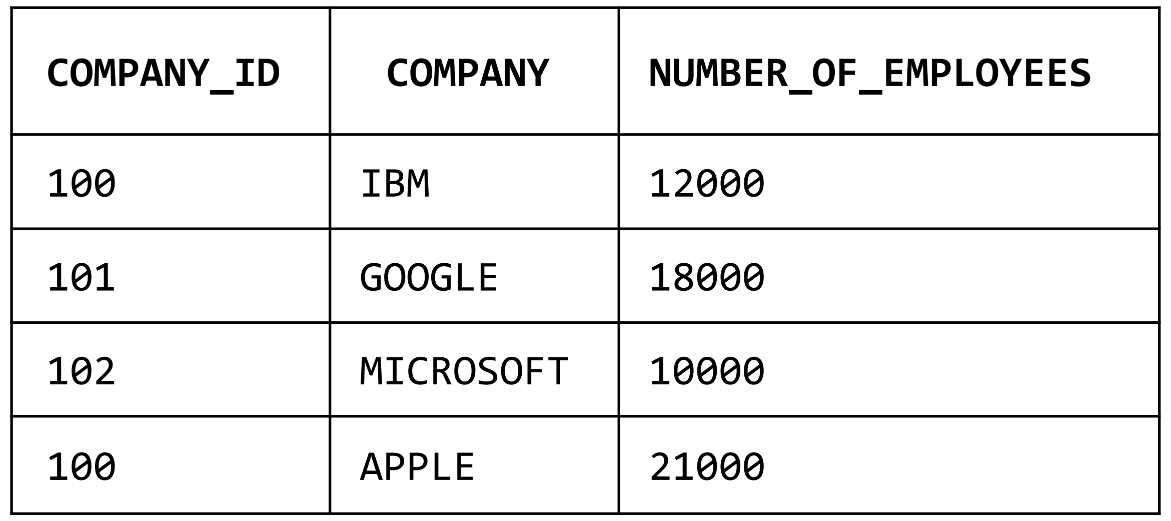
```
INSERT INTO employees VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');
INSERT INTO employees VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');
INSERT INTO employees VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');
INSERT INTO employees VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');
INSERT INTO employees VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');
INSERT INTO employees VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');
INSERT INTO employees VALUES(123456710, 'Mark Stone', 'Pennsylvania', 2500, 'IBM');
```



ID	NAME	STATE	SALARY	COMPANY
123456789	John Walker	Florida	2500	IBM
234567890	Brad Pitt	Florida	1500	APPLE
345678901	Eddie Murphy	Texas	3000	IBM
456789012	Eddie Murphy	Virginia	1000	GOOGLE
567890123	Eddie Murphy	Texas	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	GOOGLE
123456710	Mark Stone	Pennsylvania	2500	IBM

```
CREATE TABLE companies
(
  company_id number(9), company
  varchar2(20),
  number_of_employees number(20)
);
```

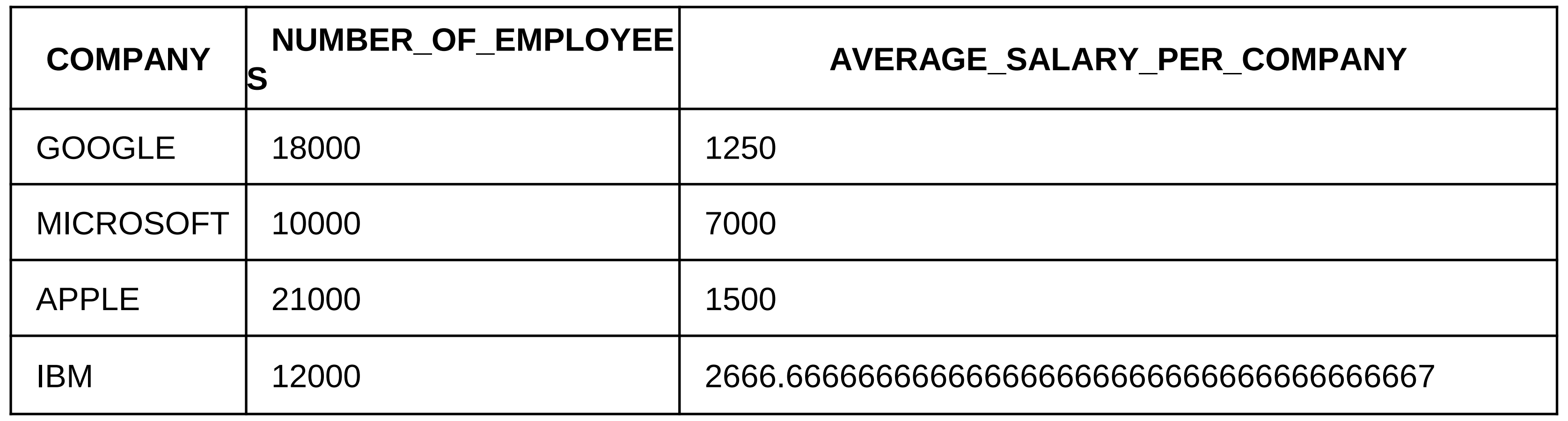
```
INSERT INTO companies VALUES(100, 'IBM', 12000);
INSERT INTO companies VALUES(101, 'GOOGLE', 18000);
INSERT INTO companies VALUES(102, 'MICROSOFT', 10000);
INSERT INTO companies VALUES(100, 'APPLE', 21000);
```



COMPANY_ID	COMPANY	NUMBER_OF_EMPLOYEES
100	IBM	12000
101	GOOGLE	18000
102	MICROSOFT	10000
100	APPLE	21000

Example: Find the number of employees and average salary for every company

```
SELECT company, number_of_employees,

  (SELECT AVG(salary)
  FROM employees
  WHERE companies.company = employees.company) Average_Salary_Per_Company

FROM companies;
```



COMPANY	NUMBER_OF_EMPLOYEES	AVERAGE_SALARY_PER_COMPANY
GOOGLE	18000	1250
MICROSOFT	10000	7000
APPLE	21000	1500
IBM	12000	2666.66666666666666666666666666666666667

Example: Find the name of the companies, company ids, and the number of states for every company

```
SELECT company, companyjd, (SELECT COUNT(state)
                  FROM employees
                  WHERE companies.company = employees.company )
                  number_of_states
FROM companies;
```

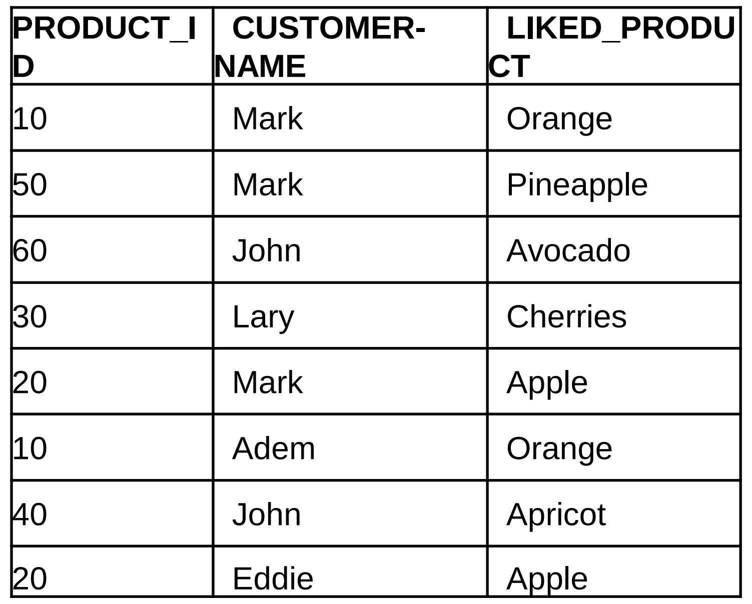
Example: Find the name of the companies, company ids, maximum and minimum salaries per company.

```
SELECT company, company_id, (SELECT MAX(salary)
                    FROM employees
                    WHERE companies.company = employees.company ) max_salary,

                    (SELECT MIN(salary)
                    FROM employees
                    WHERE companies.company = employees.company ) min_salary
FROM companies;
```

“NOT BETWEEN” Condition

```
CREATE TABLE customers_likes
(
  product_id number(10),
 customer_name varchar2(50),
 liked_product varchar2(50)
);
```

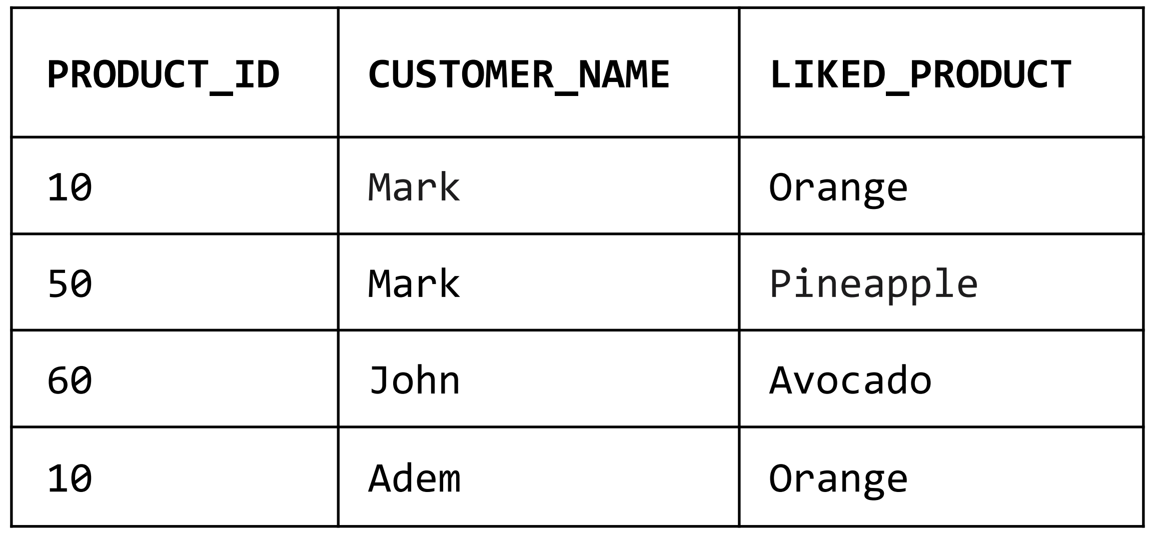
```
INSERT INTO customersjikes VALUES (10, 'Mark', 'Orange');
INSERT INTO customersjikes VALUES (50, 'Mark', 'Pineapple');
INSERT INTO customersjikes VALUES (60, 'John', 'Avocado');
INSERT INTO customersjikes VALUES (30, 'Lary', 'Cherries');
INSERT INTO customersjikes VALUES (20, 'Mark', 'Apple');
INSERT INTO customersjikes VALUES (10, 'Adem', 'Orange');
INSERT INTO customersjikes VALUES (40, 'John', 'Apricot');
INSERT INTO customersjikes VALUES (20, 'Eddie', 'Apple');
```



PRODUCT_ID	CUSTOMER-NAME	LIKED_PRODUCT
10	Mark	Orange
50	Mark	Pineapple
60	John	Avocado
30	Lary	Cherries
20	Mark	Apple
10	Adem	Orange
40	John	Apricot
20	Eddie	Apple

```
    SELECT *
    FROM customersjikes
    WHERE product_id NOT BETWEEN 20 AND
40;
```

```
    SELECT *
    FROM customersjikes
    WHERE product_id < 20 OR product_id >
40;
```



PRODUCT_ID	CUSTOMER_NAME	LIKED_PRODUCT
10	Mark	Orange
50	Mark	Pineapple
60	John	Avocado
10	Adem	Orange

Note: 20 and 40 are exclusive for NOT BETWEEN condition

LIKE Condition

LIKE condition allows wildcards to be used in the WHERE Clause of a SELECT, INSERT, UPDATE, or DELETE statement.
This allows you to perform pattern matching.

Wildcard Characters

1) % => Represents zero or more characters

WHERE customer_name LIKE 'J%' ==> Finds any values that starts with “J”

WHERE customer_name LIKE '%e' ==> Finds any values that ends with “e”

WHERE customer_name LIKE '%an%' ==> Finds any values that have “an” in any position

```
CREATE TABLE customers
(
customer_id number(10) UNIQUE,
customer_name varchar2(50) NOT NULL,
income number(6)
);
```

```
INSERT INTO customers (customer_id, customer_name, income)
VALUES (1001, 'John', 62000);

INSERT INTO customers (customer_id, customer_name, income)
VALUES (1002, 'Jane', 57500);

INSERT INTO customers (customer_id, customer_name, income)
VALUES (1003, 'Brad', 71000);

INSERT INTO customers (customer_id, customer_name, income)
VALUES (1004, ‘Manse', 42000);
```

CUSTOMER_ID	CUSTOMER_NAME	INCOME
1001	John	62000
1002	Jane	57500
1003	Brad	71000
1004	Manse	42000

2) _ => Represents just one character

WHERE CustomerName LIKE '_ohn' ==> Finds all customer’s name starting with any character, followed by "ohn"

WHERE CustomerName LIKE ‘ a e' ==> Finds all sized 4 customer’s name whose 2nd character is “a”, and 4th character is “e”

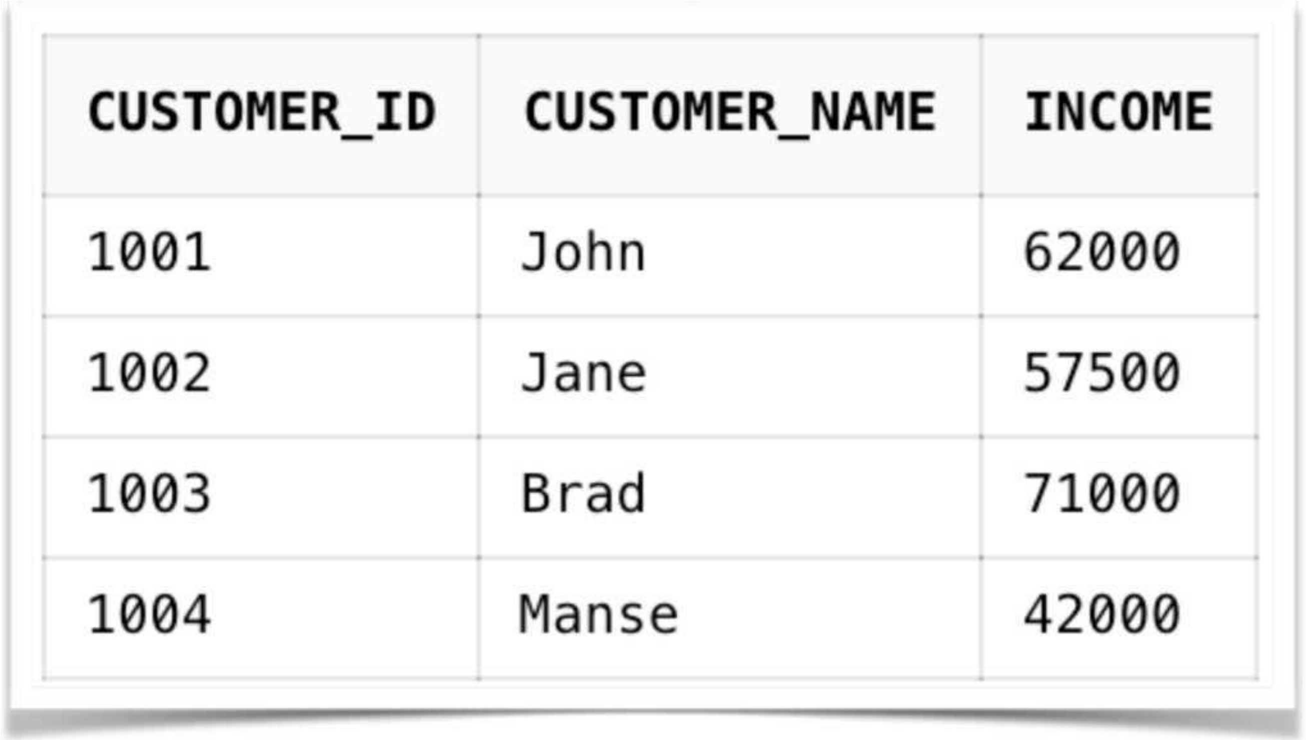
WHERE CustomerName LIKE '_r%' ==> Finds any values that have “r” in the second position

WHERE CustomerName LIKE ‘M_%_%_%’ ==> Finds any values that starts with "M" and are at least 4 characters in length

WHERE CustomerName LIKE 'B%d' ==> Finds any values that starts with “B” and ends with “d”

```
CREATE TABLE customers (
customer_id number(10) UNIQUE,
customer_name varchar2(50) NOT NULL,
income number(6) );
```

```
INSERT INTO customers (customer_id, customer_name, income)
VALUES (1001, 'John', 62000);

INSERT INTO customers (customer_id, customer_name, income)
VALUES (1002, 'Jane', 57500);

INSERT INTO customers (customer_id, customer_name, income)
VALUES (1003, 'Brad', 71000);

INSERT INTO customers (customer_id, customer_name, income)
VALUES (1004, ‘Manse', 42000);
```



CUSTOMER_ID	CUSTOMER_NAME	INCOME
1001	John	62000
1002	Jane	57500
1003	Brad	71000
1004	Manse	42000

3) REGEXP_LIKE Condition

WHERE REGEXP_LIKE(word, 'h[oa]t') ==> Finds “hot” and “hat”, but not “hit”

WHERE REGEXP_LIKE(word, ‘h(o|a)t’) ==> Finds “hot” and “hat”, but not “hit”

WHERE REGEXP_LIKE(word, 'h[a-c]t') ==> Finds “hat” and “hbt” and “hct”

WHERE REGEXP_LIKE(word, ‘h(a|b|c)t’) ==> Finds “hat” and “hbt” and “hct”

