DATABASE MANAGEMENT SYSTEMS

Couse code:CSC403

Prof. Juhi Janjua

Module 4: Structured Query Language (SQL)

- Overview of SQL
- Data Definition Commands
- Integrity constraints: key constraints, Domain Constraints, Referential integrity , check constraints
- Data Manipulation commands
- Data Control commands
- Set and string operations
- Aggregate function-group by, having
- Views in SQL, joins
- Nested and complex queries
- Triggers

What is SQL?

- SQL stands for **Structured Query Language**.
- It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in a number of ways, using English-like statements.

SQL Rules

SQL follows the following rules:

- Structure query language is not case sensitive. Generally, keywords of SQL are written in uppercase.
- Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.
- Using the SQL statements, you can perform most of the actions in a database.

Data Definition Language(DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE

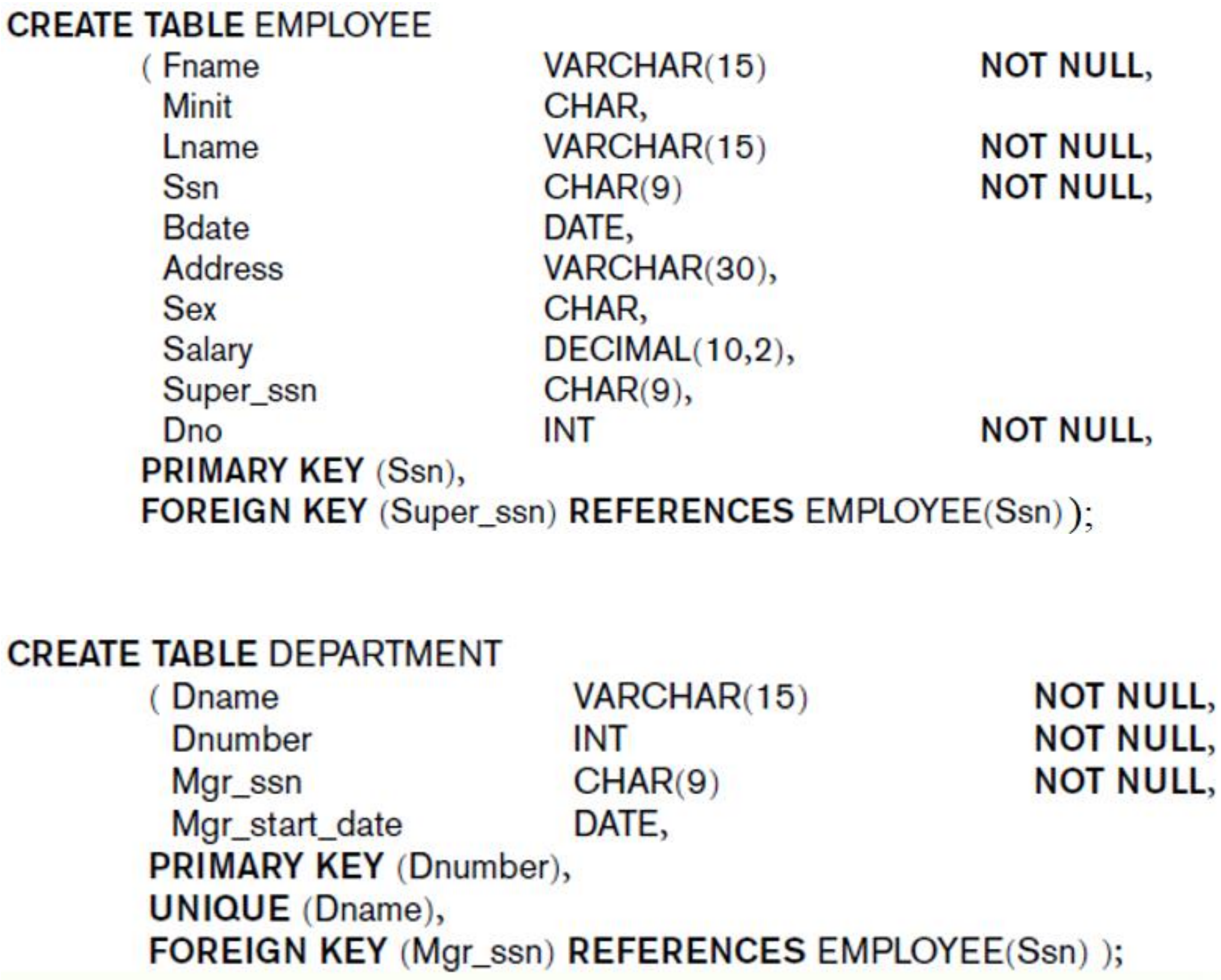
SQL CREATE table

- Creating a basic table involves naming the table and defining its columns and each column's data type (INTEGER, FLOAT, DECIMAL(i,j), CHAR(n), VARCHAR(n)).

Syntax:

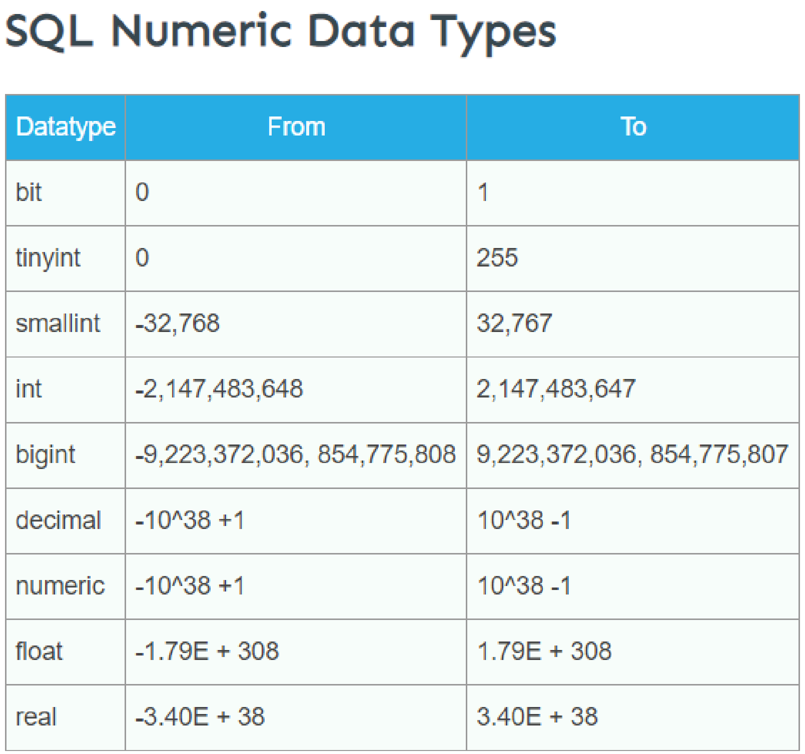
```
CREATE TABLE table_name(
  column1 datatype,
  column2 datatype,
  column3 datatype,
  .....
  columnN datatype,
);
```

Example



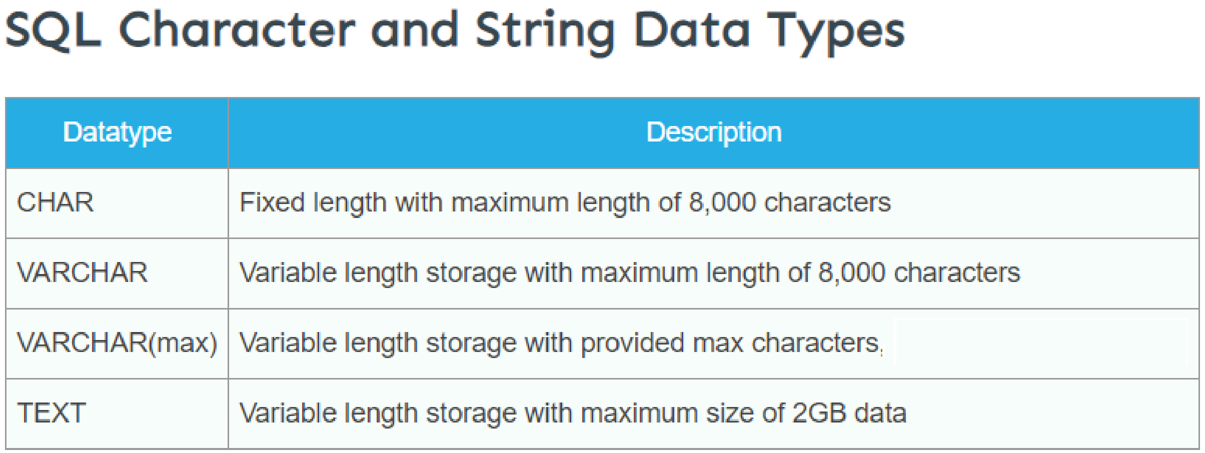
```
CREATE TABLE EMPLOYEE
        ( Fname              VARCHAR(15)          NOT NULL,
          Minit              CHAR,
          Lname              VARCHAR(15)          NOT NULL,
          Ssn                CHAR(9)              NOT NULL,
          Bdate              DATE,
          Address            VARCHAR(30),
          Sex                CHAR,
          Salary             DECIMAL(10,2),
          Super_ssn          CHAR(9),
          Dno                INT                  NOT NULL,
        PRIMARY KEY (Ssn),
        FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn) );


CREATE TABLE DEPARTMENT
        ( Dname              VARCHAR(15)          NOT NULL,
          Dnumber            INT                  NOT NULL,
          Mgr_ssn            CHAR(9)              NOT NULL,
          Mgr_start_date     DATE,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
```

SQL Datatype



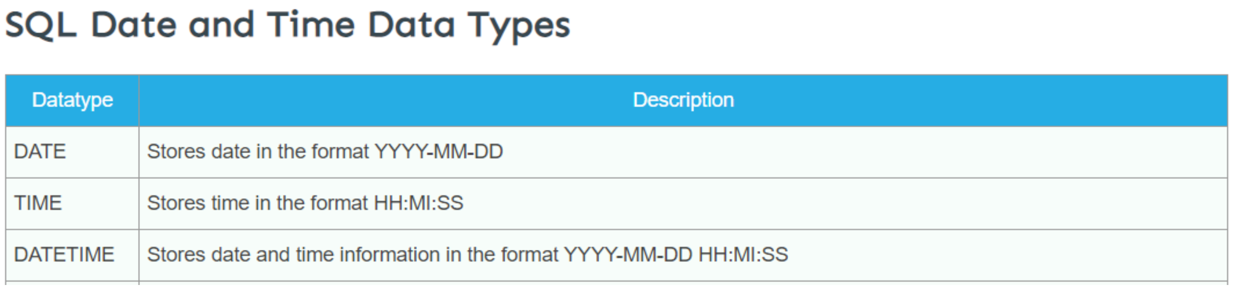
SQL Numeric Data Types

Datatype	From	To
bit	0	1
tinyint	0	255
smallint	-32,768	32,767
int	-2,147,483,648	2,147,483,647
bigint	-9,223,372,036, 854,775,808	9,223,372,036, 854,775,807
decimal	-10^38 +1	10^38 -1
numeric	-10^38 +1	10^38 -1
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38



SQL Character and String Data Types

Datatype	Description
CHAR	Fixed length with maximum length of 8,000 characters
VARCHAR	Variable length storage with maximum length of 8,000 characters
VARCHAR(max)	Variable length storage with provided max characters
TEXT	Variable length storage with maximum size of 2GB data



SQL Date and Time Data Types

Datatype	Description
DATE	Stores date in the format YYYY-MM-DD
TIME	Stores time in the format HH:MI:SS
DATETIME	Stores date and time information in the format YYYY-MM-DD HH:MI:SS

SQL ALTER table

- ALTER TABLE statement specifies how to add, modify, drop or delete columns in a table. It is also used to rename a table.