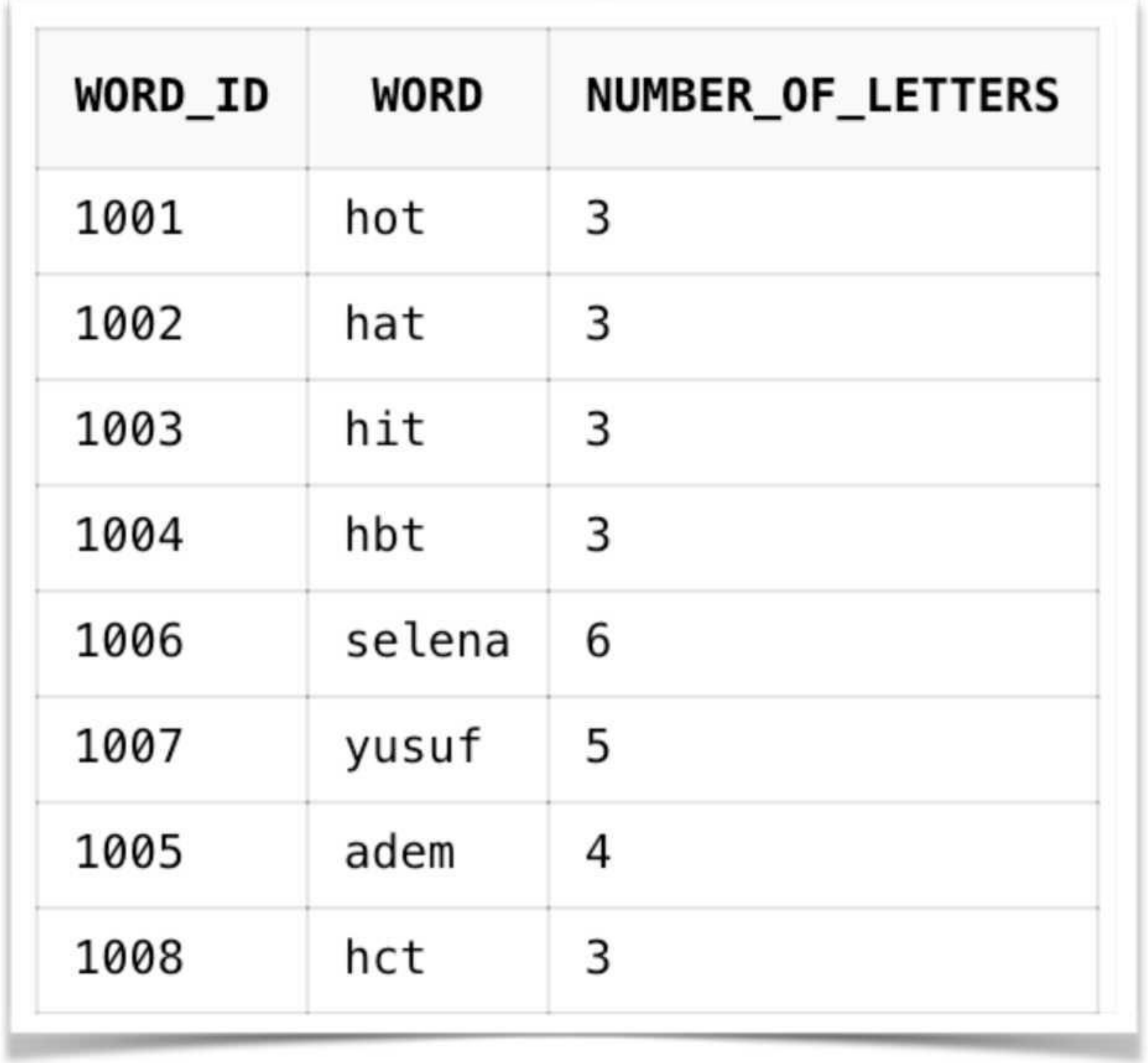
WHERE REGEXP_LIKE(word, '[au](*)') ==> Finds all contains “a” and “u”
“hat” and “selena” and “yusuf” and “adem”

WHERE REGEXP_LIKE(word, 'A[asy](*)') ==> Finds all start with “a” or “s” or “y”
“adem” and “selena” and “yusuf”

WHERE REGEXP_LIKE(word, ‘(*) f $') ==> Finds all end with “f” ==> “yusuf”

```
CREATE TABLE words
(
  word_id   number(10)   UNIQUE,
  word  varchar2(50)  NOT  NULL,
  number_of_letters number(6)
);
```

```
   INSERT INTO words VALUES (1001, 'hot', 3);
INSERT INTO words VALUES (1002, 'hat', 3);
INSERT INTO words VALUES (1003, 'hit', 3);
INSERT INTO words VALUES (1004, 'hbt', 3);
INSERT INTO words VALUES (1008, 'hct', 3);
INSERT INTO words VALUES (1005, 'adem', 4);
INSERT INTO words VALUES (1006, 'selena', 6);
INSERT INTO words VALUES (1007, 'yusuf', 5);
```



WORD_ID	WORD	NUMBER_OF_LETTERS
1001	hot	3
1002	hat	3
1003	hit	3
1004	hbt	3
1006	selena	6
1007	yusuf	5
1005	adem	4
1008	hct	3

NOT LIKE Condition

WHERE word NOT LIKE 'h%' ==> Finds all words which do NOT start with ‘h'.

WHERE word NOT LIKE '%t' ==> Finds all words which do NOT end with ’t'.

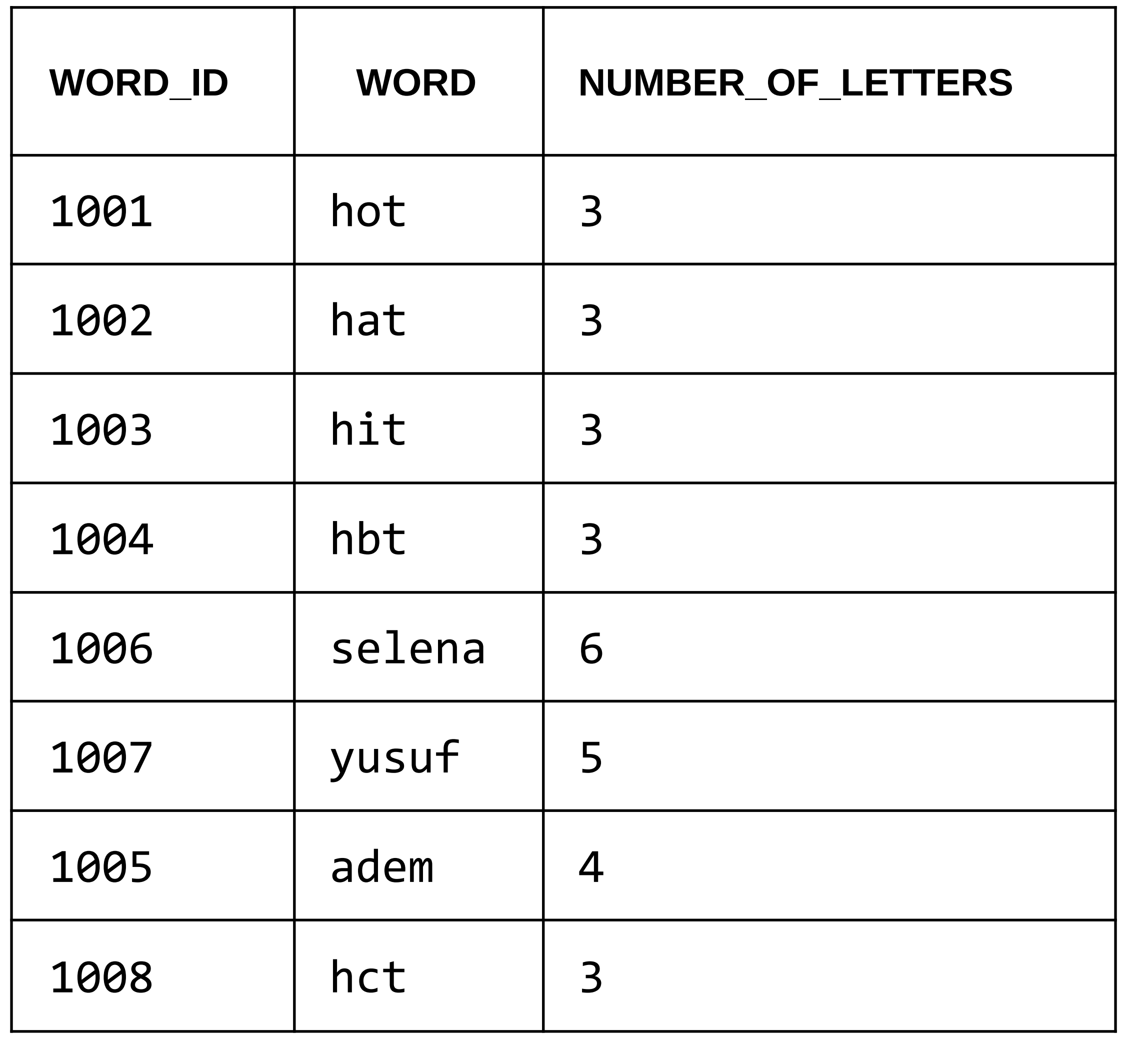
WHERE word NOT LIKE '%a%' ==> Finds all words which do NOT contain “a" in any position

WHERE word NOT LIKE '_us%' ==> Finds all customer’s name starting with any character, NOT followed by “us"

WHERE NOT REGEXP_LIKE(word, ‘[_ead](*)’); ==> Find all words starting with any character, NOT following by 'e' or 'a' or 'd'

```
CREATE TABLE words
(
  word_id   number(10)  UNIQUE,
  word  varchar2(50)  NOT  NULL,
  number_of_letters number(6)
);
```

```
  INSERT INTO words VALUES (1001, 'hot', 3);
INSERT INTO words VALUES (1002, 'hat', 3);
INSERT INTO words VALUES (1003, 'hit', 3);
INSERT INTO words VALUES (1004, 'hbt', 3);
INSERT INTO words VALUES (1004, 'hct', 3);
INSERT INTO words VALUES (1005, 'adem', 4);
INSERT INTO words VALUES (1006, 'selena', 6);
INSERT INTO words VALUES (1007, 'yusuf', 5);
```



WORD_ID	WORD	NUMBER_OF_LETTERS
1001	hot	3
1002	hat	3
1003	hit	3
1004	hbt	3
1006	selena	6
1007	yusuf	5
1005	adem	4
1008	hct	3

"ORDER BY" Clause

The ORDER BY clause is used to sort the records in result set.

The ORDER BY clause can only be used in SELECT statements.

1)

	ID	NAME	STATE	GPA
	123456789	John Walker	Texas	2.8
	234567890	Eddie Murphy	Florida	3.2
	345678901	Adam Eve	New York	3.5
	456789012	Alex Tien	New York	3.8
	567890123	Chris Matala	Virginia	4

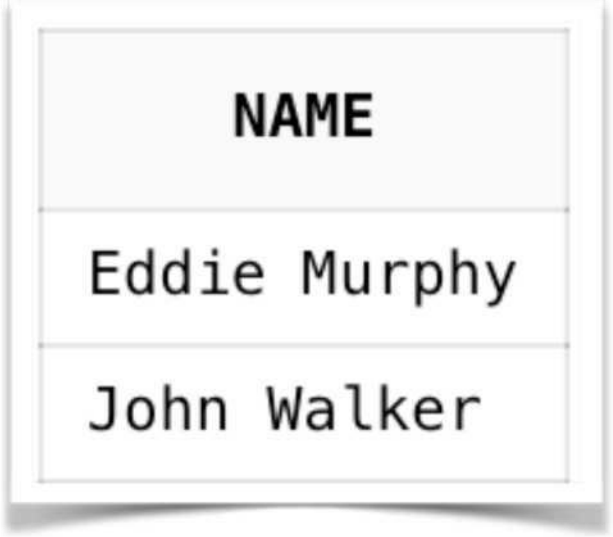
```
SELECT * FROM
students ORDER
BY name;
```

ID	NAME	STATE	GPA
345678901	Adam Eve	New York	3.5
456789012	Alex Tien	New York	3.8
567890123	Chris Matala	Virginia	4
234567890	Eddie Murphy	Florida	3.2
123456789	John Walker	Texas	2.8

2)

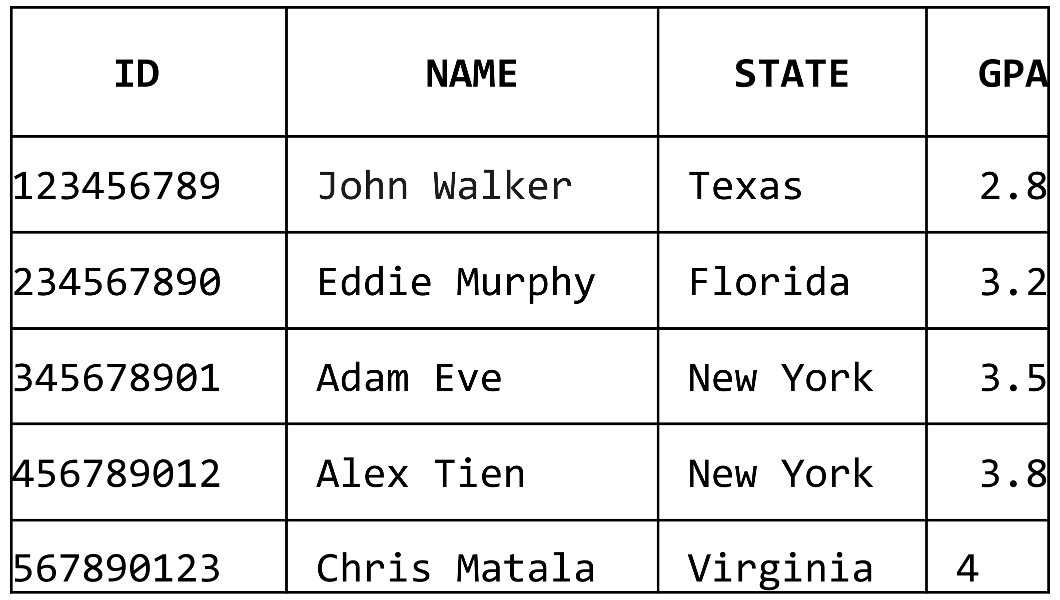
ID	NAME	STATE	GPA
123456789	John Walker	Texas	2.8
234567890	Eddie Murphy	Florida	3.2
345678901	Adam Eve	New York	3.5
456789012	Alex Tien	New York	3.8
567890123	Chris Matala	Virginia	4

```
SELECT name FROM students
WHERE gpa = 2.8 OR state ='Florida'
ORDER BY name;
```



NAME
Eddie Murphy
John Walker

3)

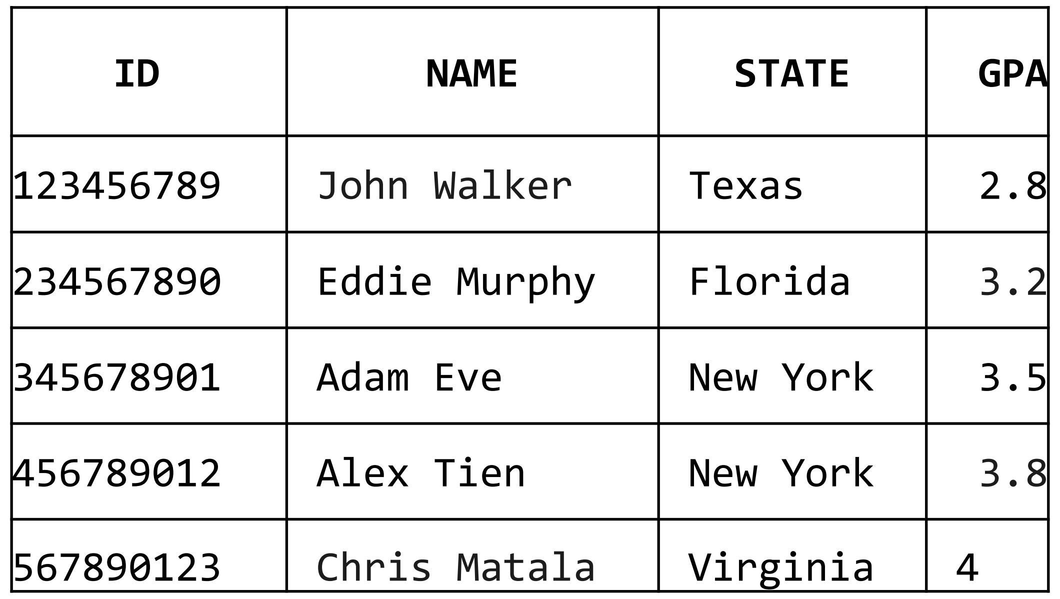


ID	NAME	STATE	GPA
123456789	John Walker	Texas	2.8
234567890	Eddie Murphy	Florida	3.2
345678901	Adam Eve	New York	3.5
456789012	Alex Tien	New York	3.8
567890123	Chris Matala	Virginia	4

```
SELECT * FROM
students ORDER BY
name DESC;
```

ID	NAME	STATE	GPA
123456789	John Walker	Texas	2.8
234567890	Eddie Murphy	Florida	3.2
567890123	Chris Matala	Virginia	4
456789012	Alex Tien	New York	3.8
345678901	Adam Eve	New York	3.5

4)



ID	NAME	STATE	GPA
123456789	John Walker	Texas	2.8
234567890	Eddie Murphy	Florida	3.2
345678901	Adam Eve	New York	3.5
456789012	Alex Tien	New York	3.8
567890123	Chris Matala	Virginia	4

```
SELECT *
FROM students
ORDER BY 3 DESC;
```

ID	NAME	STATE	GPA
567890123	Chris Matala	Virginia	4
123456789	John Walker	Texas	2.8
345678901	Adam Eve	New York	3.5
456789012	Alex Tien	New York	3.8
234567890	Eddie Murphy	Florida	3.2

Number of columns

5)

ID	NAME	STATE	GPA
123456789	John Walker	Texas	2.8
234567890	Eddie Murphy	Florida	3.2
345678901	Adam Eve	New York	3.5
456789012	Zeyna Rose	New York	3.8
567890123	Chris Matala	Virginia	4
456789012	Brad Pitt	New York	3.8

```
      SELECT * ► FROM
students
      ORDER BY 3 DESC, 2 ASC;
```

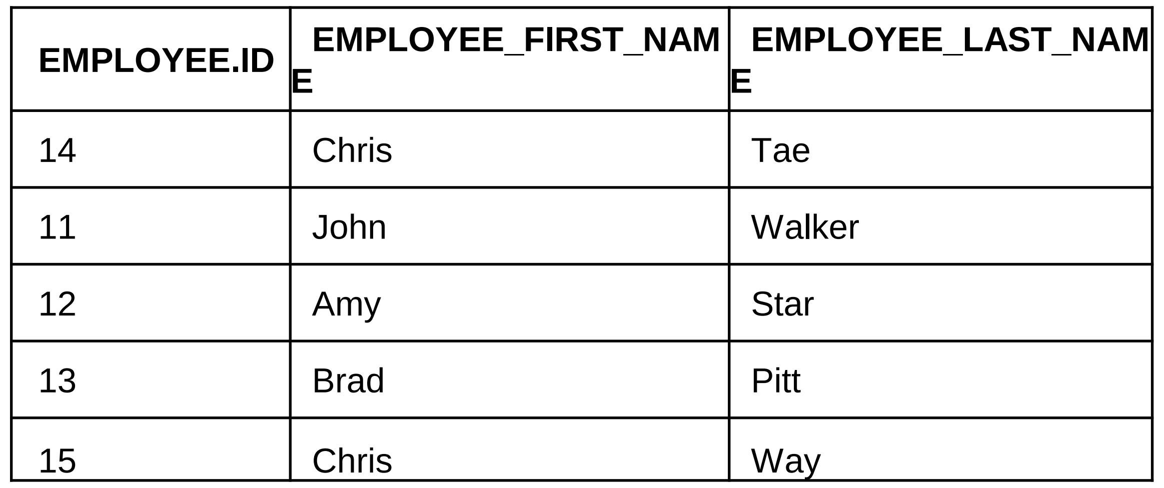
ID	NAME	STATE	GPA
567890123	Chris Matala	Virginia	4
123456789	John Walker	Texas	2.8
345678901	Adam Eve	New York	3.5
456789012	Brad Pitt	New York	3.8
456789012	Zeyna Rose	New York	3.8
234567890	Eddie Murphy	Florida	3.2

ORDER BY will return all records sorted by the *3rd* field in descending order, with a secondary sort by *2nd field* in ascending order.

“ALIASES”

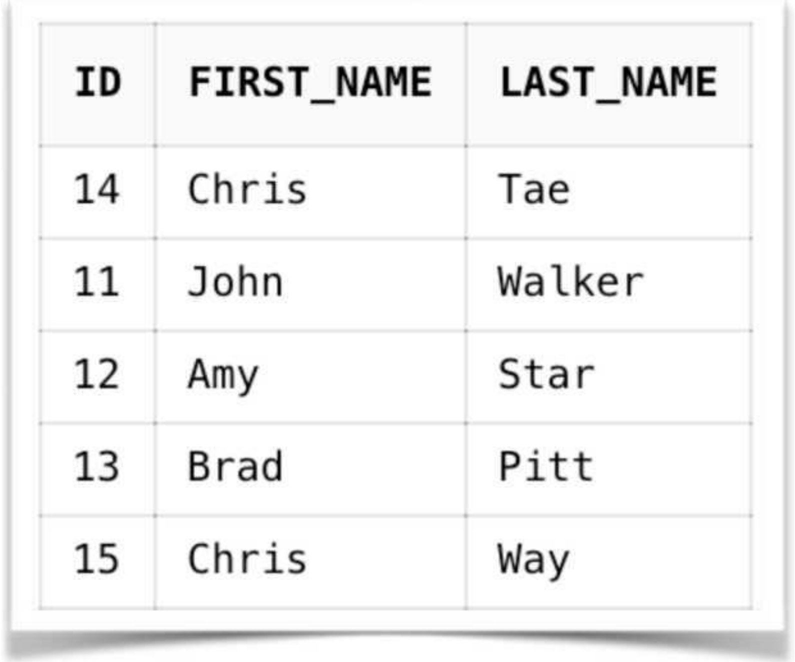
```
CREATE TABLE employees
(
  employee_id number(9),
  employee_first_name varchar2(20)
  employee_last_name varchar2(20)
);
```

```
INSERT INTO employees VALUES(14, 'Chris', ‘Tae’);
INSERT INTO employees VALUES(11, 'John', 'Walker');
► INSERT INTO employees VALUES(12, 'Amy', 'Star');
INSERT INTO employees VALUES(13, 'Brad', 'Pitt');
INSERT INTO employees VALUES(15, 'Chris', ‘Way’);
```



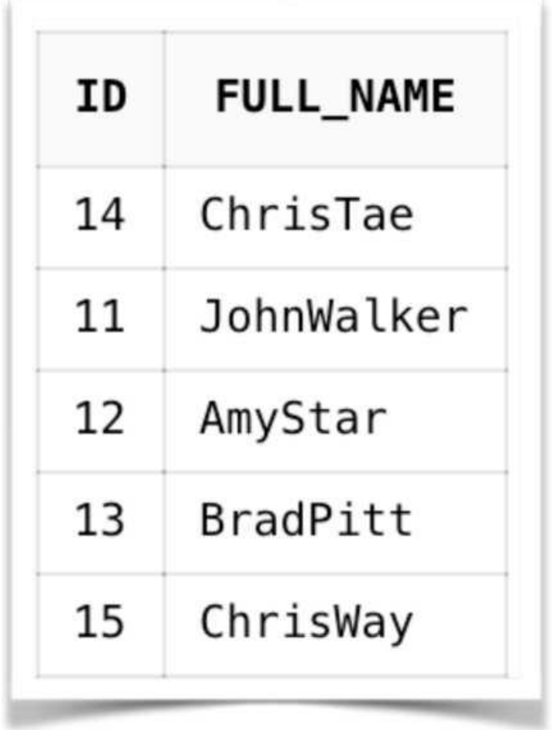
EMPLOYEE.ID	EMPLOYEE_FIRST_NAM E	EMPLOYEE_LAST_NAM E
14	Chris	Tae
11	John	Walker
12	Amy	Star
13	Brad	Pitt
15	Chris	Way

1)
```
SELECT employee_id AS id, employee_first_name AS first_name, employee_last_name AS last_name
FROM employees;
```



ID	FIRST_NAME	LAST_NAME
14	Chris	Tae
11	John	Walker
12	Amy	Star
13	Brad	Pitt
15	Chris	Way

2)
```
SELECT employee_id AS id, employee_first_name | | employee_last_name AS full_name
FROM employees;
```



ID	FULL_NAME
14	ChrisTae
11	JohnWalker
12	AmyStar
13	BradPitt
15	ChrisWay

```
CREATE TABLE employees
(
  employee_id number(9),
  employee_first_name varchar2(20)
  employee_last_name varchar2(20)
);
```

```
 INSERT INTO employees VALUES(14, 'Chris', ‘Tae’);
 INSERT INTO employees VALUES(11, 'John', 'Walker');
► INSERT INTO employees VALUES(12, 'Amy', 'Star');
 INSERT INTO employees VALUES(13, 'Brad', 'Pitt');
 INSERT INTO employees VALUES(15, 'Chris', ‘Way’);
```

	EMPLOYEE_ID	EMPLOYEE_FIRST_NAME	EMPLOYEE_LAST_NAME
	14	Chris	Tae
	11	John	Walker
	12	Amy	Star
	13	Brad	Pitt
	15	Chris	Way

```
CREATE TABLE addresses
( employee_id number(9),
street varchar2(20), city
varchar2(20), state char(2),
zipcode char(5)
);
```

```
INSERT INTO addresses VALUES(11, '32nd Star 1234', 'Miami', 'FL', '33018');
INSERT INTO addresses VALUES(12, '23rd Rain 567', 'Jacksonville', 'FL', '32256');
INSERT INTO addresses VALUES(13, '5th Snow 765', 'Hialeah', 'VA', '20121');
INSERT INTO addresses VALUES(14, '3rd Man 12', 'Weston', 'MI', '12345');
INSERT INTO addresses VALUES(15, '11th Chris 12', 'St. Johns', 'FL', ‘32259');
```

EMPLOYEE_ID	STREET	CITY	STATE	ZIPCODE
11	32nd Star 1234	Miami	FL	33018
12	23rd Rain 567	Jacksonville	FL	32256
13	5th Snow 765	Hialeah	VA	20121
14	3rd Man 12	Weston	MI	12345
15	11th Chris 12	St. Johns	FL	32259

3)
```
SELECT e.employee_first_name, e.employee_last_name, a.city
FROM employees e, addresses a
WHERE e.employee_id = a.employee_id;
```

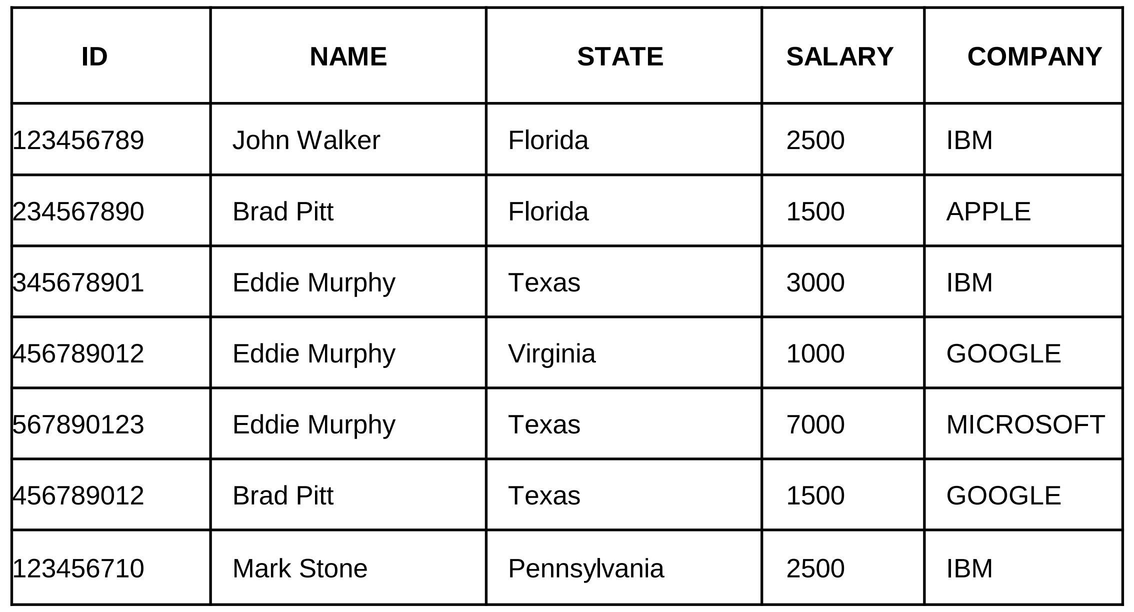
EMPLOYEE_FIRST_NAME	EMPLOYEE_LAST_NAME	CITY
John	Walker	Miami
Amy	Star	Jacksonville
Brad	Pitt	Hialeah
Chris	Tae	Weston
Chris	Way	St. Johns

“GROUP BY” Clause

GROUP BY clause is used in a SELECT statement to collect data across multiple records and group the results by one or more columns.

```
CREATE TABLE employees
(
 id number(9), name
 varchar2(50), state
 varchar2(50), salary
 number(20), company
 varchar2(20)
);
```

```
INSERT INTO employees VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');
INSERT INTO employees VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');
INSERT INTO employees VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');
   ► INSERT INTO employees VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');
INSERT INTO employees VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');
INSERT INTO employees VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');
INSERT INTO employees VALUES(123456710, 'Mark Stone', 'Pennsylvania', 2500, 'IBM');
```



ID	NAME	STATE	SALARY	COMPANY
123456789	John Walker	Florida	2500	IBM
234567890	Brad Pitt	Florida	1500	APPLE
345678901	Eddie Murphy	Texas	3000	IBM
456789012	Eddie Murphy	Virginia	1000	GOOGLE
567890123	Eddie Murphy	Texas	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	GOOGLE
123456710	Mark Stone	Pennsylvania	2500	IBM

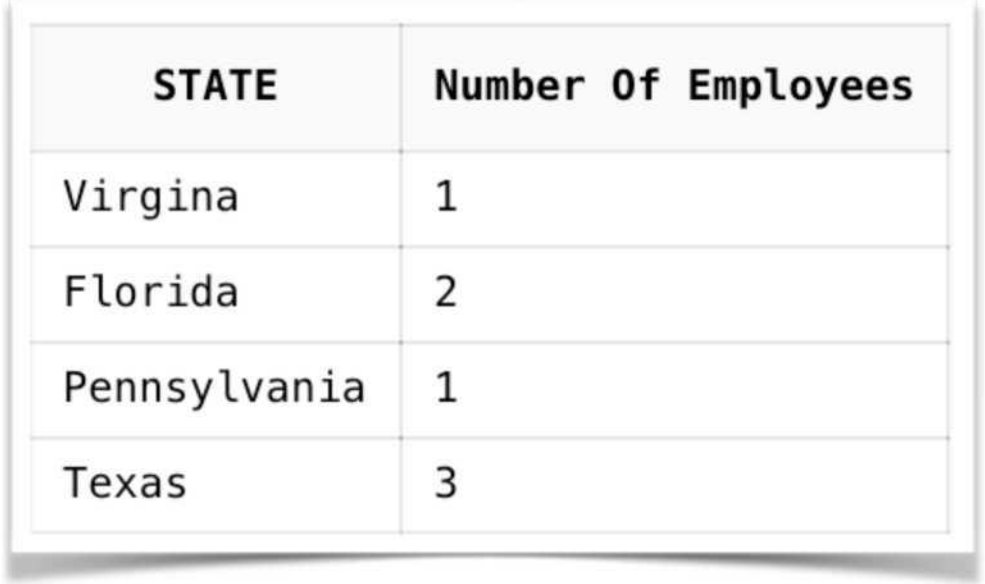
1) Example: Find the total salary for every employee

```
SELECT name, SUM(salary) AS "Total Salary"
FROM employees
GROUP BY name;
```

```
                        NAME Total Salary
        Brad Pitt 3000
----► Mark Stone 2500
,
                Eddie Murphy 11000
```

2) Example: Find the number of employees per state

```
SELECT state, COUNT(state) AS “Number Of Employees"
FROM employees
GROUP BY state;
```



STATE	Number Of Employees
Virgina	1
Florida	2
Pennsylvania	1
Texas	3

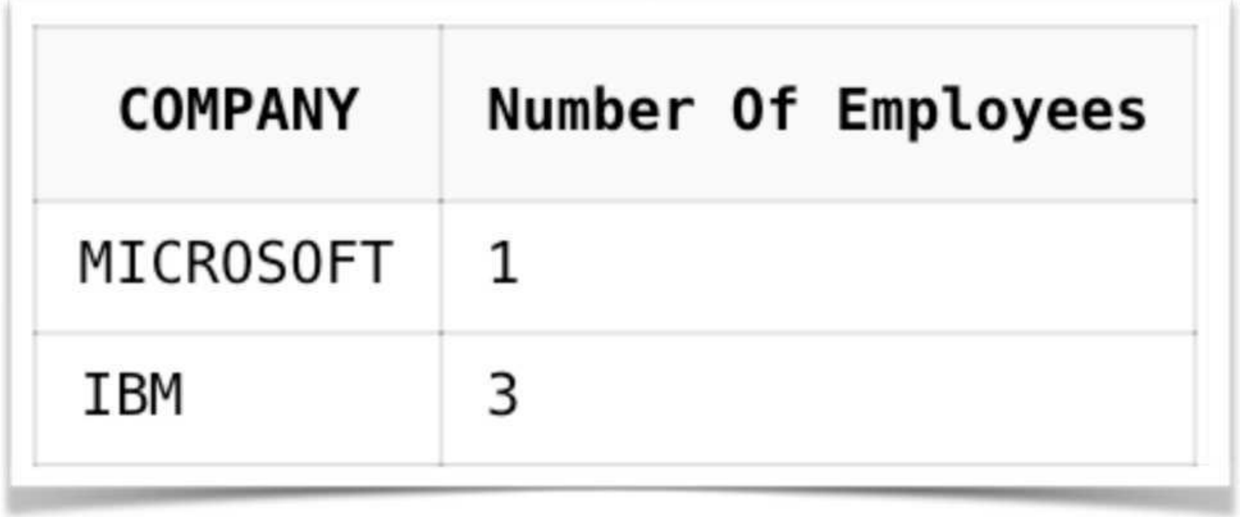
```
CREATE TABLE employees
(
 id number(9), name
 varchar2(50), state
 varchar2(50), salary
 number(20), company
 varchar2(20)
);
```

```
INSERT INTO employees VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');
INSERT INTO employees VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');
INSERT INTO employees VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');
   ► INSERT INTO employees VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');
INSERT INTO employees VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');
INSERT INTO employees VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');
INSERT INTO employees VALUES(123456710, 'Mark Stone', ‘Pennsylvania', 2500, 'IBM');
```

ID	NAME	STATE	SALARY	COMPANY
123456789	John Walker	Florida	2500	IBM
234567890	Brad Pitt	Florida	1500	APPLE
345678901	Eddie Murphy	Texas	3000	IBM
456789012	Eddie Murphy	Virginia	1000	GOOGLE
567890123	Eddie Murphy	Texas	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	GOOGLE
123456710	Mark Stone	Pennsylvania	2500	IBM

3) Example: Find the number of the employees whose salary is more than $2000 per company