Adding columns in a table:

```
ALTER TABLE customers

ADD customer_age INT;
```

Adding multiple columns in the existing table:

```
ALTER TABLE customers

  ADD (customer_type varchar2(50),

       customer_address varchar2(50));
```

SQL ALTER table …

- **Modifying column of a table:**

```
ALTER TABLE customers
 MODIFY customer_address varchar2(100)
```

- **Dropping column of a table:**

```
ALTER TABLE customers
  DROP COLUMN customer_name;
```

SQL ALTER table …

- **Renaming column of a table:**

```
ALTER TABLE customers
 RENAME COLUMN customer_name to cname;
```

- **Renaming table:**

```
ALTER TABLE customers
RENAME TO retailers;
```

SQL DROP table

- Used to remove a relation (base table) and its definition
- The relation can no longer be used in queries, updates, or any other commands since its description no longer exists
- Example:

```
DROP TABLE customers;
```

SQL TRUNCATE table

- It is used to delete complete data from an existing table.
- You can also use DROP TABLE command to delete complete table but it would remove complete table structure form the database and you would need to re-create this table once again if you wish to store some data.

Syntax:

TRUNCATE TABLE table_name;

Example:

TRUNCATE TABLE retailers;

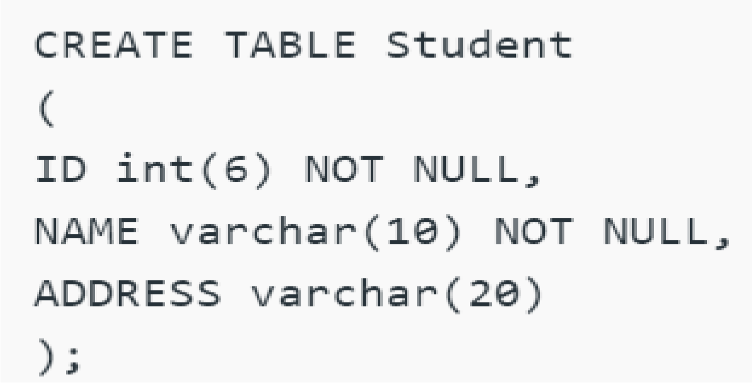
Integrity constraints

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- It ensure that the data insertion, updating, and other processes must be performed in such a way that data integrity is not affected.
- It is used to guard against accidental damage to the database.

Constraints

NOT NULL

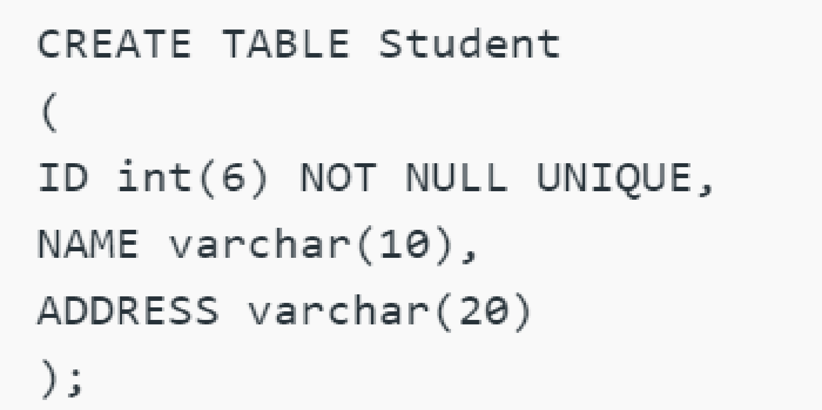
- Ensures that a column cannot have a NULL value. That is, you will be not allowed to insert a new row in the table without specifying any value to this field.



```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
ADDRESS varchar(20)
);
```

UNIQUE

- Ensures that all values in a column are different.



```
CREATE TABLE Student
(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20)
);
```

PRIMARY KEY constraint

- A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

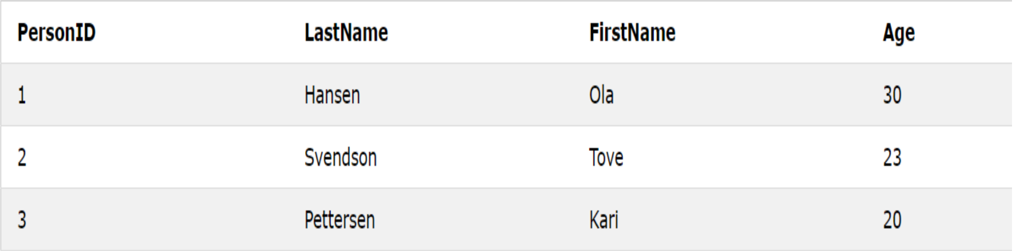
```
CREATE TABLE Persons (
    ID int NOT NULL PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);
```

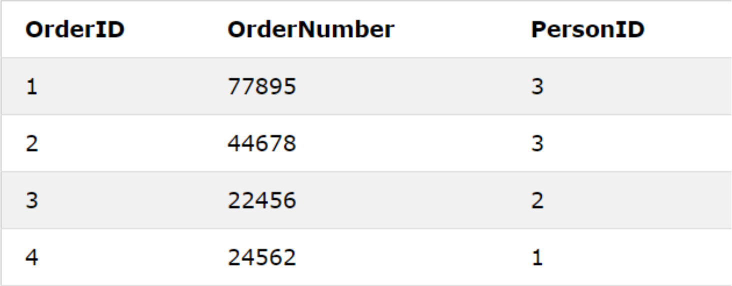
```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

FOREIGN KEY constraint

- Uniquely identifies a row/record in another table



PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20



OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

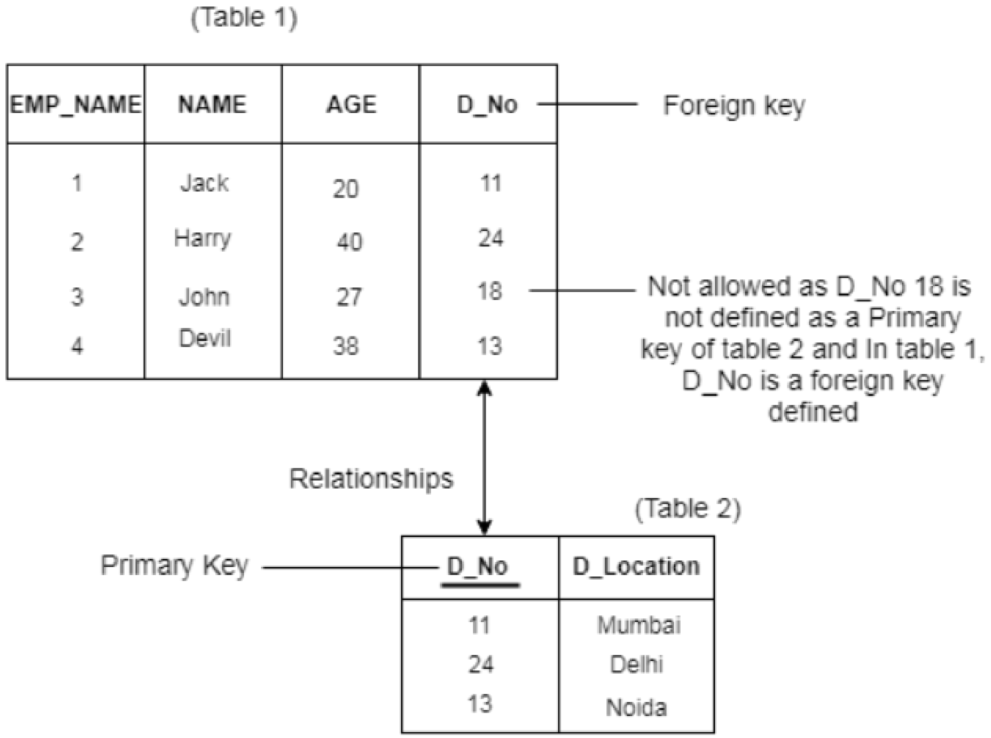
```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

```
ALTER TABLE Orders
ADD CONSTRAINT FK_PersonOrder
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

```
ALTER TABLE Orders
DROP CONSTRAINT FK_PersonOrder;
```

Referential Integrity

- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.



(Table 1)

EMP_NAME	NAME	AGE	D_No
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	Devil	38	13

Foreign key

Not allowed as D_No 18 is not defined as a Primary key of table 2 and In table 1, D_No is a foreign key defined

Relationships

(Table 2)

D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

Primary Key

CHECK constraint

- CHECK constraint is used to limit the value range that can be placed in a column.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18 AND City=‘Mumbai')
);
```

DEFAULT constraint

- The default value will be added to all new records if no other value is specified.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT ’Mumbai’
);
```

```
ALTER TABLE Persons
MODIFY City DEFAULT ’Pune’;