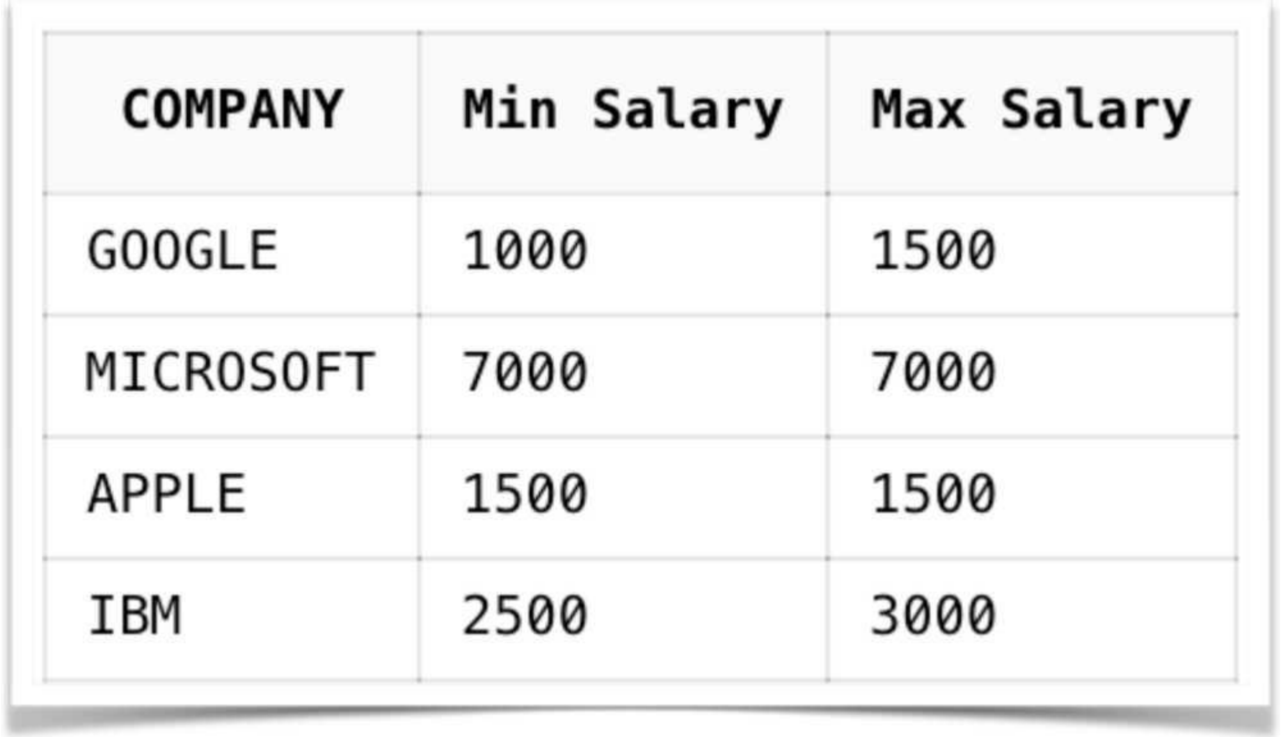
```
 SELECT company, COUNT(*) AS “Number Of Employees"
FROM employees
WHERE salary > 2000
GROUP BY company;
```



COMPANY	Number Of Employees
MICROSOFT	1
IBM	3

4) Example: Find the minimum and maximum salary for every company

```
 SELECT company, MIN(salary) AS “Min Salary”, MAX(salary) AS “Max Salary”
FROM employees
GROUP BY company;
```



COMPANY	Min Salary	Max Salary
GOOGLE	1000	1500
MICROSOFT	7000	7000
APPLE	1500	1500
IBM	2500	3000

“HAVING” Clause

HAVING clause is used in combination with the GROUP BY clause to restrict the groups of returned rows to only those whose the condition is TRUE.

```
CREATE TABLE employees
(
  id number(9), name
  varchar2(50), state
  varchar2(50), salary
  number(20), company
  varchar2(20)
);
```

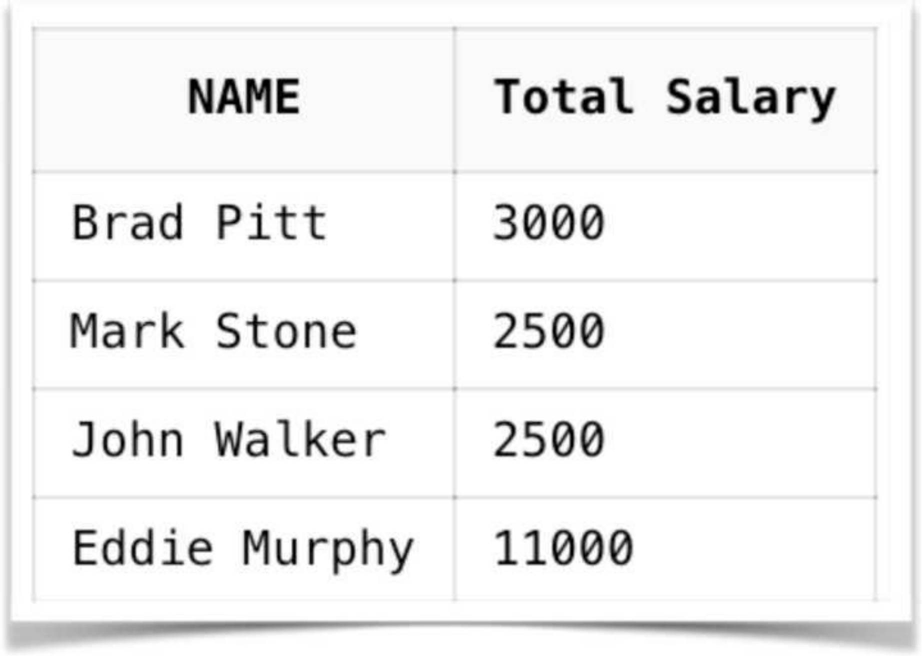
```
INSERT INTO employees VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');
INSERT INTO employees VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');
INSERT INTO employees VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');
  ► INSERT INTO employees VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');
INSERT INTO employees VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');
INSERT INTO employees VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');
INSERT INTO employees VALUES(123456710, 'Mark Stone', 'Pennsylvania', 2500, 'IBM');
```

ID	NAME	STATE	SALARY	COMPANY
123456789	John Walker	Florida	2500	IBM
234567890	Brad Pitt	Florida	1500	APPLE
345678901	Eddie Murphy	Texas	3000	IBM
456789012	Eddie Murphy	Virginia	1000	GOOGLE
567890123	Eddie Murphy	Texas	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	GOOGLE
123456710	Mark Stone	Pennsylvania	2500	IBM

1) Example: Find the total salary if it is greater than 2500 for every employee

```
SELECT name, SUM(salary)
AS "Total Salary" FROM
employees
GROUP BY name
HAVING SUM(salary) >= 2500;
```

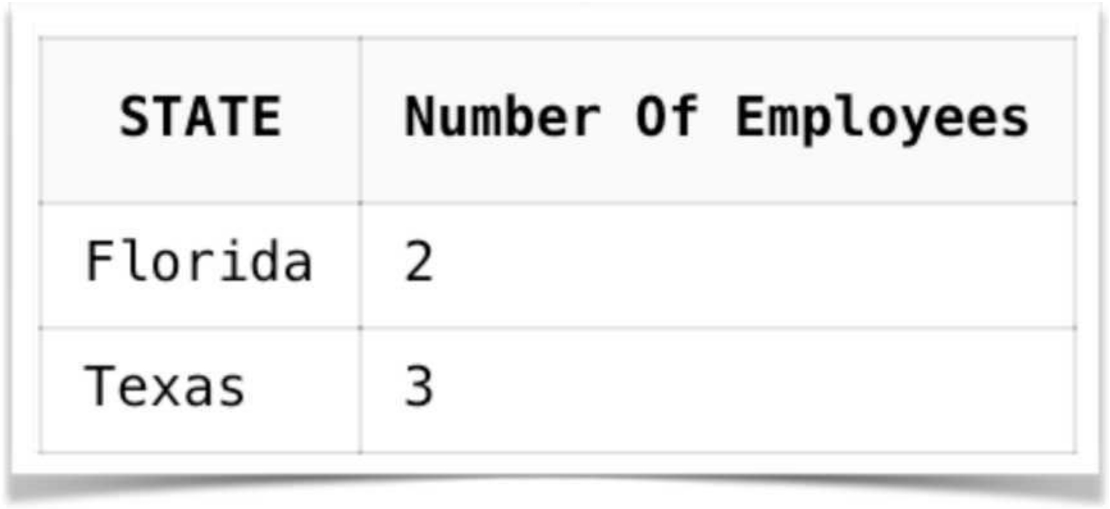
→



NAME	Total Salary
Brad Pitt	3000
Mark Stone	2500
John Walker	2500
Eddie Murphy	11000

2) Example: Find the number of employees if it is more than 1 per state

```
SELECT state, COUNT(state) AS “Number Of Employees"
FROM employees
GROUP BY state
HAVING COUNT(state) > 1;
```



STATE	Number Of Employees
Florida	2
Texas	3

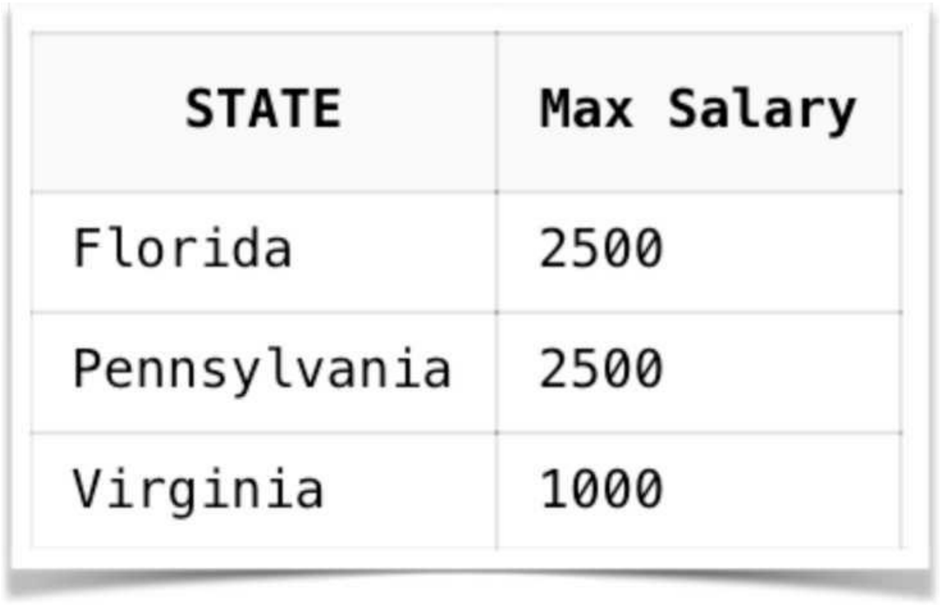
3) Example: Find the minimum salary if it is more than 2000 for every company

```
SELECT company, MIN(salary) AS “Min Salary"
FROM employees
GROUP BY company
HAVING MIN(salary) > 2000;
```

COMPANY	Min Salary
MICROSOFT	7000
IBM	2500

4) Example: Find the maximum salary if it is less than 3000 for every state

```
SELECT state, MAX(salary) AS “Max Salary
FROM employees GROUP BY state
HAVING MAX(salary) < 3000;
```



STATE	Max Salary
Florida	2500
Pennsylvania	2500
Virginia	1000

“UNION” Operator

UNION operator is used to combine the result sets of two or more SELECT statements.

It removes duplicate rows between the SELECT statements.

Each SELECT statement within the UNION operator must have the same number of fields in the result sets with similar data types.

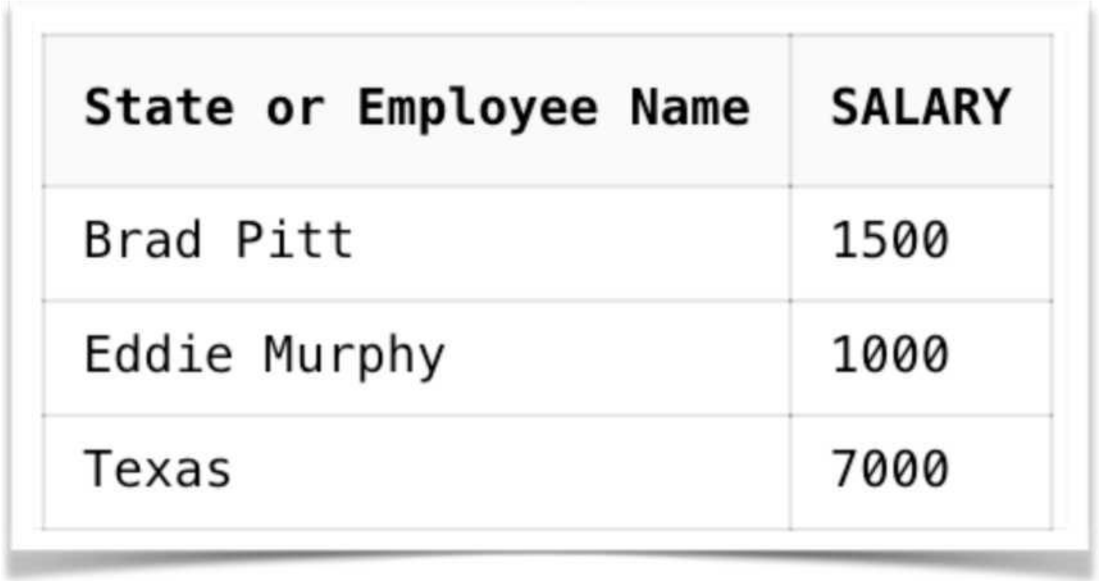
```
CREATE TABLE employees
(
  id number(9), name
  varchar2(50), state
  varchar2(50), salary
  number(20), company
  varchar2(20)
);
```

```
INSERT INTO employees VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');
INSERT INTO employees VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');
INSERT INTO employees VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');
INSERT INTO employees VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');
INSERT INTO employees VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');
INSERT INTO employees VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');
INSERT INTO employees VALUES(123456710, 'Mark Stone', 'Pennsylvania', 2500, 'IBM');
```

ID	NAME	STATE	SALARY	COMPANY
123456789	John Walker	Florida	2500	IBM
234567890	Brad Pitt	Florida	1500	APPLE
345678901	Eddie Murphy	Texas	3000	IBM
456789012	Eddie Murphy	Virginia	1000	GOOGLE
567890123	Eddie Murphy	Texas	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	GOOGLE
123456710	Mark Stone	Pennsylvania	2500	IBM

1) Example: Find the state or employee names whose salary is greater than 3000, less than 2000 without duplication.

```
 SELECT state AS "State or Employee Name", salary
FROM employees
WHERE salary >3000
UNION
SELECT name AS “State or Employee Name", salary
FROM employees
WHERE salary < 2000;
```



State or Employee Name	SALARY
Brad Pitt	1500
Eddie Murphy	1000
Texas	7000

Note: If you add ORDER BY 2 after the last WHERE statement, you get the salary in ascending order.

“UNION ALL” Operator

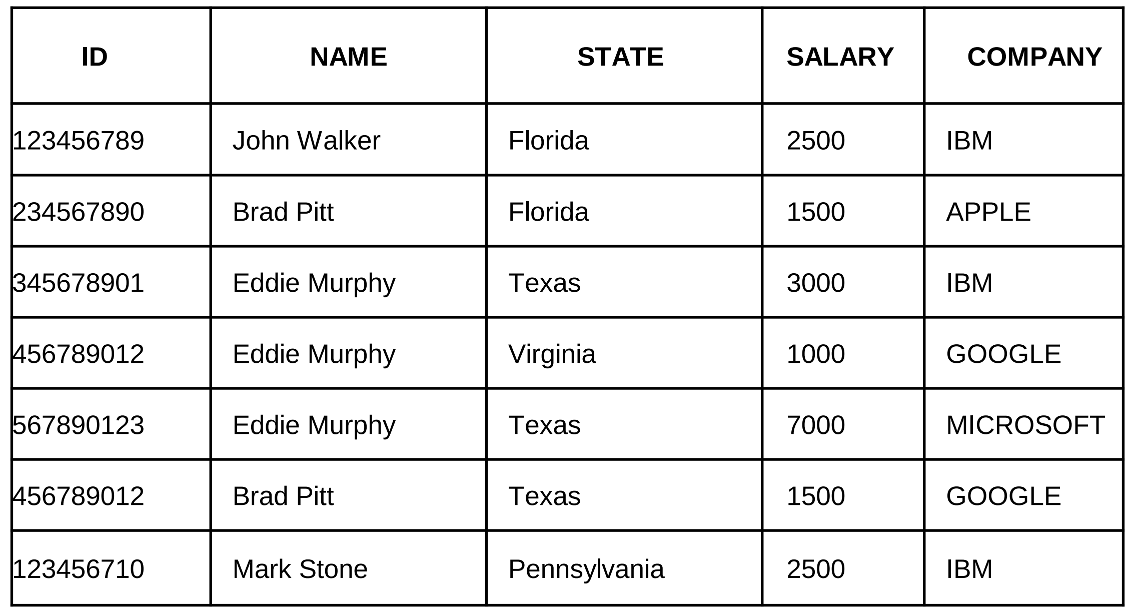
UNION operator is used to combine the result sets of two or more SELECT statements.

It returns all rows from the query and does not remove duplicate rows between the SELECT statements.