ALTER TABLE Persons
ALTER City DROP DEFAULT;
```

Retrieval Queries in SQL

- SQL has one basic statement for retrieving information from a database; the **SELECT** statement
- Basic form of the SQL SELECT statement is called a *mapping* or a SELECT-FROM-WHERE *block*

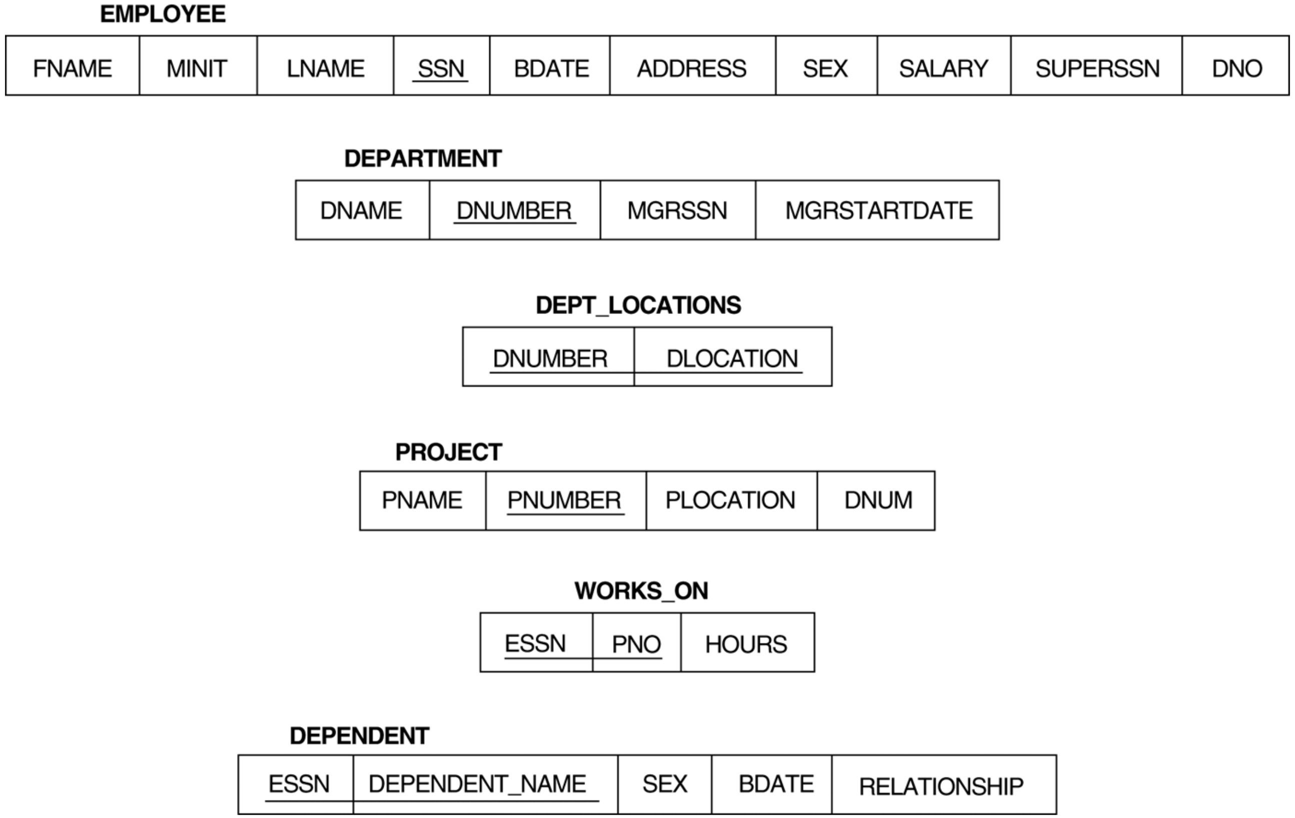
SELECT <attribute list>

FROM <table list>

WHERE <condition>

- <attribute list> is a list of attribute names whose values are to be retrieved by the query
- <table list> is a list of the relation names required to process the query
- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query

Relational Database Schema



EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

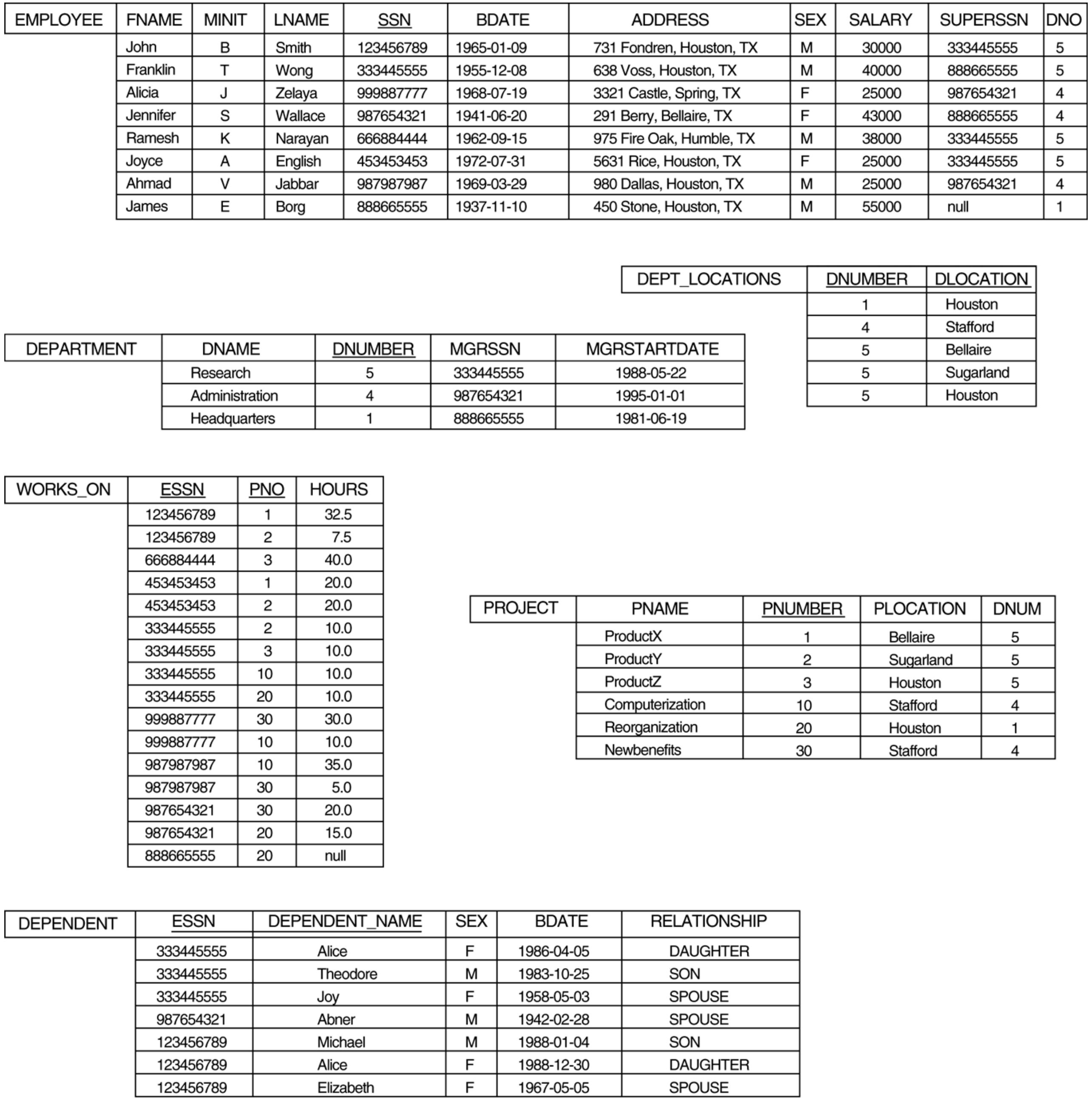
WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

Populated Database



EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

DEPT_LOCATIONS	DNUMBER	DLOCATION
	1	Houston
	4	Stafford
	5	Bellaire
	5	Sugarland
	5	Houston

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

WORKS_ON	ESSN	PNO	HOURS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40.0
	453453453	1	20.0
	453453453	2	20.0
	333445555	2	10.0
	333445555	3	10.0
	333445555	10	10.0
	333445555	20	10.0
	999887777	30	30.0
	999887777	10	10.0
	987987987	10	35.0
	987987987	30	5.0
	987654321	30	20.0
	987654321	20	15.0
	888665555	20	null

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	ProductX	1	Bellaire	5
	ProductY	2	Sugarland	5
	ProductZ	3	Houston	5
	Computerization	10	Stafford	4
	Reorganization	20	Houston	1
	Newbenefits	30	Stafford	4

DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
	333445555	Alice	F	1986-04-05	DAUGHTER
	333445555	Theodore	M	1983-10-25	SON
	333445555	Joy	F	1958-05-03	SPOUSE
	987654321	Abner	M	1942-02-28	SPOUSE
	123456789	Michael	M	1988-01-04	SON
	123456789	Alice	F	1988-12-30	DAUGHTER
	123456789	Elizabeth	F	1967-05-05	SPOUSE

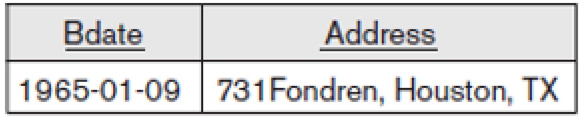
Simple SQL Queries

- Basic SQL queries correspond to using the following operations of the relational algebra:
  - SELECT
  - PROJECT
  - JOIN
- All subsequent examples use the COMPANY database

Simple SQL Queries (contd.)

Retrieve the birthdate and address of the employee whose name is 'John B. Smith'.

```
Q0:  SELECT    BDATE, ADDRESS
     FROM   EMPLOYEE
     WHERE   FNAME='John' AND MINIT=‘B’ AND LNAME='Smith’
```



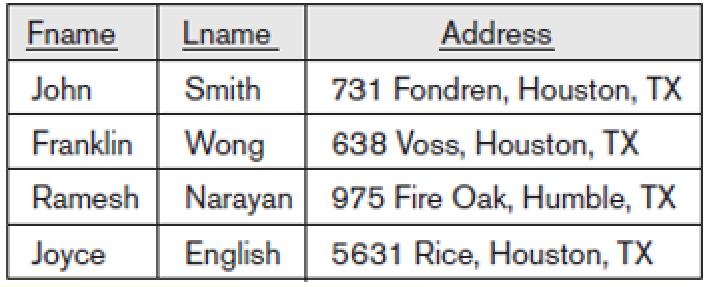
Bdate	Address
1965-01-09	731Fondren, Houston, TX

- Similar to a SELECT-PROJECT pair of relational algebra operations:
  - The SELECT-clause specifies the projection attributes and the WHERE-clause specifies the selection condition

Simple SQL Queries (contd.)

Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:  SELECT     FNAME, LNAME, ADDRESS
     FROM       EMPLOYEE, DEPARTMENT
     WHERE      DNAME='Research' AND DNUMBER=DNO
```

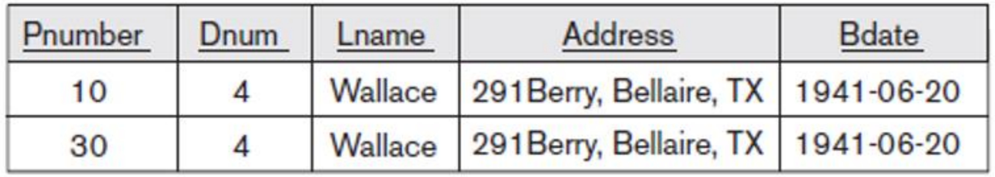


Fname	Lname	Address
John	Smith	731 Fondren, Houston, TX
Franklin	Wong	638 Voss, Houston, TX
Ramesh	Narayan	975 Fire Oak, Humble, TX
Joyce	English	5631 Rice, Houston, TX

- Similar to a SELECT-PROJECT-JOIN sequence of relational algebra operations
- (DNAME='Research') is a selection condition (corresponds to a SELECT operation in relational algebra)
- (DNUMBER=DNO) is a join condition (corresponds to a JOIN operation in relational algebra)

Simple SQL Queries (contd.)

For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birthdate.



Pnumber	Dnum	Lname	Address	Bdate
10	4	Wallace	291Berry, Bellaire, TX	1941-06-20
30	4	Wallace	291Berry, Bellaire, TX	1941-06-20

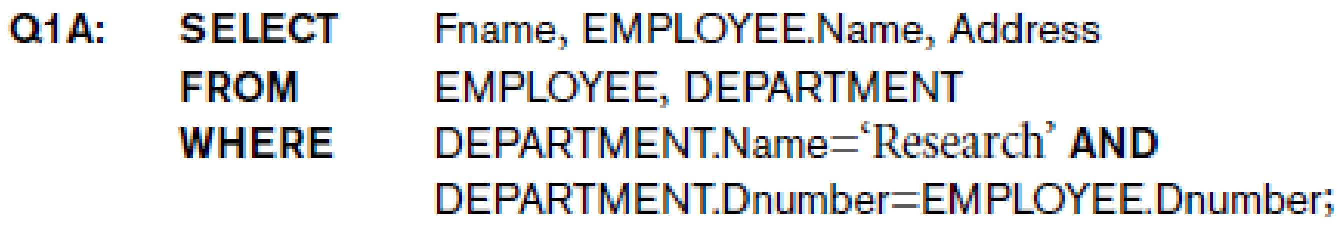
```
Q2: SELECT  PNUMBER, DNUM, LNAME, BDATE, ADDRESS
            FROM PROJECT, DEPARTMENT, EMPLOYEE
            WHERE DNUM=DNUMBER AND MGRSSN=SSN AND PLOCATION='Stafford'
```

- In Q2, there are two join conditions
- The join condition DNUM=DNUMBER relates a project to its controlling department
- The join condition MGRSSN=SSN relates the controlling department to the employee who manages that department

Ambiguous Attribute Names

Same name can be used for two (or more) attributes

- As long as the attributes are in different relations
- Must **qualify** the attribute name with the relation name to prevent ambiguity



```
Q1A:   SELECT   Fname, EMPLOYEE.Name, Address
       FROM     EMPLOYEE, DEPARTMENT
       WHERE    DEPARTMENT.Name='Research' AND
                DEPARTMENT.Dnumber=EMPLOYEE.Dnumber;
```

UNSPECIFIED WHERE-clause

- A *missing WHERE-clause* indicates no condition; hence, all tuples of the relations in the FROM-clause are selected
  - This is equivalent to the condition WHERE TRUE

Retrieve the SSN values for all employees.

```
SELECT SSN
     FROM EMPLOYEE
```

- If more than one relation is specified in the FROM-clause *and* there is no join condition, then the *CARTESIAN PRODUCT* of tuples is selected

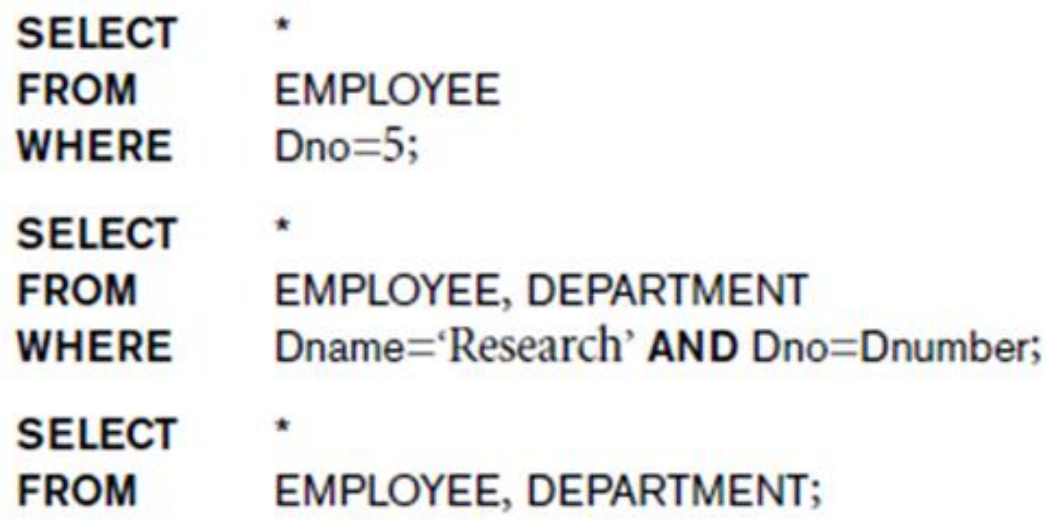
ALIASES (contd.)

- Aliasing can also be used in any SQL query for convenience
- Can also use the AS keyword to specify aliases

```
SELECT    E.FNAME, E.LNAME, S.FNAME, S.LNAME
          FROM   EMPLOYEE AS E, EMPLOYEE AS S
          WHERE E.SUPERSSN=S.SSN
```

Use of the Asterisk

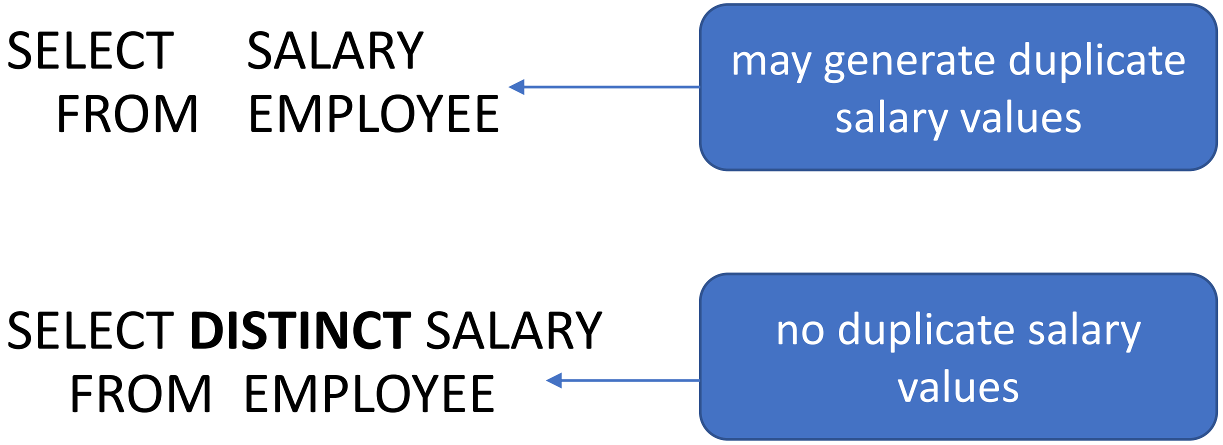
- Specify an asterisk (*)
    - Retrieve all the attribute values of the selected tuples



```
SELECT    *
FROM      EMPLOYEE
WHERE     Dno=5;

SELECT    *
FROM      EMPLOYEE, DEPARTMENT
WHERE     Dname='Research' AND Dno=Dnumber;

SELECT    *
FROM      EMPLOYEE, DEPARTMENT;
```

USE OF DISTINCT

- SQL does not treat a relation as a set; duplicate tuples can appear
- To eliminate duplicate tuples in a query result, the keyword **DISTINCT** is used



```
SELECT    SALARY
  FROM    EMPLOYEE
```

may generate duplicate salary values

```
SELECT DISTINCT SALARY
   FROM  EMPLOYEE
```

no duplicate salary values

Data Manipulation Language(DML)

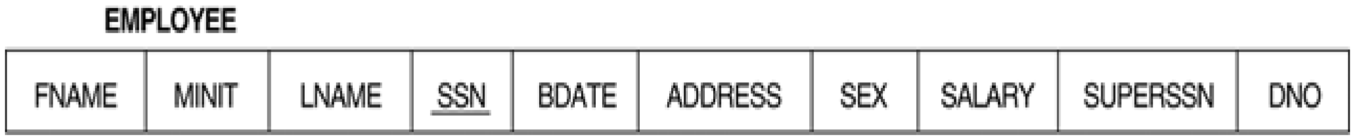
- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback
- Here are some commands that come under DML:
- INSERT
- UPDATE
- DELETE

INSERT

- In its simplest form, it is used to add one or more tuples to a relation
- Attribute values should be listed in the same order as the attributes were specified in the **CREATE TABLE** command

Example:

```
    U1: INSERT INTO  EMPLOYEE
          VALUES ('Richard','K','Marini', '653298653', '30-DEC-52’,
        '98 Oak Forest,Katy,TX', 'M’, 37000,'987654321', 4 )
```



EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

INSERT (contd.)

- An alternate form of INSERT specifies explicitly the attribute names that correspond to the values in the new tuple
    - Attributes with NULL values can be left out
- Example: Insert a tuple for a new EMPLOYEE for whom we only know the FNAME, LNAME, and SSN attributes.

```
U1A:   INSERT INTO EMPLOYEE (FNAME, LNAME, SSN)
          VALUES ('Richard', 'Marini', '653298653')
```

INSERT (contd.)

- Example: Suppose we want to create a temporary table that has the name, number of employees, and total salaries for each department.
  - A table DEPTS_INFO is created by U3A, and is loaded with the summary information retrieved from the database by the query in U3B.

```
U3A:CREATE TABLE  DEPTS_INFO
                          (DEPT_NAME          VARCHAR(10),
                           NO_OF_EMPS         INTEGER,
                           TOTAL_SAL          INTEGER);


U3B:INSERT INTO DEPTS_INFO (DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)
               SELECT DNAME, COUNT (*), SUM (SALARY)
               FROM  DEPARTMENT, EMPLOYEE
               WHERE    DNUMBER=DNO
               GROUP BY DNAME ;
```

INSERT (contd.)

- Note: The DEPTS_INFO table may not be up-to-date if we change the tuples in either the DEPARTMENT or the EMPLOYEE relations *after* issuing U3B. We have to create a view (see later) to keep such a table up to date.

DELETE

- Removes tuples from a relation
  - Includes a WHERE-clause to select the tuples to be deleted
  - A missing WHERE-clause specifies that *all tuples* in the relation are to be deleted; the table then becomes an empty table
  - The number of tuples deleted depends on the number of tuples in the relation that satisfy the WHERE-clause

DELETE (contd.)

- Examples:

```
U4A:         DELETE FROM EMPLOYEE
             WHERE LNAME='Brown’

U4B:         DELETE FROM EMPLOYEE
             WHERE SSN='123456789’

U4C:         DELETE FROM EMPLOYEE
             WHERE DNO  IN (SELECT DNUMBER FROM   DEPARTMENT
                              WHERE DNAME='Research')

U4D:         DELETE FROM EMPLOYEE
```

UPDATE

- Used to modify attribute values of one or more selected tuples
- A **WHERE-clause** selects the tuples to be modified
- An additional **SET-clause** specifies the attributes to be modified and their new values
- Each command modifies tuples *in the same relation*
- Referential integrity should be enforced

UPDATE (contd.)

- Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively.

```
U5:  UPDATE PROJECT
                SET PLOCATION = 'Bellaire', DNUM = 5
                WHERE PNUMBER=10
```

UPDATE (contd.)

- Example: Give all employees in the 'Research' department a 10% raise in salary.

```
U6: UPDATE  EMPLOYEE
    SET SALARY = SALARY *1.1
    WHERE   DNO  IN (SELECT DNUMBER
                     FROM DEPARTMENT
                     WHERE DNAME='Research')
```

- In this request, the modified SALARY value depends on the original SALARY value in each tuple
  - The reference to the SALARY attribute on the right of = refers to the old SALARY value before modification
  - The reference to the SALARY attribute on the left of = refers to the new SALARY value after modification

Data Control Language(DCL)

- Data Control Language (DCL) helps users to retrieve and modify the data stored in the database with some specified queries.
- DCL commands are used to grant and take back authority from any database user.
- Here are some commands that come under DCL:
- Grant
- Revoke

GRANT

- SQL Grant command is specifically used to provide privileges to database objects for an user.
- This command also allows users to grant permissions for other users too.

Syntax:

GRANT privilege_name on object_name

to user_name

Example:

GRANT INSERT,SELECT on Department to U1

GRANT ALL PRIVELEGES on Employee to U2

REVOKE

- Revoke command withdraw user privileges on database objects if any granted. It does operations opposite to the Grant command. When a privilege is revoked from a particular user U, then the privileges granted to all other users by user U will be revoked.