Each SELECT statement within the UNION ALL operator must have the same number of fields in the result sets with similar data types.

```
CREATE TABLE employees
(
  id number(9), name
  varchar2(50), state
  varchar2(50), salary
  number(20), company
  varchar2(20)
);
```

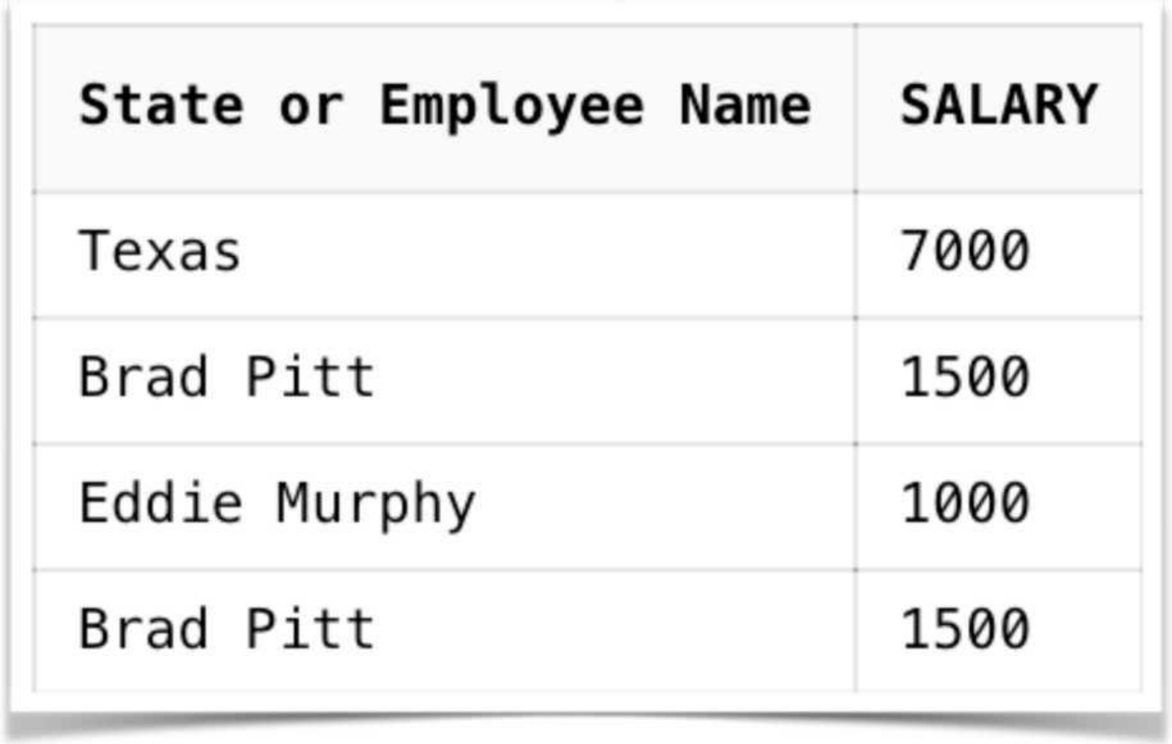
```
INSERT INTO employees VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');
INSERT INTO employees VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');
INSERT INTO employees VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');
k INSERT INTO employees VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');
INSERT INTO employees VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');
INSERT INTO employees VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');
INSERT INTO employees VALUES(123456710, 'Mark Stone', 'Pennsylvania', 2500, 'IBM');
```



ID	NAME	STATE	SALARY	COMPANY
123456789	John Walker	Florida	2500	IBM
234567890	Brad Pitt	Florida	1500	APPLE
345678901	Eddie Murphy	Texas	3000	IBM
456789012	Eddie Murphy	Virginia	1000	GOOGLE
567890123	Eddie Murphy	Texas	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	GOOGLE
123456710	Mark Stone	Pennsylvania	2500	IBM

1) Example: Find all state or employee names whose salary is greater than 3000, less than 2000

```
SELECT state AS "State or Employee Name", salary
FROM employees
WHERE salary >3000
UNION ALL
SELECT name AS "State or Employee Name", salary
FROM employees
WHERE salary < 2000;
```



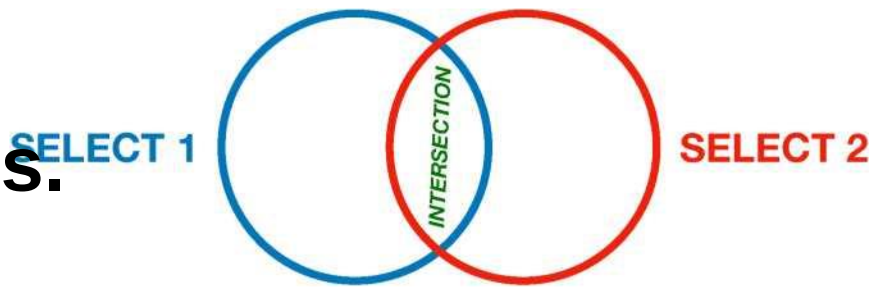
State or Employee Name	SALARY
Texas	7000
Brad Pitt	1500
Eddie Murphy	1000
Brad Pitt	1500

Note: When you use “UNION”, Brad Pitt is printed just once as you can see in the previous slide.

Note: If you add ORDER BY 1 after the last WHERE statement, you get the salary in ascending order.

“INTERCEST” Operator

INTERSECT operator is used to return the common results of 2 or more SELECT statements.



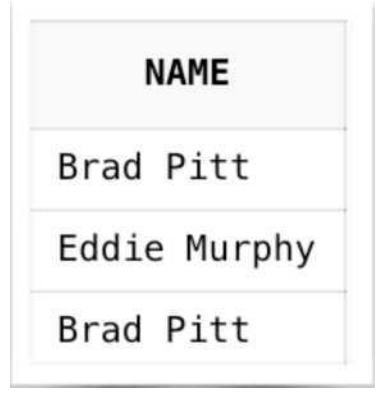
```
CREATE TABLE employees
(
 id number(9), name
 varchar2(50), state
 varchar2(50), salary
 number(20), company
 varchar2(20)
);
```

```
INSERT INTO employees VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');
INSERT INTO employees VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');
INSERT INTO employees VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');
INSERT INTO employees VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');
INSERT INTO employees VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');
INSERT INTO employees VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');
INSERT INTO employees VALUES(123456710, 'Mark Stone', 'Pennsylvania', 2500, 'IBM');
```

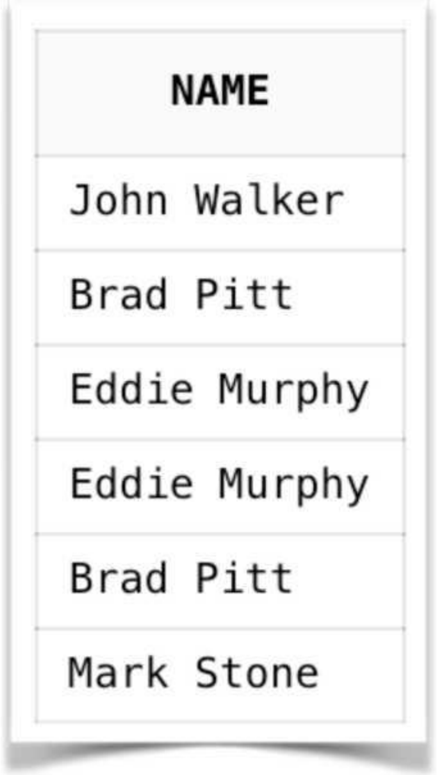
ID	NAME	STATE	SALARY	COMPANY
123456789	John Walker	Florida	2500	IBM
234567890	Brad Pitt	Florida	1500	APPLE
345678901	Eddie Murphy	Texas	3000	IBM
456789012	Eddie Murphy	Virginia	1000	GOOGLE
567890123	Eddie Murphy	Texas	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	GOOGLE
123456710	Mark Stone	Pennsylvania	2500	IBM

1) Example: Find all common employee names whose salary is greater than 1000, less than 2000

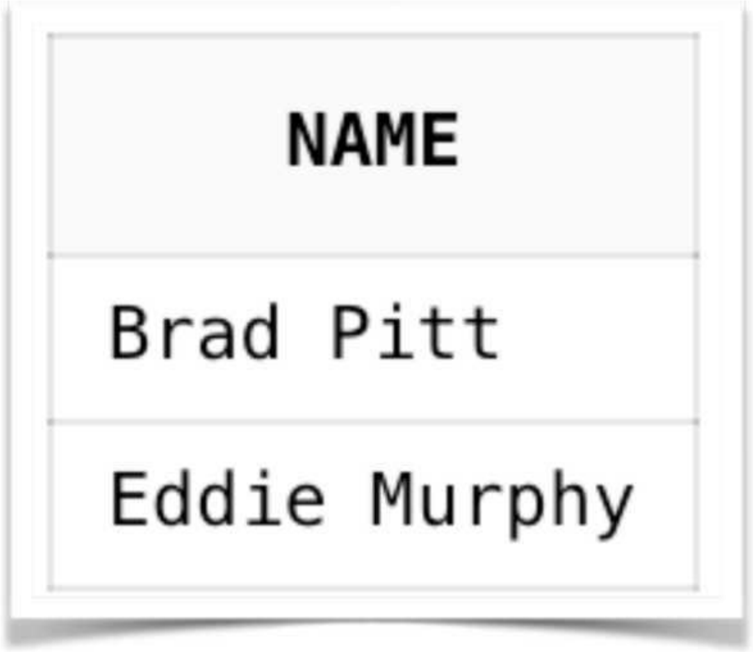
```
SELECT name FROM
employees WHERE
salary < 2000

INTERSECT

SELECT name FROM
employees WHERE
salary > 1000;
```



NAME
Brad Pitt
Eddie Murphy
Brad Pitt



NAME
John Walker
Brad Pitt
Eddie Murphy
Eddie Murphy
Brad Pitt
Mark Stone



NAME
Brad Pitt
Eddie Murphy

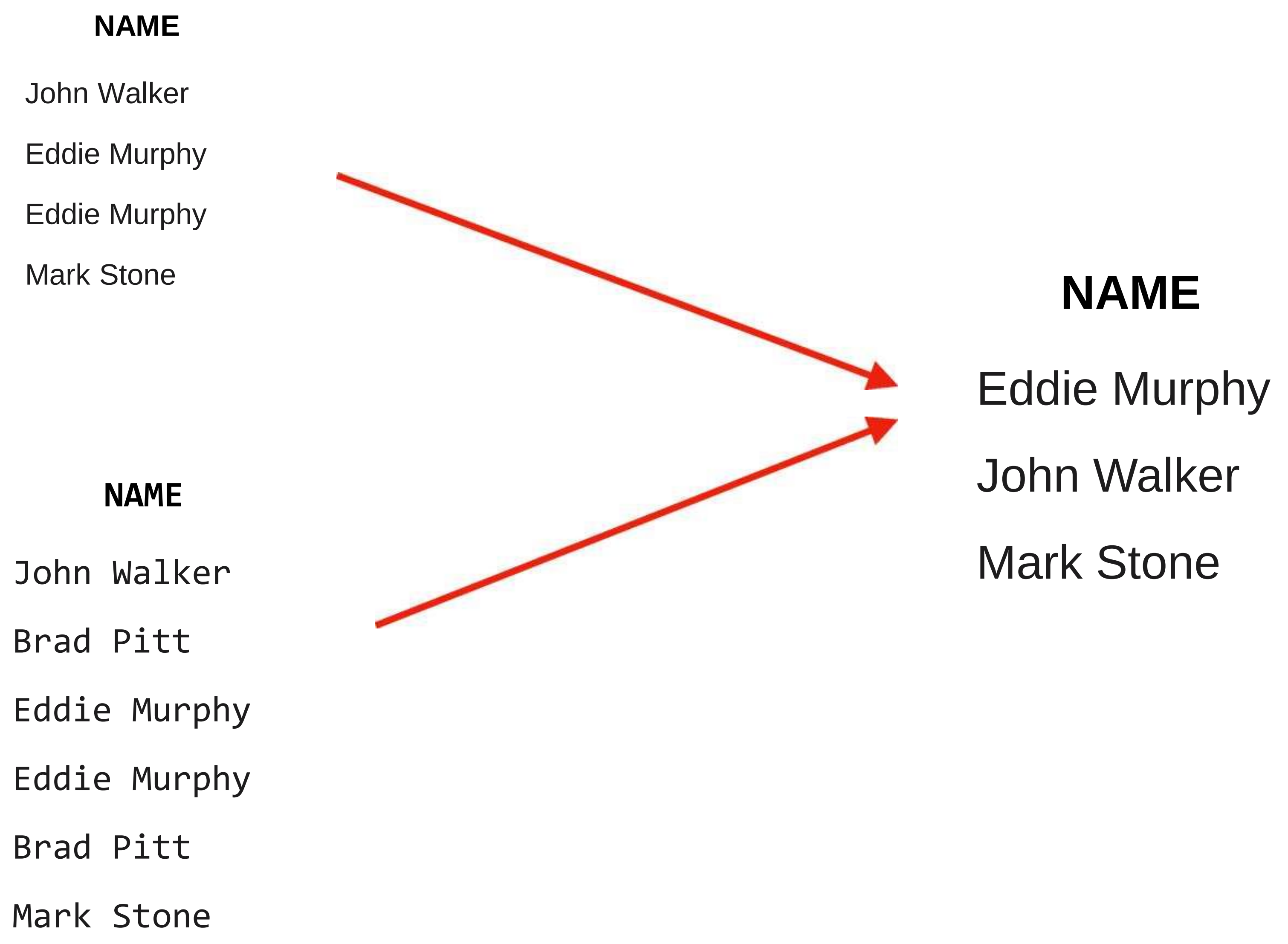
```
CREATE TABLE employees
(
  id number(9), name
  varchar2(50), state
  varchar2(50), salary
  number(20), company
  varchar2(20)
);
```

```
INSERT INTO employees VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');
INSERT INTO employees VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');
INSERT INTO employees VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');
INSERT INTO employees VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');
INSERT INTO employees VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');
INSERT INTO employees VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');
INSERT INTO employees VALUES(123456710, 'Mark Stone', 'Pennsylvania', 2500, 'IBM');
```

ID	NAME	STATE	SALARY	COMPANY
123456789	John Walker	Florida	2500	IBM
234567890	Brad Pitt	Florida	1500	APPLE
345678901	Eddie Murphy	Texas	3000	IBM
456789012	Eddie Murphy	Virginia	1000	GOOGLE
567890123	Eddie Murphy	Texas	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	GOOGLE
123456710	Mark Stone	Pennsylvania	2500	IBM

2) Example: Find all common employee names whose salary is greater than 3000 and company name is IBM, APPLE or GOOGLE

```
SELECT name FROM
employees WHERE
salary > 2000

INTERSECT

SELECT name
FROM employees
WHERE company in (‘IBM’, ‘APPLE’, ‘GOOGLE’);
```



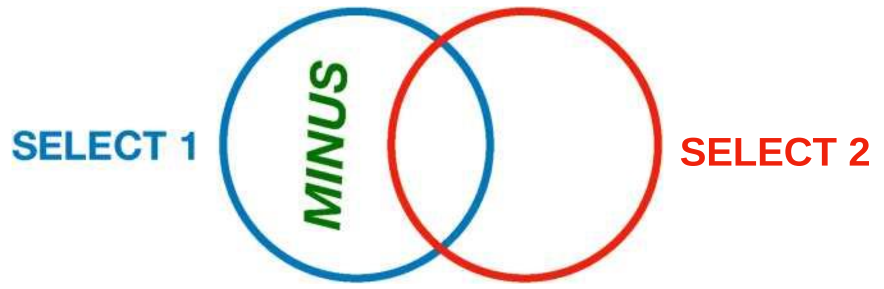
NAME
John Walker
Eddie Murphy
Eddie Murphy
Mark Stone

NAME
John Walker
Brad Pitt
Eddie Murphy
Eddie Murphy
Brad Pitt
Mark Stone

NAME
Eddie Murphy
John Walker
Mark Stone

“MINUS” Operator

MINUS operator is used to return all rows in the first SELECT statement that are not returned by the second SELECT statement.



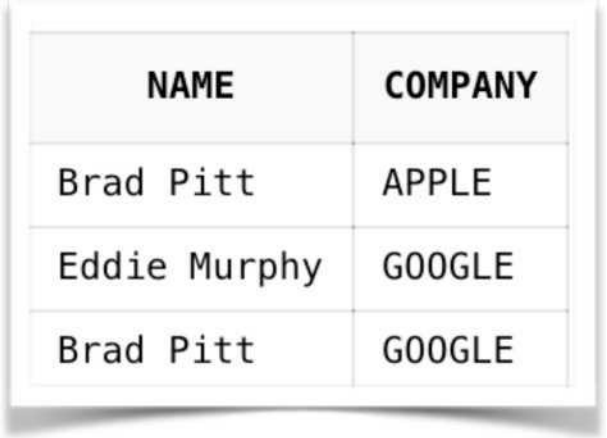
```
CREATE TABLE employees
(
 id number(9), name
 varchar2(50), state
 varchar2(50), salary
 number(20), company
 varchar2(20)
);
```

```
INSERT INTO employees VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');
INSERT INTO employees VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');
INSERT INTO employees VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');
INSERT INTO employees VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');
INSERT INTO employees VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');
INSERT INTO employees VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');
INSERT INTO employees VALUES(123456710, 'Mark Stone', 'Pennsylvania', 2500, 'IBM');
```

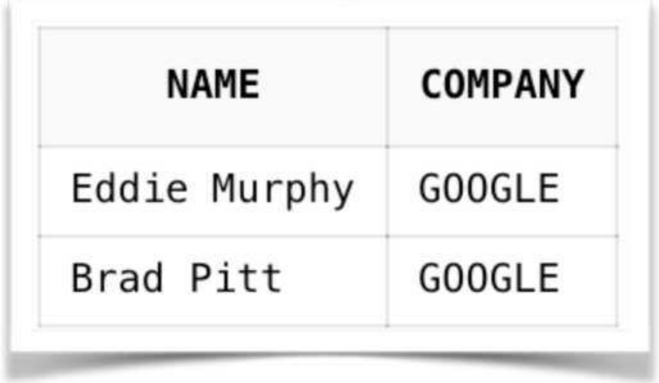
ID	NAME	STATE	SALARY	COMPANY
123456789	John Walker	Florida	2500	IBM
234567890	Brad Pitt	Florida	1500	APPLE
345678901	Eddie Murphy	Texas	3000	IBM
456789012	Eddie Murphy	Virginia	1000	GOOGLE
567890123	Eddie Murphy	Texas	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	GOOGLE
123456710	Mark Stone	Pennsylvania	2500	IBM

2) Example: Find the employee names whose salary is less than 3000 and not working in GOOGLE

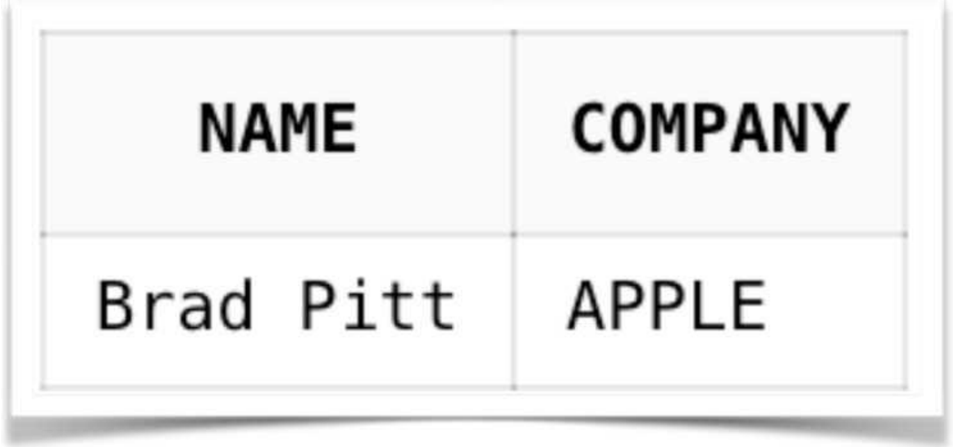
```
SELECT name, company
FROM employees
WHERE salary < 3000

MINUS

SELECT name, company FROM
employees
WHERE company IN (‘GOOGLE’);
```