Syntax:

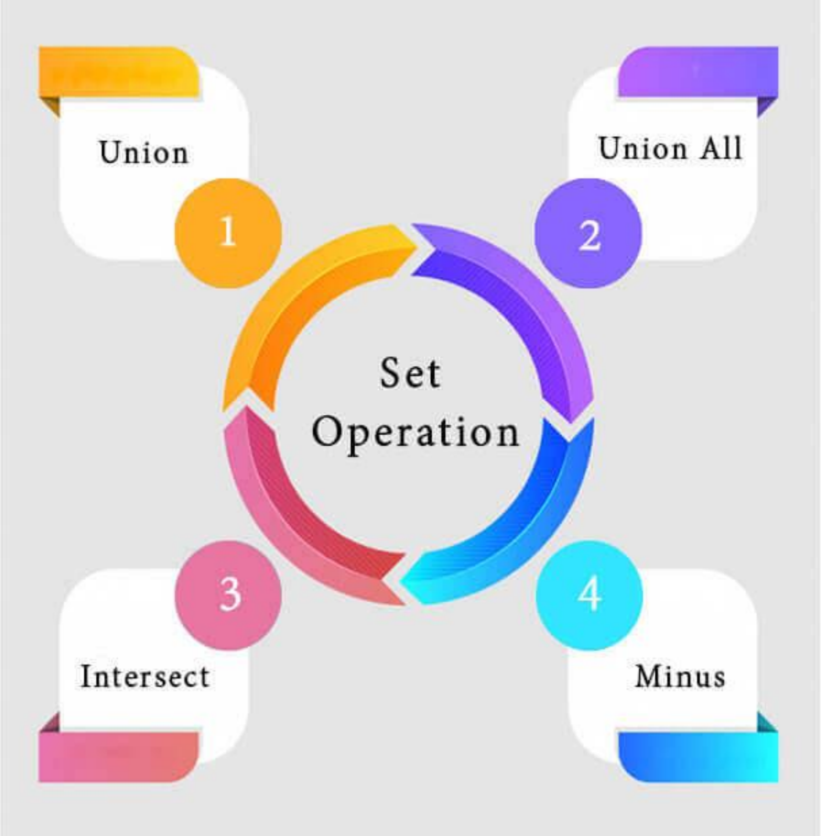
REVOKE privilege_name on object_name from user_name

Example:

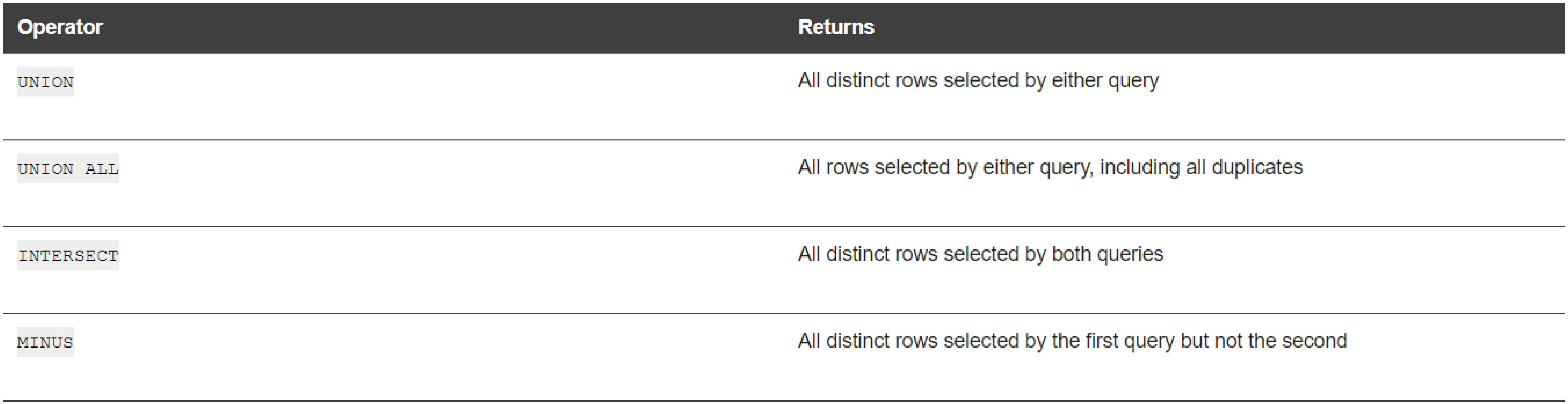
REVOKE INSERT on Department from U1

Set operations

- Set operators combine the results of two component queries into a single result.
- Queries containing set operators are called compound queries



Set operators(contd..)



Operator	Returns
`UNION`	All distinct rows selected by either query
`UNION ALL`	All rows selected by either query, including all duplicates
`INTERSECT`	All distinct rows selected by both queries
`MINUS`	All distinct rows selected by the first query but not the second

UNION

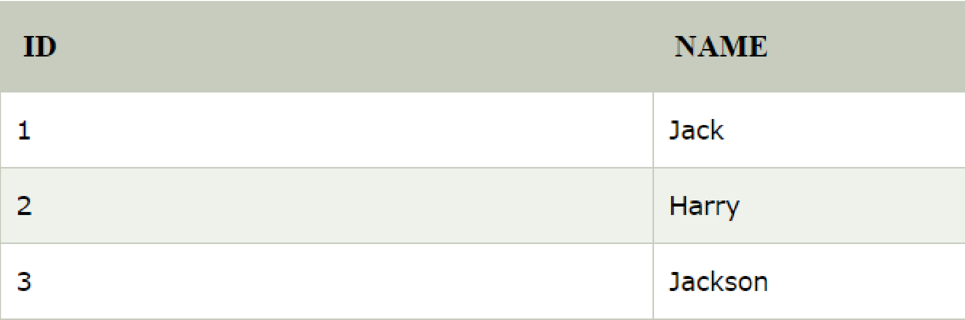
- The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
- In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
- The union operation eliminates the duplicate rows from its resultset.

Syntax:

```
SELECT column_name FROM table1

UNION

SELECT column_name FROM table2;
```

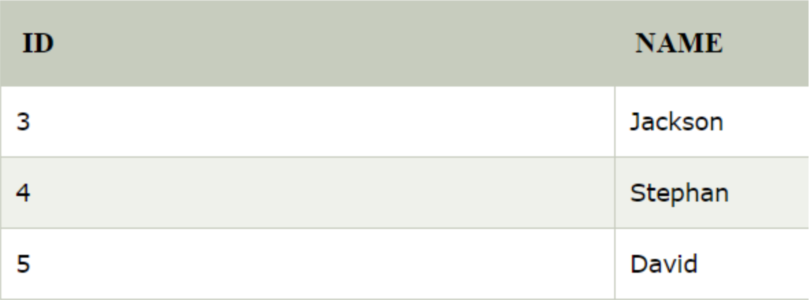
UNION(contd..)

The First table



ID	NAME
1	Jack
2	Harry
3	Jackson

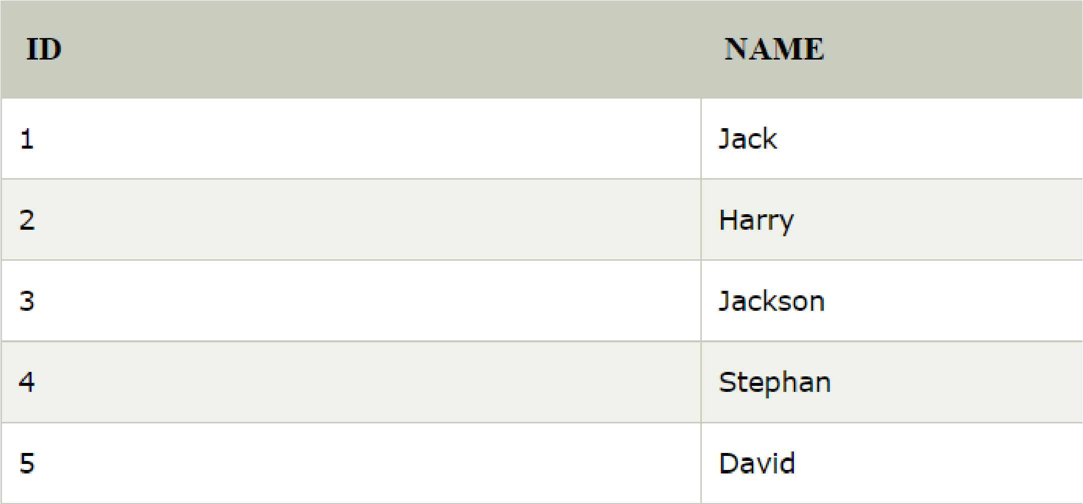
The Second table



ID	NAME
3	Jackson
4	Stephan
5	David

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

The resultset table will look like:



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

UNION ALL

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

Syntax:

SELECT column_name FROM table1

UNION ALL

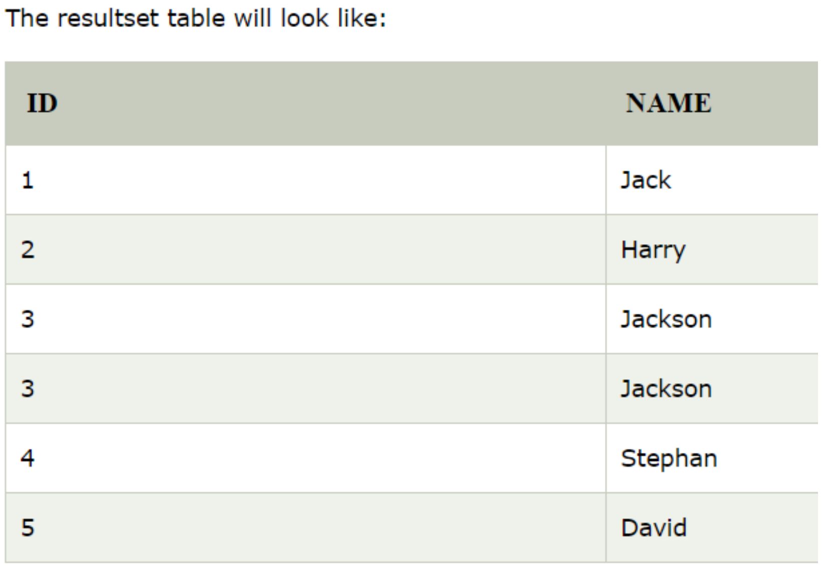
SELECT column_name FROM table2;

Example:

SELECT * FROM First

UNION ALL

SELECT * FROM Second



The resultset table will look like:

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

INTERSECT

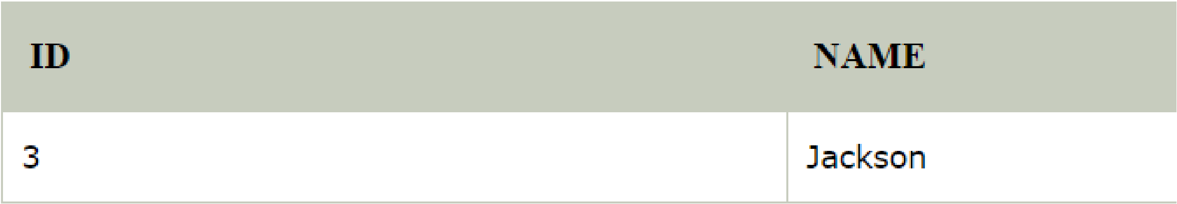
- The Intersect operation returns the common rows from both the SELECT statements.

Syntax:

```
SELECT column_name FROM table1
INTERSECT
SELECT column_name FROM table2;
```

Example:

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



ID	NAME
3	Jackson

MINUS

- Minus operator is used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates.

Syntax:

SELECT column_name FROM table1

MINUS

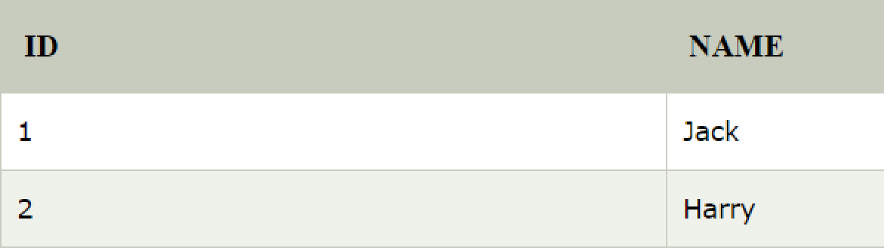
SELECT column_name FROM table2;

Example:

SELECT * FROM First

MINUS

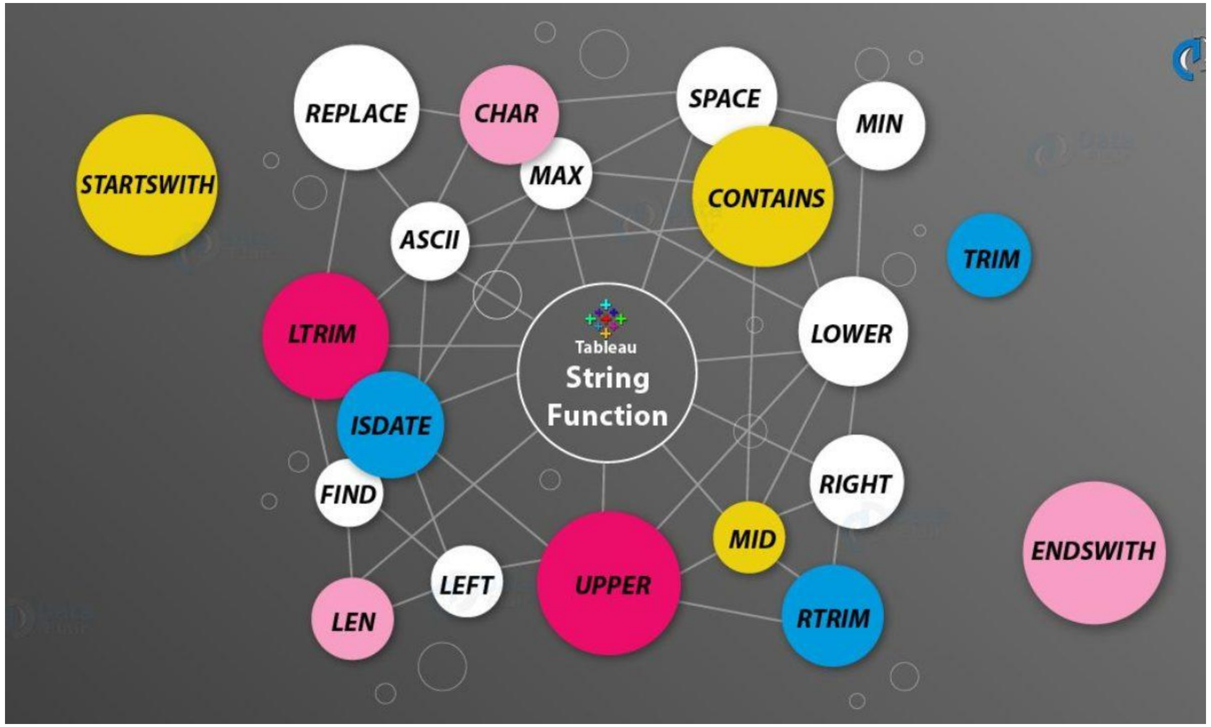
SELECT * FROM Second



ID	NAME
1	Jack
2	Harry

String operations

- String functions are used to perform an operation on input string and return an output string.
- Following are the string functions defined in SQL:



String functions (contd..)

- **ASCII():** This function is used to find the ASCII value of a character.

Syntax: SELECT ascii('t’) from dual;

Output: 116

- **CHAR_LENGTH():** This function is used to find the length of a word.

Syntax: SELECT char_length('Hello!’);

For oracle: SELECT length('Hello!’) from dual;

Output: 6

- **CONCAT_WS():** This function is used to add two words or strings with a symbol as concatenating symbol.

Syntax: SELECT CONCAT_WS('_', ‘SQL’,’Programming');

Output: SQL_Programming

Note: for oracle method is concat(), it will concatenate two strings

String functions (contd..)

LOWER(): This function is used to convert the upper case string into lower case.

Syntax: SELECT LOWER(‘SQL PROGRAMMING’);

Output: sql programming

Example:

SELECT LOWER(FNAME) AS LowercaseEmployeeName

FROM Employee;

String functions (contd..)

UPPER(): This function is used to make the string in upper case.

Syntax: SELECT UPPER('sql programming');

Output: SQL PROGRAMMING

Example:

SELECT UPPER(FNAME) AS UpperCaseEmployeeName

FROM Employee;

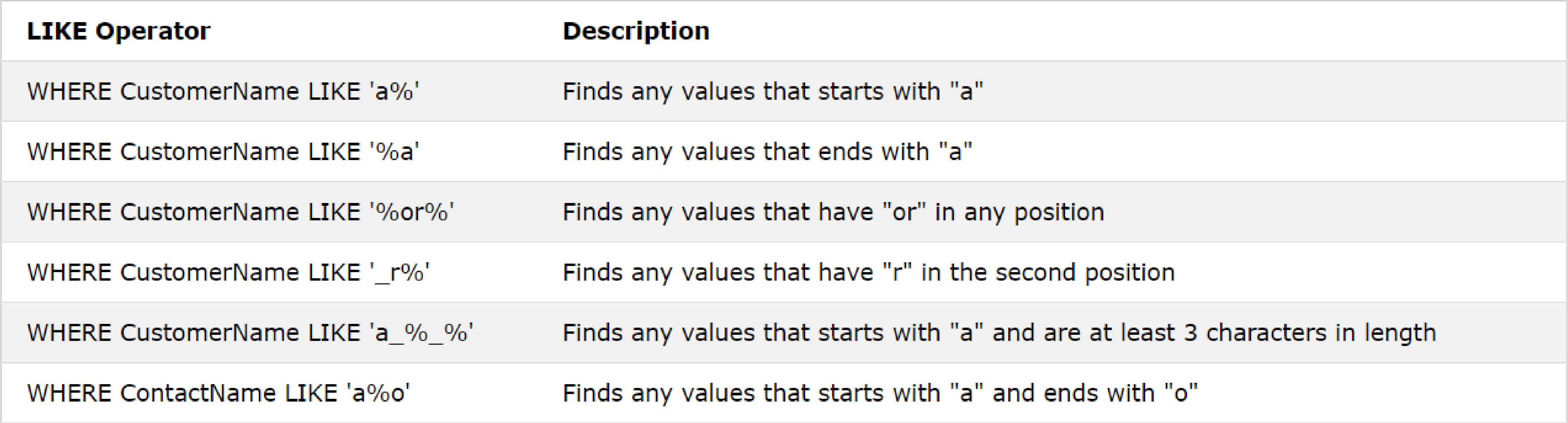
SQL LIKE operator

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- There are two wildcards often used in conjunction with the LIKE operator:
  - The percent sign (%) represents zero, one, or multiple characters
  - The underscore sign (_) represents one single character

Syntax:

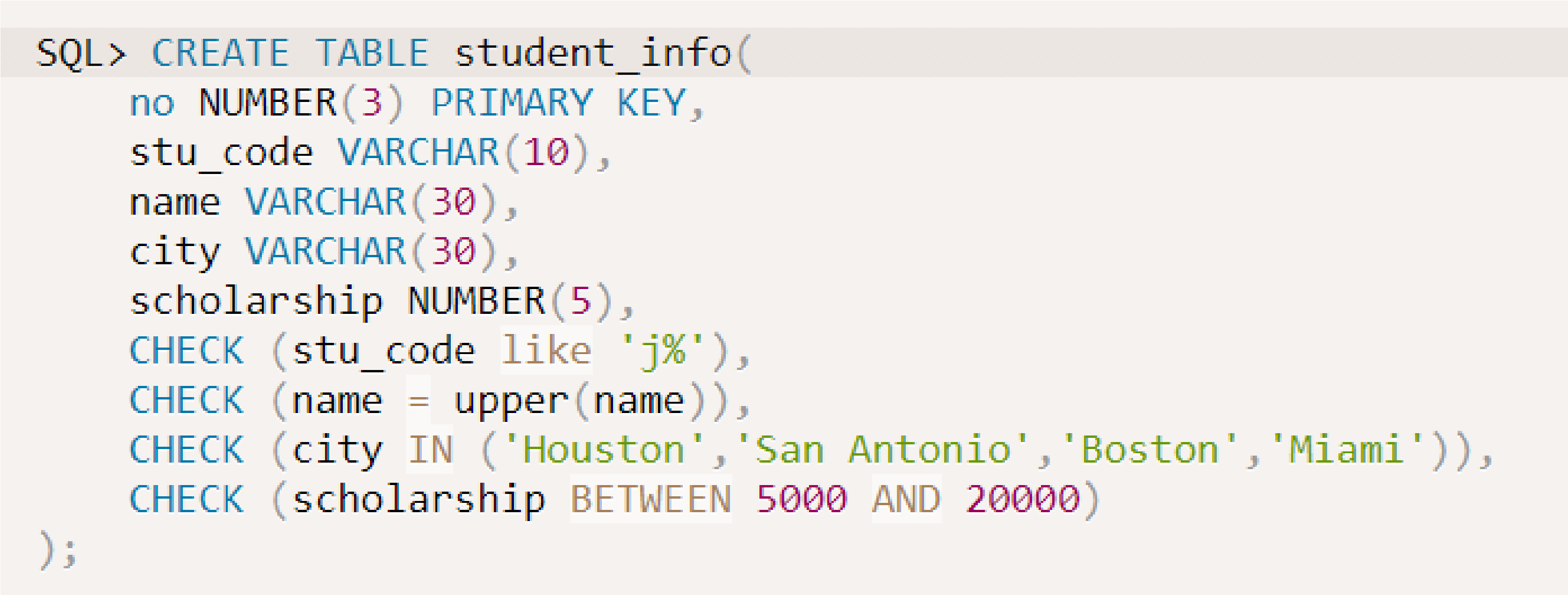
```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

SQL LIKE operator(contd..)



LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%_%'	Finds any values that starts with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

Example



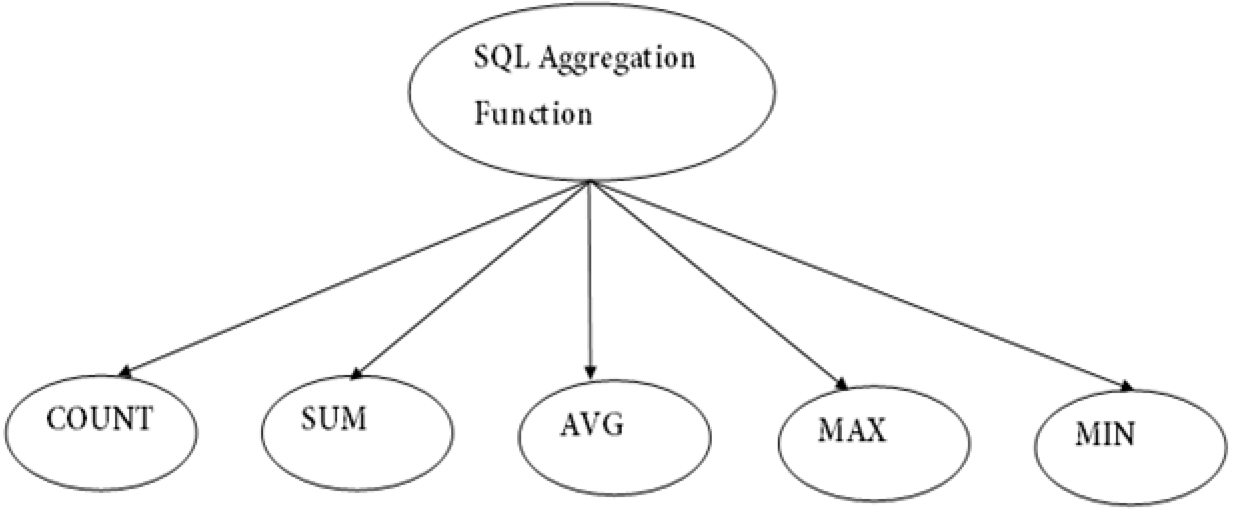
```
SQL> CREATE TABLE student_info(
    no NUMBER(3) PRIMARY KEY,
    stu_code VARCHAR(10),
    name VARCHAR(30),
    city VARCHAR(30),
    scholarship NUMBER(5),
    CHECK (stu_code like 'j%'),
    CHECK (name = upper(name)),
    CHECK (city IN ('Houston','San Antonio','Boston','Miami')),
    CHECK (scholarship BETWEEN 5000 AND 20000)
);
```

Oracle string functions link

https://docs.oracle.com/middleware/1221/biee/BIVUG/GUID-BBA975C7-B2C5-4C94-A007-28775680F6A5.htm#BILUG685

Aggregate functions

- Function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning



Aggregate functions(contd…)

COUNT()

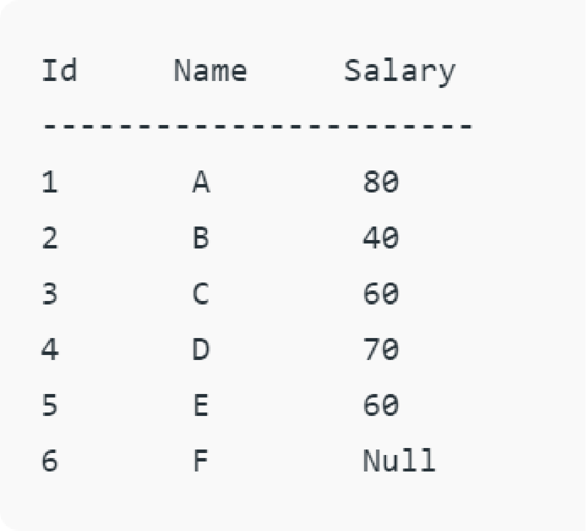
- It is used to Count the number of rows in a database table.
- It can work on both numeric and non-numeric data types.
- COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table.
- COUNT(*) considers duplicate and Null.

Syntax:

COUNT(*)

or

COUNT([ALL|DISTINCT] expression)