NAME	COMPANY
Brad Pitt	APPLE
Eddie Murphy	GOOGLE
Brad Pitt	GOOGLE

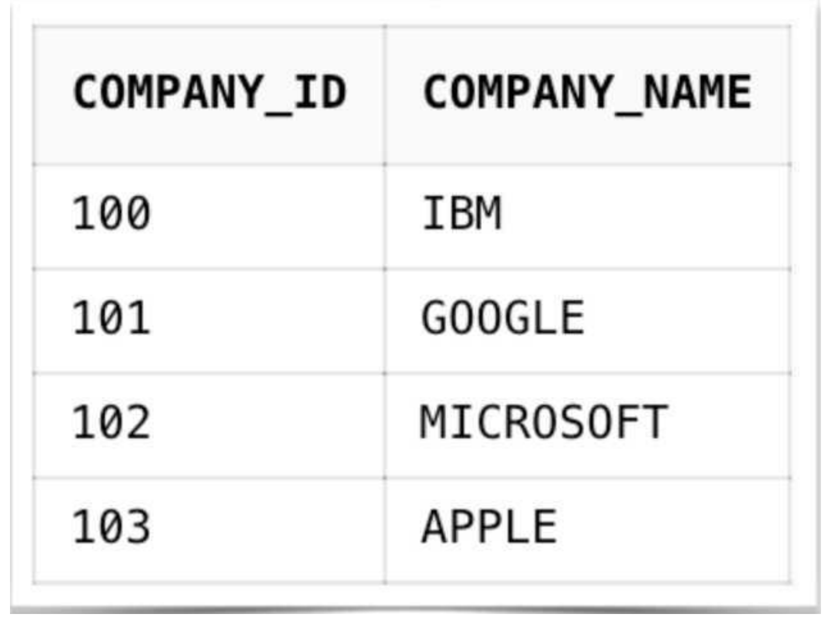


NAME	COMPANY
Eddie Murphy	GOOGLE
Brad Pitt	GOOGLE



NAME	COMPANY
Brad Pitt	APPLE

Review Question



COMPANY_ID	COMPANY_NAME
100	IBM
101	GOOGLE
102	MICROSOFT
103	APPLE

Companies

ORDER_ID	COMPANY_ID	ORDER_DATE
11	101	17-APR-20
22	102	18-APR-20
33	103	19-APR-20
44	104	20-APR-20
55	105	21-APR-20

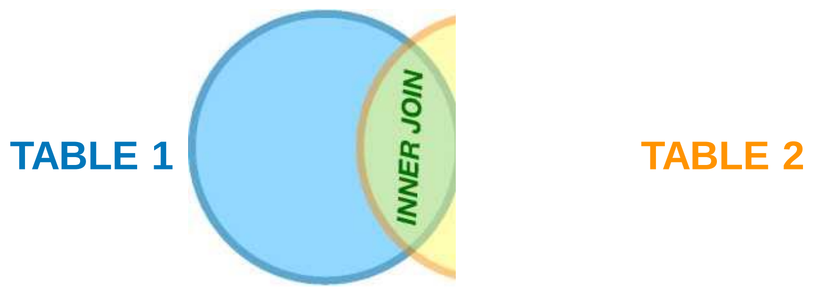
Orders

1) **Create the given tables and insert data**
2) **Find the common company ids**

“JOINS”

1) INNER JOIN

The INNER JOIN would return the common records of two tables



```
CREATE TABLE companies (
  company_id number(9),
  company_name varchar2(20)
);
```

```
INSERT INTO companies VALUES(100, 'IBM');
INSERT INTO companies VALUES(101, 'GOOGLE');
INSERT INTO companies VALUES(102, 'MICROSOFT');
INSERT INTO companies VALUES(103, 'APPLE');
```

COMPANY_ID	COMPANY_NAME
100	IBM
101	GOOGLE
102	MICROSOFT
103	APPLE

```
CREATE TABLE orders
(
  order_id number(9),
  company_id number(9),
  order_date date
);
```

```
      INSERT INTO orders VALUES(11, 101, '17-Apr-2020');
      INSERT INTO orders VALUES(22, 102, '18-Apr-2020');
----► INSERT INTO orders VALUES(33, 103, '19-Apr-2020');
      INSERT INTO orders VALUES(44, 104, '20-Apr-2020');
      INSERT INTO orders VALUES(55, 105, '21-Apr-2020');
```

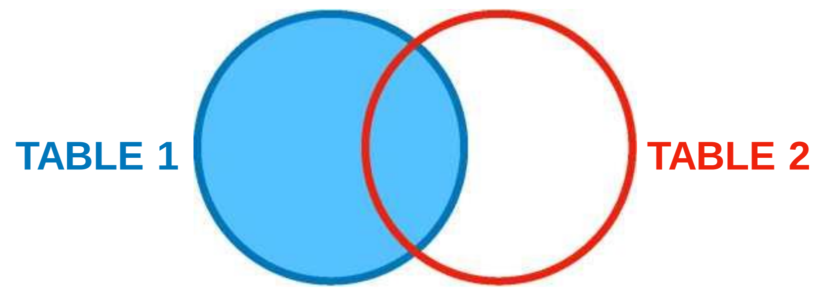
ORDER_ID	COMPANY_ID	ORDER_DATE
11	101	17-APR-20
22	102	18-APR-20
33	103	19-APR-20
44	104	20-APR-20
55	105	21-APR-20

```
SELECT companies.company_name, orders.order_id, orders.order_date
FROM companies INNER JOIN orders
ON companies.company_id = orders.company_id;
```

COMPANY_NAME	ORDER_ID	ORDER_DATE
GOOGLE	11	17-APR-20
MICROSOFT	22	18-APR-20
APPLE	33	19-APR-20

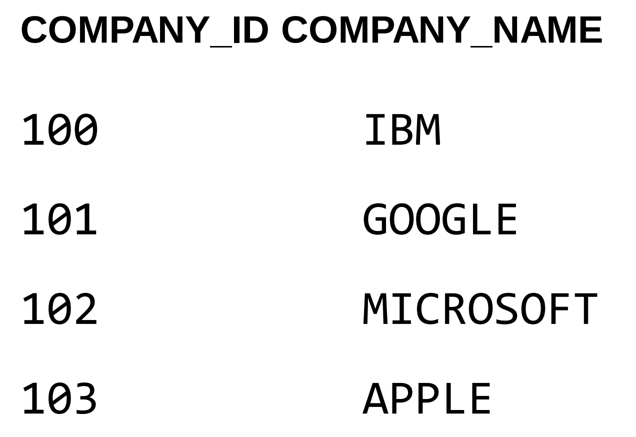
2) LEFT JOIN

The LEFT JOIN returns all rows from the LEFT-Hand table



```
CREATE TABLE companies (
  company_id number(9),
  company_name varchar2(20)
);
```

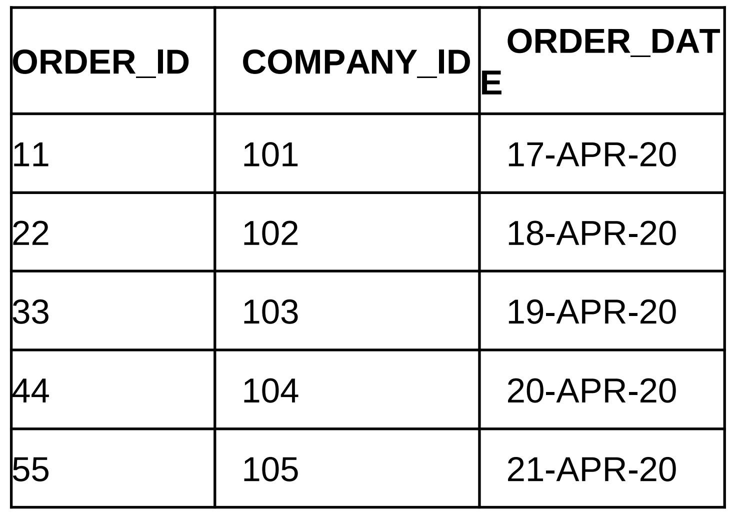
```
INSERT INTO companies VALUES(100, 'IBM');
INSERT INTO companies VALUES(101, 'GOOGLE');
INSERT INTO companies VALUES(102, 'MICROSOFT');
INSERT INTO companies VALUES(103, 'APPLE');
```



```
COMPANY_ID COMPANY_NAME

100           IBM
101           GOOGLE
102           MICROSOFT
103           APPLE
```

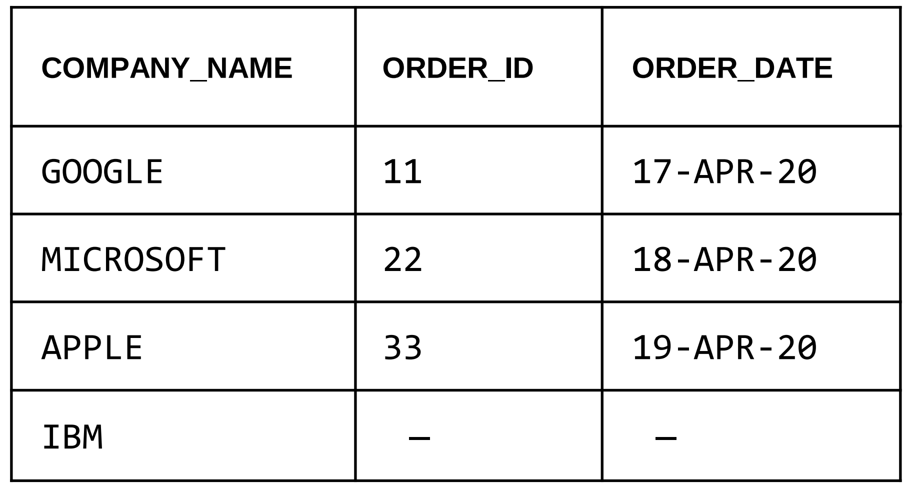
```
CREATE TABLE orders
(
  order_id number(9),
 company_id number(9),
 order_date date
);
```

```
INSERT INTO orders VALUES(11, 101, '17-Apr-2020');
INSERT INTO orders VALUES(22, 102, '18-Apr-2020');
----► INSERT INTO orders VALUES(33, 103, '19-Apr-2020');
INSERT INTO orders VALUES(44, 104, '20-Apr-2020');
INSERT INTO orders VALUES(55, 105, '21-Apr-2020');
```



ORDER_ID	COMPANY_ID	ORDER_DATE
11	101	17-APR-20
22	102	18-APR-20
33	103	19-APR-20
44	104	20-APR-20
55	105	21-APR-20

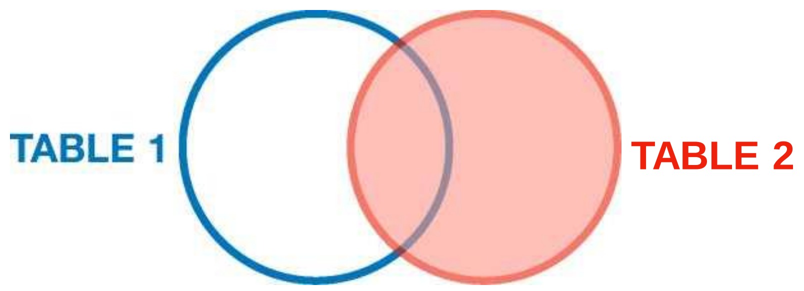
```
SELECT companies.company_name, orders.order_id, orders.order_date
FROM companies LEFT JOIN orders
ON companies.company_id = orders.company_id;
```



COMPANY_NAME	ORDER_ID	ORDER_DATE
GOOGLE	11	17-APR-20
MICROSOFT	22	18-APR-20
APPLE	33	19-APR-20
IBM	–	–

3) RIGHT JOIN

The RIGHT JOIN returns all rows from the RIGHT-Hand table



```
CREATE TABLE companies (
  company_id number(9),
  company_name varchar2(20)
);
```

```
INSERT INTO companies VALUES(100, 'IBM');
INSERT INTO companies VALUES(101, 'GOOGLE');
INSERT INTO companies VALUES(102, 'MICROSOFT');
INSERT INTO companies VALUES(103, 'APPLE');
```

```
COMPANY_ID COMPANY_NAME

100          IBM
101          GOOGLE
102          MICROSOFT
103          APPLE
```

```
CREATE TABLE orders
(
  order_id number(9),
  company_id number(9),
  order_date date
);
```

```
      INSERT INTO orders VALUES(11, 101, '17-Apr-2020');
      INSERT INTO orders VALUES(22, 102, '18-Apr-2020');
----► INSERT INTO orders VALUES(33, 103, '19-Apr-2020');
      INSERT INTO orders VALUES(44, 104, '20-Apr-2020');
      INSERT INTO orders VALUES(55, 105, '21-Apr-2020');
```

ORDER_ID	COMPANY_ID	ORDER_DATE
11	101	17-APR-20
22	102	18-APR-20
33	103	19-APR-20
44	104	20-APR-20
55	105	21-APR-20

```
SELECT companies.company_name, orders.order_id, orders.order_date
FROM companies
RIGHT JOIN orders
ON companies.company_id = orders.company_id;
```

COMPANY-NAME	ORDER_ID	ORDER_DATE
GOOGLE	11	17-APR-20
MICROSOFT	22	18-APR-20
APPLE	33	19-APR-20
—	55	21-APR-20
—	44	20-APR-20

4) FULL JOIN

The FULL JOIN returns all rows from the LEFT-Hand table and RIGHT-Hand table with nulls in place where the join condition is not met.

TABLE 1

```
CREATE TABLE companies (
  company_id number(9),
  company_name varchar2(20)
);
```

```
INSERT INTO companies VALUES(100, 'IBM');
INSERT INTO companies VALUES(101, 'GOOGLE');
INSERT INTO companies VALUES(102, 'MICROSOFT');
INSERT INTO companies VALUES(103, 'APPLE');
```

COMPANY ID	COMPANY NAME
100	IBM
101	GOOGLE
102	MICROSOFT
103	APPLE

TABLE 2

```
CREATE TABLE orders
(
  order_id number(9),
  company_id number(9),
  order_date date
);
```

```
      INSERT INTO orders VALUES(11, 101, '17-Apr-2020');
      INSERT INTO orders VALUES(22, 102, '18-Apr-2020');
----▶ INSERT INTO orders VALUES(33, 103, '19-Apr-2020');  ----1
      INSERT INTO orders VALUES(44, 104, '20-Apr-2020');
      INSERT INTO orders VALUES(55, 105, '21-Apr-2020');
```

ORDER_ID	COMPANY_ID	ORDER_DATE
11	101	17-APR-20
22	102	18-APR-20
33	103	19-APR-20
44	104	20-APR-20
55	105	21-APR-20

```
SELECT companies.company_name, orders.order_id, orders.order_date
FROM companies FULL JOIN orders
ON companies.company_id = orders.company_id;
```

COMPANY-NAME	ORDER_ID	ORDER_DATE
GOOGLE	11	17-APR-20
MICROSOFT	22	18-APR-20
APPLE	33	19-APR-20
—	44	20-APR-20
—	55	21-APR-20
IBM	—	—

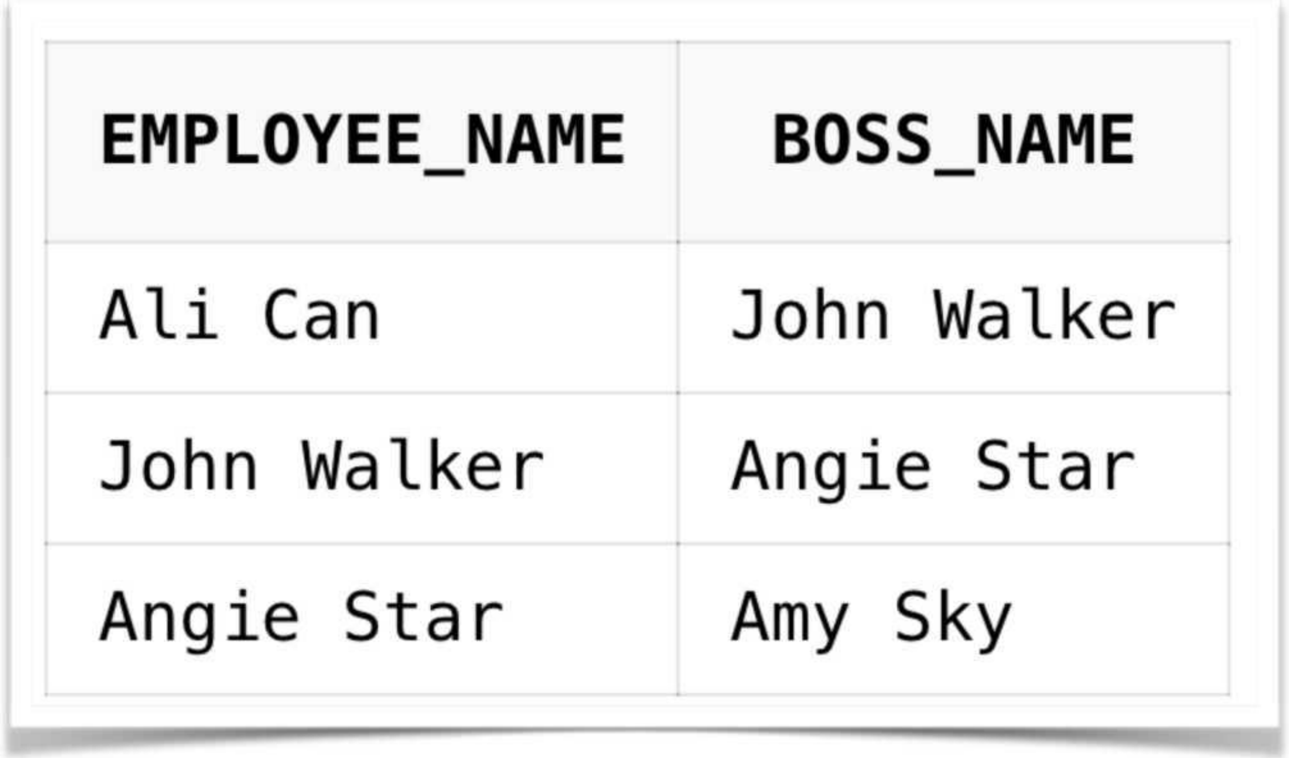
5) SELF JOIN

```
CREATE TABLE employees
(
 id number(2), name
 varchar2(20), title
 varchar2(60), boss_id
 number(2)
);
```

```
INSERT INTO employees VALUES(1, 'Ali Can', 'SDET', 2);
                    INSERT INTO employees VALUES(2, 'John Walker', 'QA', 3);
INSERT INTO employees VALUES(3, 'Angie Star', 'QA Lead', 4);
INSERT INTO employees VALUES(4, 'Amy Sky', 'CEO', 5);
```

ID	NAME	TITLE	BOSS_ID
1	Ali Can	SDET	2
2	John Walker	QA	3
3	Angie Star	QA Lead	4
4	Amy Sky	CEO	5

```
SELECT e1.name AS employee_name, e2.name AS boss_name
FROM employees e1
INNER JOIN employees e2
ON e1.boss_id = e2.id;
```



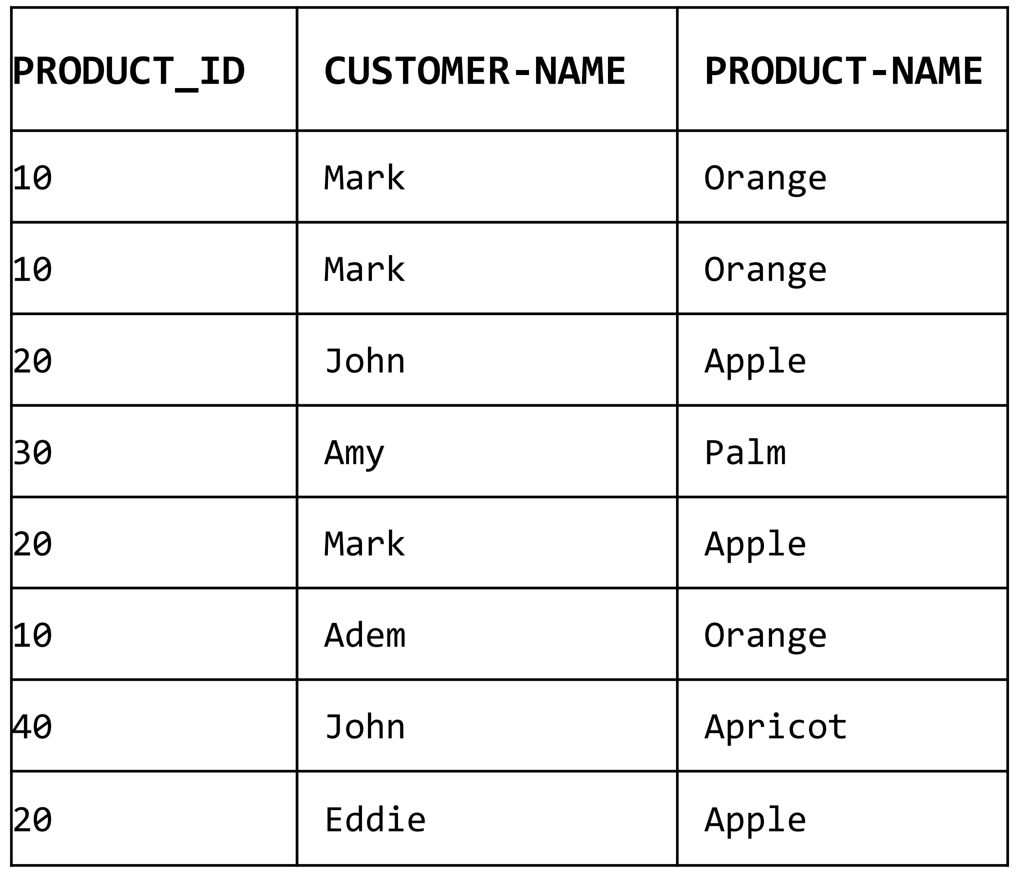
EMPLOYEE_NAME	BOSS_NAME
Ali Can	John Walker
John Walker	Angie Star
Angie Star	Amy Sky

“PIVOT” Clause

PIVOT Clause allows you to aggregate your results and rotate rows into columns

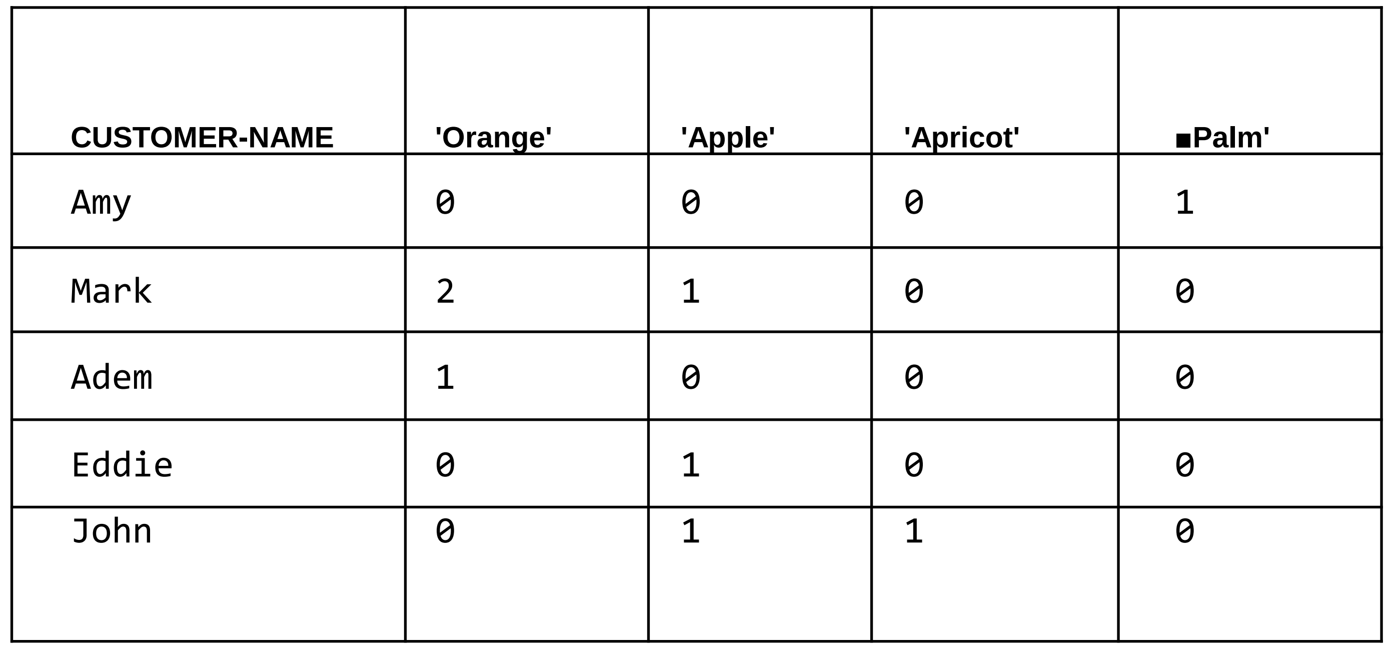
```
CREATE TABLE customers_products (
  product_id number(10),
 customer_name varchar2(50),
 product_name varchar2(50)
);
```

```
INSERT INTO customers_products VALUES (10, 'Mark', 'Orange');
INSERT INTO customers_products VALUES (10, 'Mark', 'Orange');
INSERT INTO customers_products VALUES (20, 'John', 'Apple');
INSERT INTO customers_products VALUES (30, 'Amy', 'Palm');
INSERT INTO customers_products VALUES (20, 'Mark', 'Apple');
INSERT INTO customers_products VALUES (10, 'Adem', 'Orange');
INSERT INTO customers_products VALUES (40, 'John', 'Apricot');
INSERT INTO customers_products VALUES (20, 'Eddie', ‘Apple');
```



PRODUCT_ID	CUSTOMER-NAME	PRODUCT-NAME
10	Mark	Orange
10	Mark	Orange
20	John	Apple
30	Amy	Palm
20	Mark	Apple
10	Adem	Orange
40	John	Apricot
20	Eddie	Apple

```
SELECT * FROM (SELECT product_name,
customer_name FROM customers_products) PIVOT
(COUNT(product_name) FOR product_name IN
('Orange','Apple','Apricot','Palm'));
```



CUSTOMER-NAME	'Orange'	'Apple'	'Apricot'	■Palm'
Amy	0	0	0	1
Mark	2	1	0	0
Adem	1	0	0	0
Eddie	0	1	0	0
John	0	1	1	0

“ALTER TABLE” Statement

ALTER TABLE statement is used to add, modify, or drop/delete columns in a table.
ALTER TABLE statement is also used to rename a table.

```
CREATE TABLE employees
(
 id number(9), name
 varchar2(50), state
 varchar2(50), salary
 number(20), company
 varchar2(20)
);
```

```
INSERT INTO employees VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');
INSERT INTO employees VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');
INSERT INTO employees VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');
INSERT INTO employees VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');
INSERT INTO employees VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');
INSERT INTO employees VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');
INSERT INTO employees VALUES(123456710, 'Mark Stone', 'Pennsylvania', 2500, 'IBM');
```

ID	NAME	STATE	SALARY	COMPANY
123456789	John Walker	Florida	2500	IBM
234567890	Brad Pitt	Florida	1500	APPLE
345678901	Eddie Murphy	Texas	3000	IBM
456789012	Eddie Murphy	Virginia	1000	GOOGLE
567890123	Eddie Murphy	Texas	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	GOOGLE
123456710	Mark Stone	Pennsylvania	2500	IBM

1) ADD a column into a table with a default value

```
ALTER TABLE employees
ADD country_name varchar2(20)
DEFAULT 'USA';
```

ID	NAME	STATE	SALARY	COMPANY	COUNTRY-NAME
123456789	John Walker	Florida	2500	IBM	USA
234567890	Brad Pitt	Florida	1500	APPLE	USA
345678901	Eddie Murphy	Texas	3000	IBM	USA
456789012	Eddie Murphy	Virginia	1000	GOOGLE	USA
567890123	Eddie Murphy	Texas	7000	MICROSOFT	USA
456789012	Brad Pitt	Texas	1500	GOOGLE	USA
123456710	Mark Stone	Pennsylvania	2500	IBM	USA

2) ADD multiple columns into a table

```
ALTER TABLE employees
ADD (gender varchar2(6),
age number(3) );
```

ID	NAME	STATE	SALARY	COMPANY	COUNTRY-NAME	GENDER	AGE
123456789	John Walker	Florida	2500	IBM	USA	—	-
234567890	Brad Pitt	Florida	1500	APPLE	USA	—	—
345678901	Eddie Murphy	Texas	3000	IBM	USA	—	—
456789012	Eddie Murphy	Virginia	1000	GOOGLE	USA	—	—
567890123	Eddie Murphy	Texas	7000	MICROSOFT	USA	-	—
456789012	Brad Pitt	Texas	1500	GOOGLE	USA	—	—
123456710	Mark Stone	Pennsylvania	2500	IBM	USA	—	—

3)DROP COLUMN in a table

ID	NAME	STATE	SALARY	COMPANY	COUNTRY_NAME	GENDER	AGE
123456789	John Walker	Florida	2500	IBM	USA	-	-
234567890	Brad Pitt	Florida	1500	APPLE	USA	-	-
345678901	Eddie Murphy	Texas	3000	IBM	USA	-	-
456789012	Eddie Murphy	Virginia	1000	GOOGLE	USA	-	-
567890123	Eddie Murphy	Texas	7000	MICROSOFT	USA	-	-
456789012	Brad Pitt	Texas	1500	GOOGLE	USA	-	-
123456710	Mark Stone	Pennsylvania	2500	IBM	USA	-	-

```
ALTER TABLE employees DROP
COLUMN age;
```

ID	NAME	STATE	SALARY	COMPANY	COUNTRY_NAME	GENDER
123456789	John Walker	Florida	2500	IBM	USA	-
234567890	Brad Pitt	Florida	1500	APPLE	USA	—
345678901	Eddie Murphy	Texas	3000	IBM	USA	—
456789012	Eddie Murphy	Virginia	1000	GOOGLE	USA	—
567890123	Eddie Murphy	Texas	7000	MICROSOFT	USA	-
456789012	Brad Pitt	Texas	1500	GOOGLE	USA	-
123456710	Mark Stone	Pennsylvania	2500	IBM	USA	-

4)RENAME COLUMN in a table

ID	NAME	STATE	SALARY	COMPANY	COUNTRY-NAME	GENDER
123456789	John Walker	Florida	2500	IBM	USA	-
234567890	Brad Pitt	Florida	1500	APPLE	USA	-
345678901	Eddie Murphy	Texas	3000	IBM	USA	-
456789012	Eddie Murphy	Virginia	1000	GOOGLE	USA	-
567890123	Eddie Murphy	Texas	7000	MICROSOFT	USA	-
456789012	Brad Pitt	Texas	1500	GOOGLE	USA	-
123456710	Mark Stone	Pennsylvania	2500	IBM	USA	-

```
ALTER TABLE employees
RENAME COLUMN company TO company_name;
```

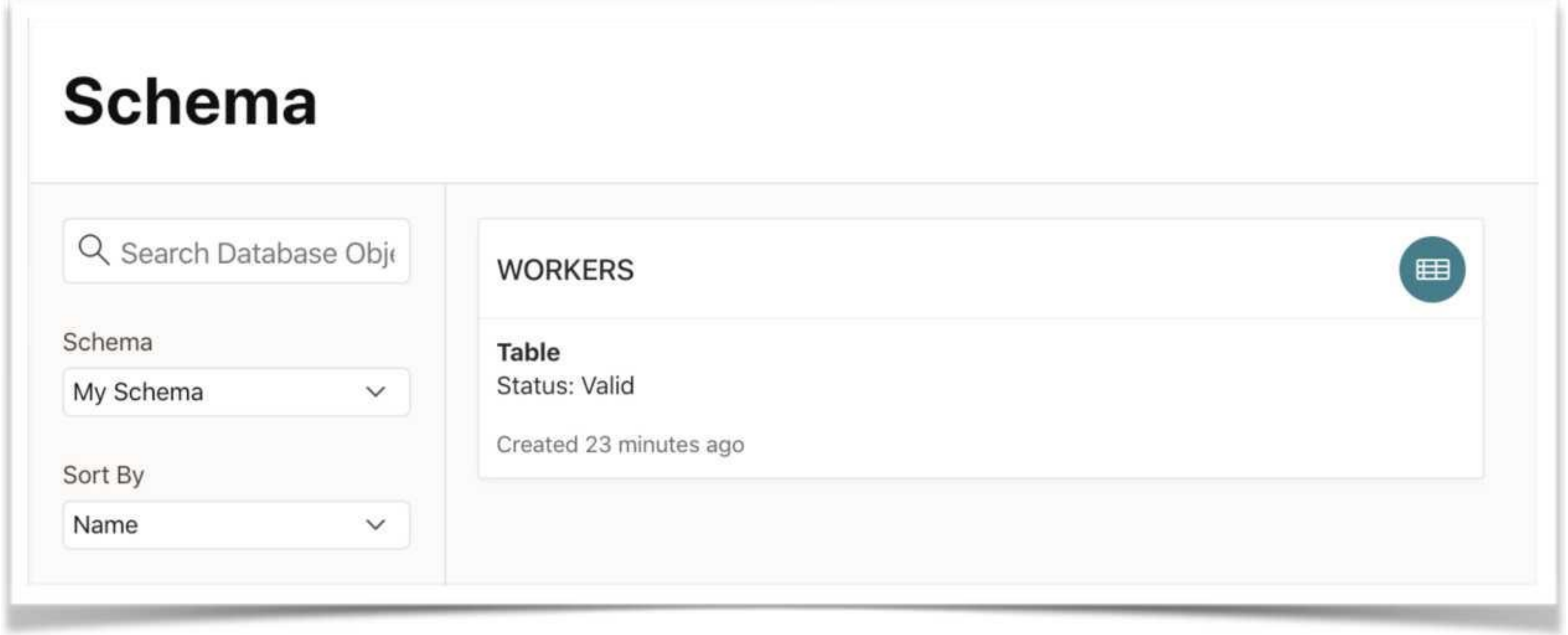
ID	NAME	STATE	SALARY	COMPANY-NAME	COUNTRY-NAME	GENDER
123456789	John Walker	Florida	2500	IBM	USA	-
234567890	Brad Pitt	Florida	1500	APPLE	USA	—
345678901	Eddie Murphy	Texas	3000	IBM	USA	-
456789012	Eddie Murphy	Virginia	1000	GOOGLE	USA	-
567890123	Eddie Murphy	Texas	7000	MICROSOFT	USA	—
456789012	Brad Pitt	Texas	1500	GOOGLE	USA	-
123456710	Mark Stone	Pennsylvania	2500	IBM	USA	—

5)RENAME table name in a table

ALTER TABLE employees
RENAME TO workers;

SELECT * FROM workers;

ID	NAME	STATE	SALARY	COMPANY_NAME	COUNTRY_NAME	GENDER
123456789	John Walker	Florida	2500	IBM	USA	-
234567890	Brad Pitt	Florida	1500	APPLE	USA	—
345678901	Eddie Murphy	Texas	3000	IBM	USA	—
456789012	Eddie Murphy	Virginia	1000	GOOGLE	USA	-
567890123	Eddie Murphy	Texas	7000	MICROSOFT	USA	-
456789012	Brad Pitt	Texas	1500	GOOGLE	USA	—
123456710	Mark Stone	Pennsylvania	2500	IBM	USA	-