```
Id      Name      Salary
------------------------
1         A         80
2         B         40
3         C         60
4         D         70
5         E         60
6         F         Null
```

Example:

Select COUNT(*) from emp: Returns total number of records .i.e 6.

COUNT(salary): Return number of Non Null values over the column salary. i.e 5.

COUNT(Distinct Salary): Return number of distinct Non Null values over the column salary i.e. 4

Aggregate functions(contd…)

SUM()

- Sum function is used to calculate the sum of all selected columns.
- It works on numeric fields only.

Syntax:

SUM()

or

SUM([ALL|DISTINCT] expression)

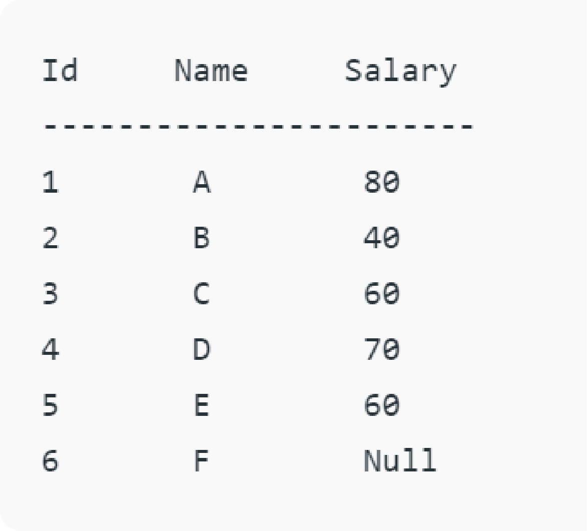
Example:

Select SUM(salary) as Sum from emp

Sum all Non Null values of Column salary i.e., 310

Select SUM(Distinct salary) from emp

Sum of all distinct Non-Null values i.e., 250.



```
Id      Name      Salary
-----------------------
1         A         80
2         B         40
3         C         60
4         D         70
5         E         60
6         F         Null
```

Aggregate functions(contd...)

- It is used to calculate the average value of the numeric type.
- It returns the average of all non-Null values.

Syntax:

AVG()

or

AVG([ALL|DISTINCT] expression)

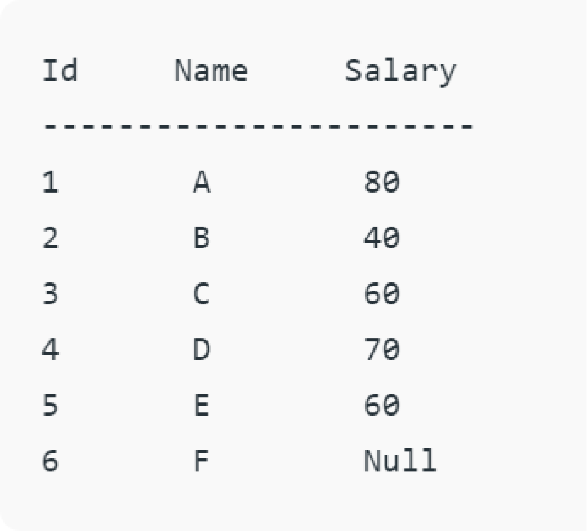
Example:

Select AVG(salary) from emp

= Sum(salary) / count(salary) = 310/5

Select AVG(Distinct salary) as Average from emp

= sum(Distinct salary) / Count(Distinct Salary) = 250/4



```
Id      Name      Salary
-----------------------
1         A         80
2         B         40
3         C         60
4         D         70
5         E         60
6         F         Null
```

Aggregate functions(contd...)

MAX()

- It is used to find the maximum value of a certain column.
- This function determines the largest value of all selected values of a column.

Syntax:

MAX()

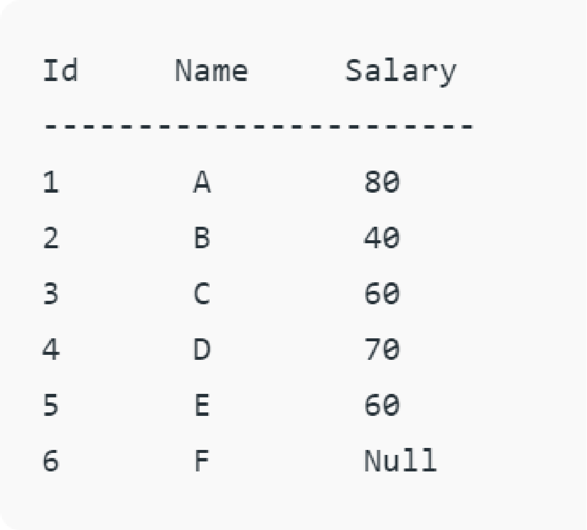
or

MAX([ALL|DISTINCT] expression)

Example:

SELECT MAX(Salary)

FROM emp;



```
Id      Name      Salary
------------------------
1         A         80
2         B         40
3         C         60
4         D         70
5         E         60
6         F         Null
```

Aggregate functions(contd...)

MIN()

- It is used to find the minimum value of a certain column.
- This function determines the smallest value of all selected values of a column.

Syntax:

MIN()

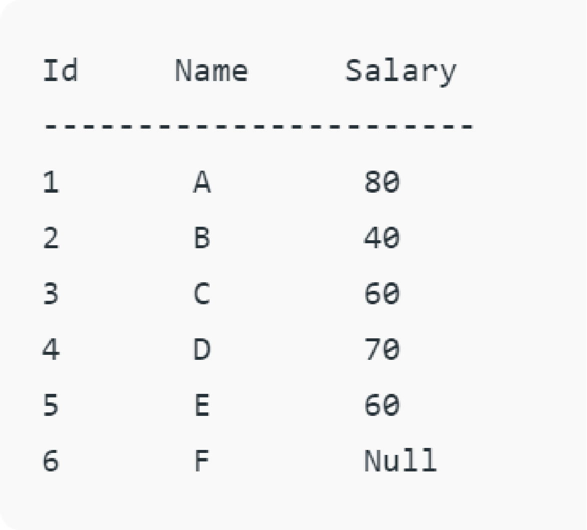
or

MIN([ALL|DISTINCT] expression)

Example:

SELECT MIN(Salary)

FROM emp;



```
Id      Name      Salary
-----------------------
1         A         80
2         B         40
3         C         60
4         D         70
5         E         60
6         F         Null
```

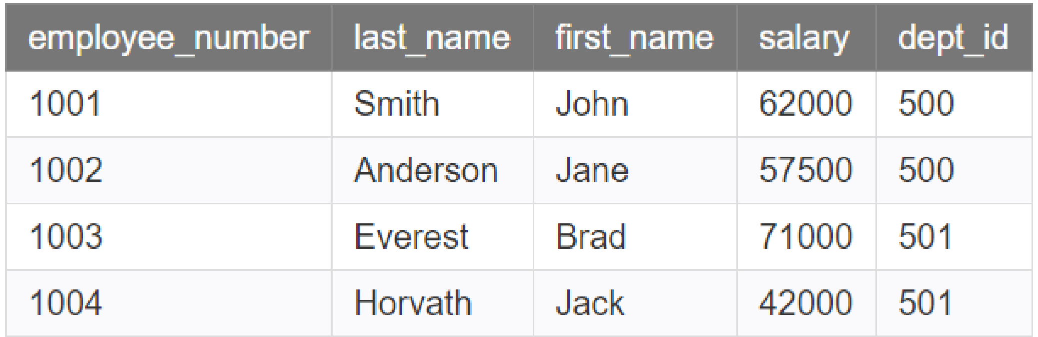
Aggregate functions with GROUP BY

- SQL has a GROUP BY-clause for specifying the grouping attributes, which must also appear in the SELECT-clause.
- In many cases, we want to apply the aggregate functions to subgroups of tuples in a relation.

Syntax:

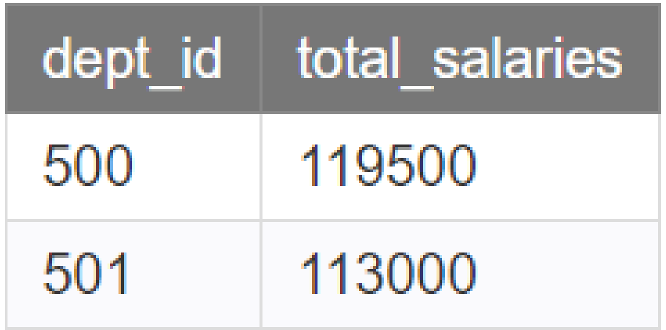
```
SELECT column_name(s), aggregate_function (aggregate_expression)

FROM table_name

WHERE condition

GROUP BY column_name(s)

[ORDER BY column_name(s) [ASC|DESC];]
```

Using GROUP BY with the SUM Function



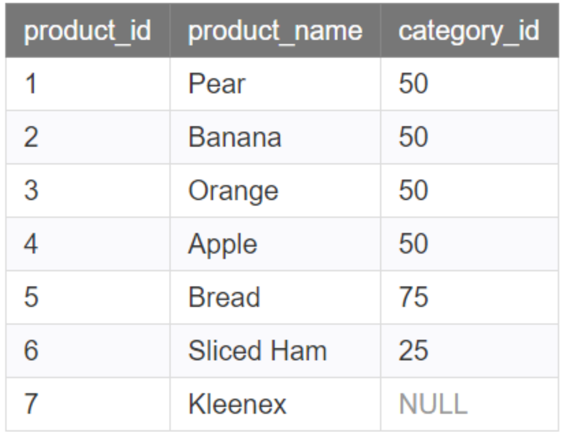
employee_number	last_name	first_name	salary	dept_id
1001	Smith	John	62000	500
1002	Anderson	Jane	57500	500
1003	Everest	Brad	71000	501
1004	Horvath	Jack	42000	501