Schema

Search Database Obj

Schema: My Schema

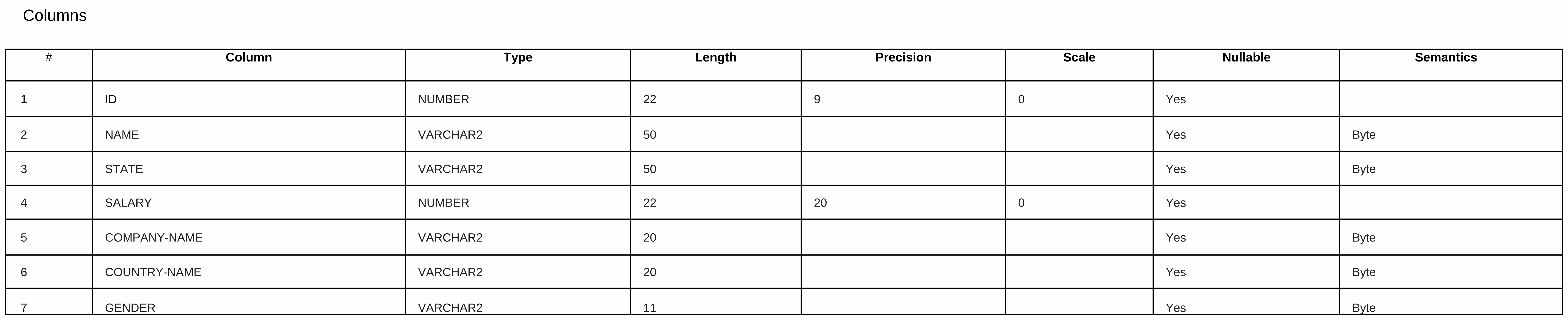
Sort By: Name

WORKERS

Table
Status: Valid

Created 23 minutes ago

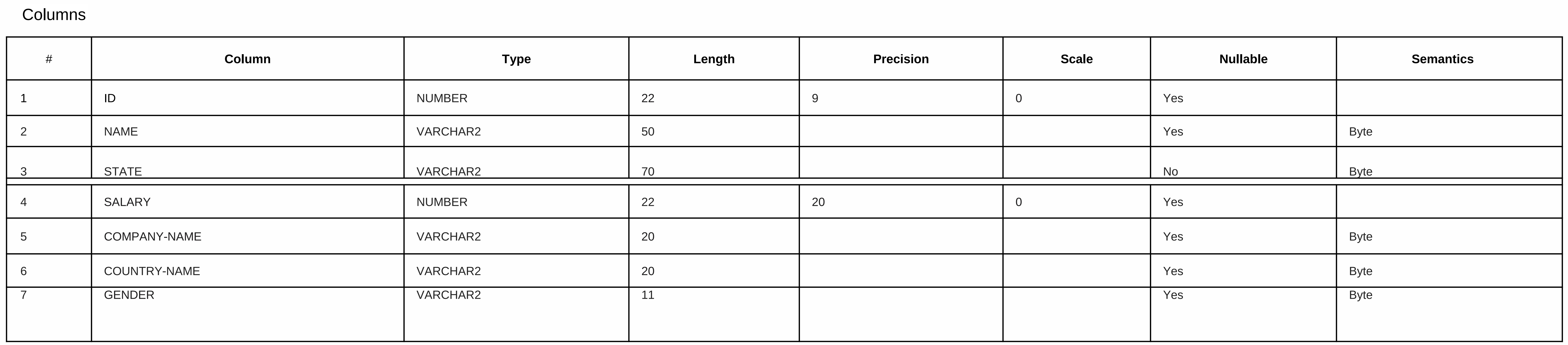
6)MODIFY column or columns in a table



Columns

#	Column	Type	Length	Precision	Scale	Nullable	Semantics
1	ID	NUMBER	22	9	0	Yes	
2	NAME	VARCHAR2	50			Yes	Byte
3	STATE	VARCHAR2	50			Yes	Byte
4	SALARY	NUMBER	22	20	0	Yes	
5	COMPANY-NAME	VARCHAR2	20			Yes	Byte
6	COUNTRY-NAME	VARCHAR2	20			Yes	Byte
7	GENDER	VARCHAR2	11			Yes	Byte

```
ALTER TABLE workers
MODIFY state varchar2(70) NOT NULL;
```



Columns

#	Column	Type	Length	Precision	Scale	Nullable	Semantics
1	ID	NUMBER	22	9	0	Yes	
2	NAME	VARCHAR2	50			Yes	Byte
3	STATE	VARCHAR2	70			No	Byte
4	SALARY	NUMBER	22	20	0	Yes	
5	COMPANY-NAME	VARCHAR2	20			Yes	Byte
6	COUNTRY-NAME	VARCHAR2	20			Yes	Byte
7	GENDER	VARCHAR2	11			Yes	Byte

To modify multiple columns

```
ALTER TABLE workers MODIFY (state
varchar2(70) NOT NULL, id number(11)
NULL);
```

SQL Technical Interview Questions

```
CREATE TABLE students
(
 id number(9), name
 varchar2(50), state
 varchar2(50), salary
 number(20), company
 varchar2(20)
);
```

```
INSERT INTO students VALUES(123456789, 'Johnny Walk', 'New Hampshire', 2500, 'IBM');
INSERT INTO students VALUES(234567891, 'Brian Pitt', 'Florida', 1500, 'LINUX');
INSERT INTO students VALUES(245678901, 'Eddie Murphy', 'Texas', 3000, 'WELLS FARGO');
INSERT INTO students VALUES(456789012, 'Teddy Murphy', 'Virginia', 1000, 'GOOGLE');
INSERT INTO students VALUES(567890124, 'Eddie Murphy', 'Massachuset', 7000, 'MICROSOFT');
INSERT INTO students VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'TD BANK');
INSERT INTO students VALUES(123456719, 'Adem Stone', 'New Jersey', 2500, 'IBM');
```

1

ID	NAME	STATE	SALARY	COMPANY
123456789	Johnny Walk	New Hampshire	2500	IBM
234567891	Brian Pitt	Florida	1500	LINUX
245678901	Eddie Murphy	Texas	3000	WELLS FARGO
456789012	Teddy Murphy	Virginia	1000	GOOGLE
567890124	Eddie Murphy	Massachuset	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	TD BANK
123456719	Adem Stone	New Jersey	2500	IBM

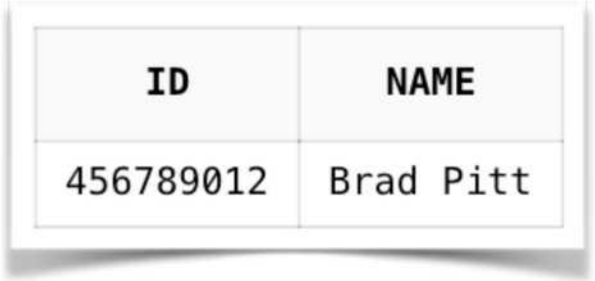
```
CREATE TABLE employees
(
  id number(9),
 name varchar2(50), state
 varchar2(50), salary
 number(20), company
 varchar2(20)
);
```

```
INSERT INTO employees VALUES(123456789, 'John Walker', 'Florida', 2500, 'IBM');
INSERT INTO employees VALUES(234567890, 'Brad Pitt', 'Florida', 1500, 'APPLE');
INSERT INTO employees VALUES(345678901, 'Eddie Murphy', 'Texas', 3000, 'IBM');
INSERT INTO employees VALUES(456789012, 'Eddie Murphy', 'Virginia', 1000, 'GOOGLE');
INSERT INTO employees VALUES(567890123, 'Eddie Murphy', 'Texas', 7000, 'MICROSOFT');
INSERT INTO employees VALUES(456789012, 'Brad Pitt', 'Texas', 1500, 'GOOGLE');
INSERT INTO employees VALUES(123456710, 'Mark Stone', 'Pennsylvania', 2500, 'IBM');
```

ID	NAME	STATE	SALARY	COMPANY
123456789	John Walker	Florida	2500	IBM
234567890	Brad Pitt	Florida	1500	APPLE
345678901	Eddie Murphy	Texas	3000	IBM
456789012	Eddie Murphy	Virginia	1000	GOOGLE
567890123	Eddie Murphy	Texas	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	GOOGLE
123456710	Mark Stone	Pennsylvania	2500	IBM

1) How to fetch common records from two tables?

```
SELECT id, name
FROM students
INTERSECT
SELECT id, name
FROM employees;
```



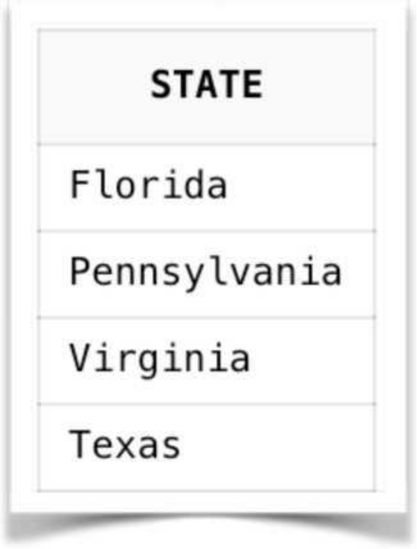
ID	NAME
456789012	Brad Pitt

```
SELECT name
FROM students
INTERSECT
SELECT name
FROM employees;
```

NAME
Brad Pitt
Eddie Murphy

2) How to fetch unique records from a table?

```
SELECT DISTINCT state
FROM employees;
```



STATE
Florida
Pennsylvania
Virginia
Texas

Note: DISTINCT Clause is used to remove duplicates from the result set.

3) What is the command used to fetch even id's?

```
SELECT *
FROM students
WHERE MOD(id,2)=0;
```

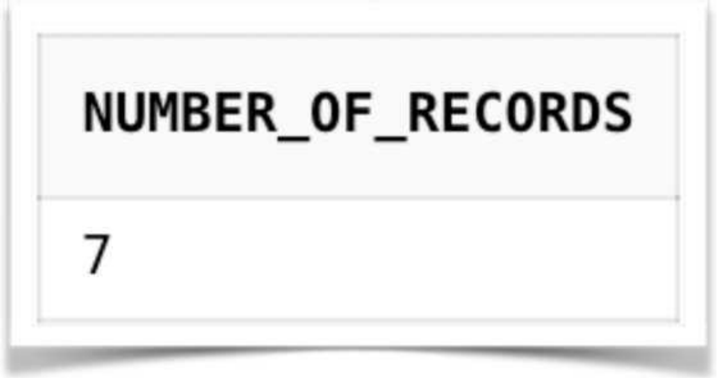
ID	NAME	STATE	SALARY	COMPANY
456789012	Teddy Murphy	Virginia	1000	GOOGLE
567890124	Eddie Murphy	Massachuset	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	TD BANK

Note: To fetch odd id's use the following script

```
SELECT *
FROM students
WHERE MOD(id,2)=1;
```

4) What is the command to count records in a table?

```
 SELECT COUNT(*) AS number_of_records
FROM students;
```

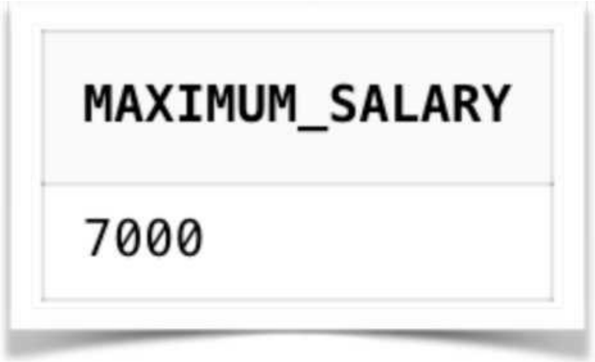


NUMBER_OF_RECORDS
7

5) What is the SQL Query to get the highest salary of a worker from a table?

```
SELECT MAX(salary) AS maximum_salary
FROM workers;
```

```
SELECT salary
FROM employees
ORDER BY salary DESC
FETCH NEXT 1 ROW ONLY;
```

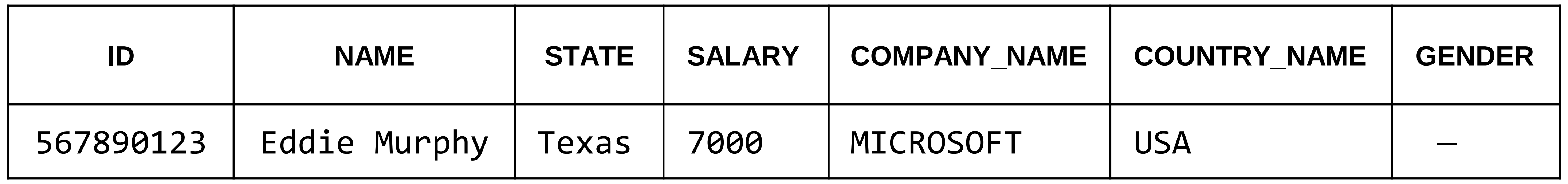


MAXIMUM_SALARY
7000

6) What is the SQL Query to get all records about the worker who has the highest salary from a table?

```
SELECT *
FROM workers
WHERE salary = (SELECT MAX(salary)
                      FROM workers);
```

```
SELECT *
FROM employees ORDER
BY salary DESC FETCH
NEXT 1 ROW ONLY;
```

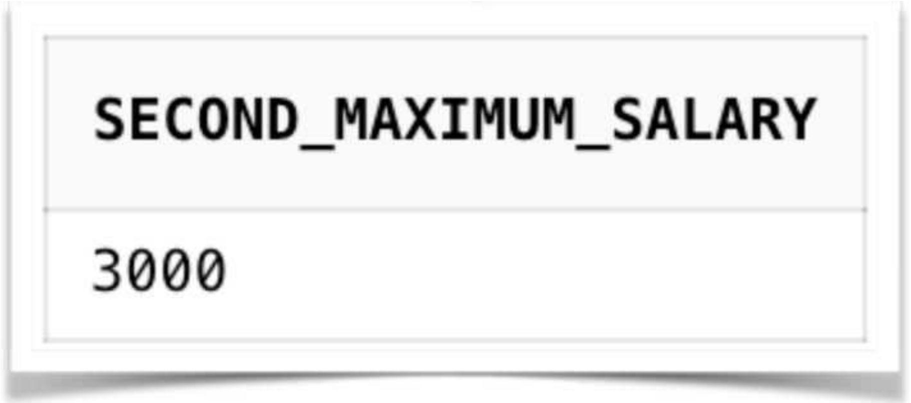


ID	NAME	STATE	SALARY	COMPANY_NAME	COUNTRY_NAME	GENDER
567890123	Eddie Murphy	Texas	7000	MICROSOFT	USA	–

7) What is the SQL Query to get the second highest salary of a worker from a table?

```
SELECT MAX(salary) AS second_maximum_salary
FROM employees
WHERE salary < (SELECT MAX(salary)
                FROM employees);
```

```
SELECT salary FROM
employees ORDER BY
salary DESC OFFSET 1
ROW
FETCH NEXT 1 ROW ONLY;
```



```
SECOND_MAXIMUM_SALARY
3000
```

8) What is the SQL Query to get all records about the worker who has the second highest salary from a table?

ID	NAME	STATE	SALARY	COMPANY
123456789	John Walker	Florida	2500	IBM
234567890	Brad Pitt	Florida	1500	APPLE
345678901	Eddie Murphy	Texas	3000	IBM
456789012	Eddie Murphy	Virginia	1000	GOOGLE
567890123	Eddie Murphy	Texas	7000	MICROSOFT
456789012	Brad Pitt	Texas	1500	GOOGLE
123456710	Mark Stone	Pennsylvania	2500	IBM

```
SELECT *
FROM (SELECT *
FROM employees
WHERE salary < (SELECT MAX(salary)
FROM employees)
ORDER BY salary DESC)
WHERE ROWNUM=1;
```

ID	NAME	STATE	SALARY	COMPANY_NAME	COUNTRY_NAME	GENDER
345678901	Eddie Murphy	Texas	3000	IBM	USA	—

```
SELECT * FROM employees
ORDER BY salary DESC
OFFSET 1 ROW
FETCH NEXT 1 ROW ONLY;
```

9) What is the SQL Query to get all records from a column in uppercase from a table?

NAME	STATE
John Walker	Florida
Brad Pitt	Florida
Eddie Murphy	Texas
Eddie Murphy	Virginia
Eddie Murphy	Texas
Brad Pitt	Texas
Mark Stone	Pennsylvania

```
SELECT name, UPPER(state)
FROM workers;
```

NAME	UPPER(STATE)
John Walker	FLORIDA
Brad Pitt	FLORIDA
Eddie Murphy	TEXAS
Eddie Murphy	VIRGINIA
Eddie Murphy	TEXAS
Brad Pitt	TEXAS
Mark Stone	PENNSYLVANIA

10) What is the SQL Query to get all records from a column in lowercase from a table?

NAME	STATE
John Walker	Florida
Brad Pitt	Florida
Eddie Murphy	Texas
Eddie Murphy	Virginia
Eddie Murphy	Texas
Brad Pitt	Texas
Mark Stone	Pennsylvania

```
SELECT name, LOWER(state)
FROM workers;
```

NAME	LOWER(STATE)
John Walker	florida
Brad Pitt	florida
Eddie Murphy	texas
Eddie Murphy	Virginia
Eddie Murphy	texas
Brad Pitt	texas
Mark Stone	Pennsylvania

11) What is the SQL Query to get all records from a column in initials uppercase rests lowercase from a table?

NAME	COMPANY_NAME
John Walker	IBM
Brad Pitt	APPLE
Eddie Murphy	IBM
Eddie Murphy	GOOGLE
Eddie Murphy	MICROSOFT
Brad Pitt	GOOGLE
Mark Stone	IBM

```
SELECT name, INITCAP(state)
FROM workers;
```

NAME	INITCAP(COMPANY_NAME)
John Walker	Ibm
Brad Pitt	Apple
Eddie Murphy	Ibm
Eddie Murphy	Google
Eddie Murphy	Microsoft
Brad Pitt	Google
Mark Stone	Ibm