```
SELECT dept_id, SUM(salary) AS total_salaries
FROM employees
GROUP BY dept_id;
```

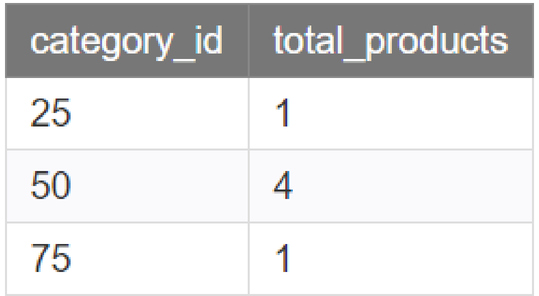


dept_id	total_salaries
500	119500
501	113000

Using GROUP BY with the COUNT Function



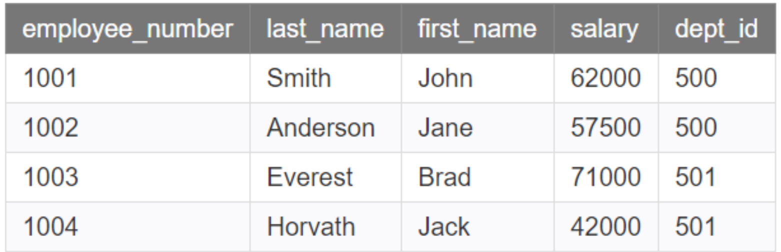
product_id	product_name	category_id
1	Pear	50
2	Banana	50
3	Orange	50
4	Apple	50
5	Bread	75
6	Sliced Ham	25
7	Kleenex	NULL



category_id	total_products
25	1
50	4
75	1

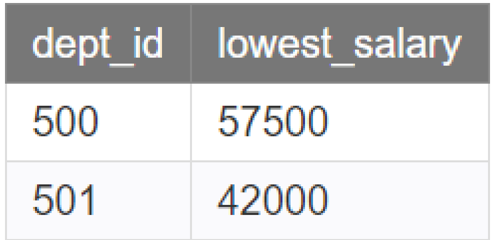
```
SELECT category_id, COUNT(*) AS total_products
FROM products
WHERE category_id IS NOT NULL
GROUP BY category_id
ORDER BY category_id;
```

Using GROUP BY with the MIN function



employee_number	last_name	first_name	salary	dept_id
1001	Smith	John	62000	500
1002	Anderson	Jane	57500	500
1003	Everest	Brad	71000	501
1004	Horvath	Jack	42000	501

```
SELECT dept_id, MIN(salary) AS lowest_salary
FROM employees
GROUP BY dept_id;
```



dept_id	lowest_salary
500	57500
501	42000

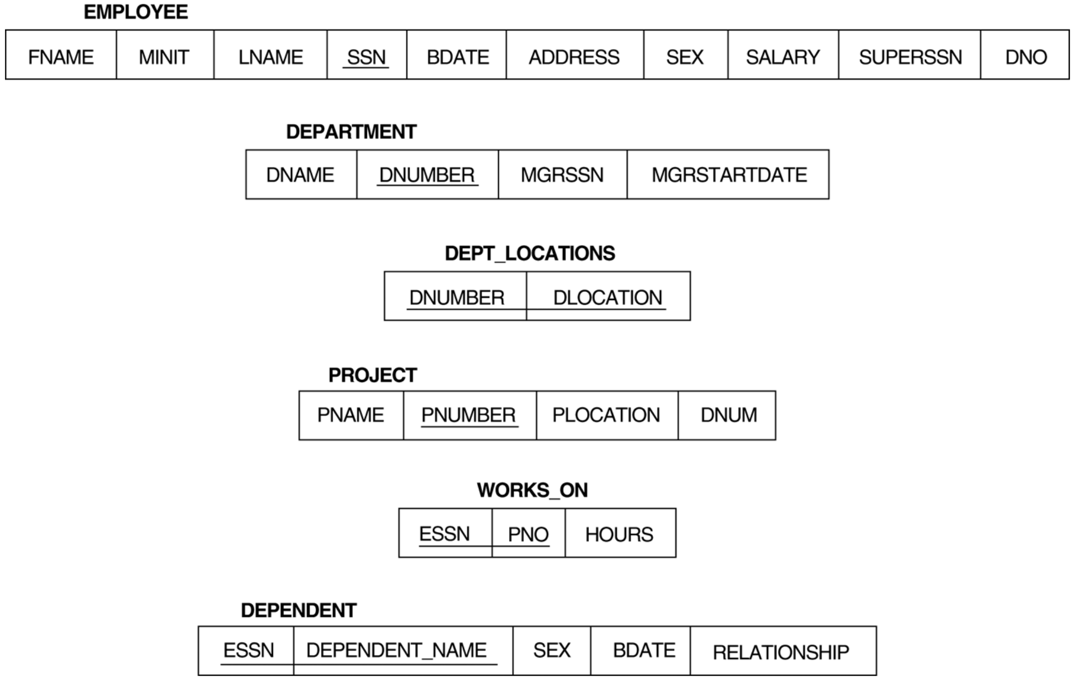
Queries

Q. For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
SELECT DNO, COUNT (*), AVG (SALARY)
FROM  EMPLOYEE
GROUP BY  DNO
```

Q. For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
SELECT PNUMBER, PNAME, COUNT (*)
FROM PROJECT, WORKS_ON
WHERE  PNUMBER=PNO
GROUP BY PNUMBER, PNAME
```



EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

THE HAVING-CLAUSE

- Sometimes we want to retrieve the values of these functions for only those *groups that satisfy certain conditions*
- The **HAVING**-clause is used for specifying a selection condition on groups (rather than on individual tuples)

Syntax:

```
SELECT column1, column2
FROM table1, table2
WHERE [ conditions ]
GROUP BY column1, column2
HAVING [ conditions ]
ORDER BY column1, column2
```

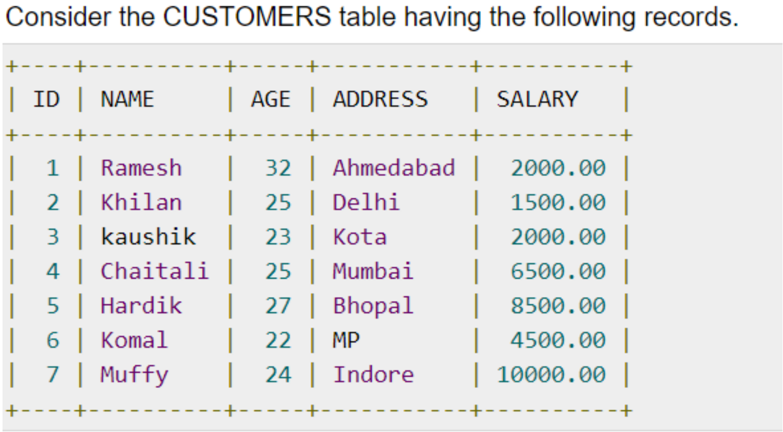
Example

SELECT ID, NAME, AGE, ADDRESS, SALARY

FROM CUSTOMERS

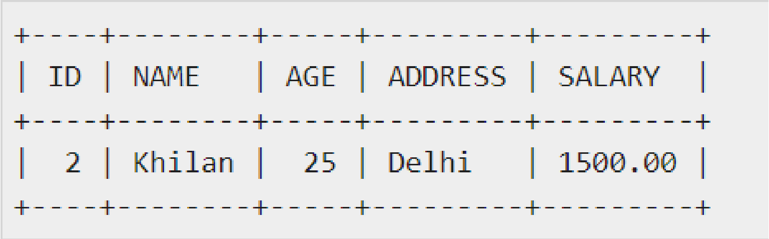
GROUP BY age

HAVING COUNT(age) >= 2;



Consider the CUSTOMERS table having the following records.

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

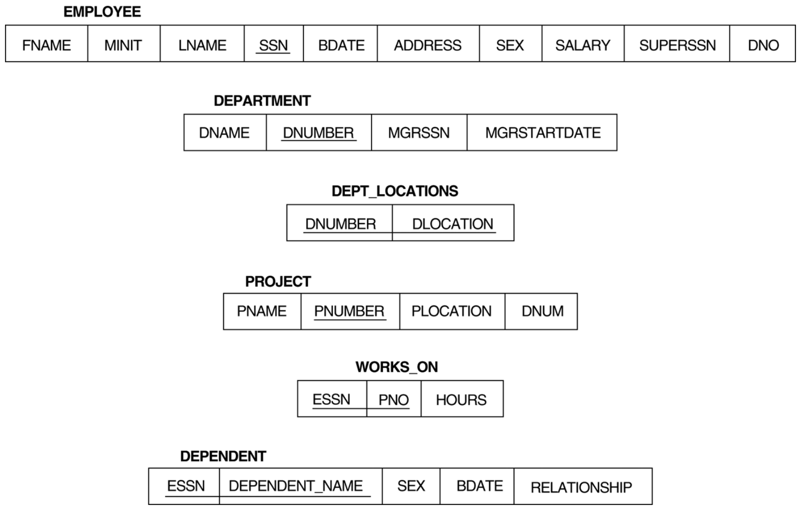


```
+----+--------+-----+---------+---------+
| ID | NAME   | AGE | ADDRESS | SALARY  |
+----+--------+-----+---------+---------+
|  2 | Khilan |  25 | Delhi   | 1500.00 |
+----+--------+-----+---------+---------+
```

Query

Q. For each project on which more than two employees work, retrieve the project number, project name, and the number of employees who work on that project.

```
SELECT PNUMBER, PNAME, COUNT(*)
FROM     PROJECT, WORKS_ON
WHERE PNUMBER=PNO
GROUP BY PNUMBER
HAVING COUNT (*) > 2
```



EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

Views in SQL

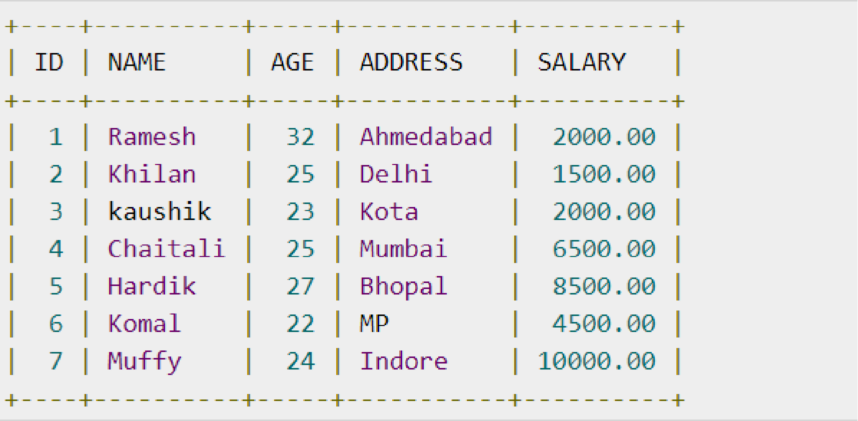
- It is a virtual table based on the result-set of an SQL statement.
- It contains rows and columns, just like a real table.
- Fields in a view are fields from one or more real tables in the database.
- Allows for limited update operations
- Allows full query operations

Syntax:

```
CREATE VIEW view_name AS

SELECT column1, column2.....

FROM table_name

WHERE [condition];
```

Creating view

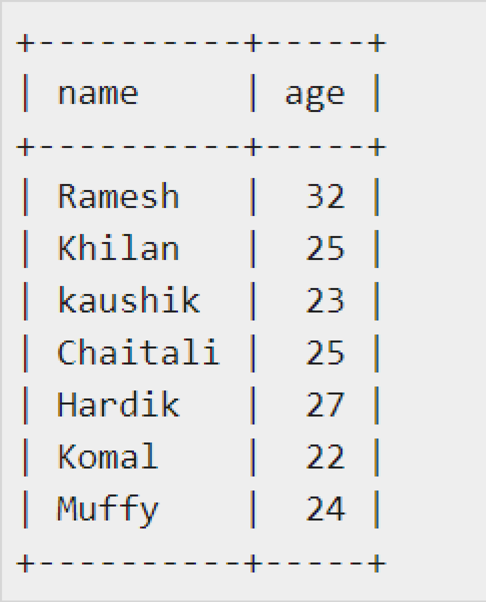
Consider the CUSTOMERS table having the following records −



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

```
CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM  CUSTOMERS;
```

```
SELECT * FROM CUSTOMERS_VIEW;
```



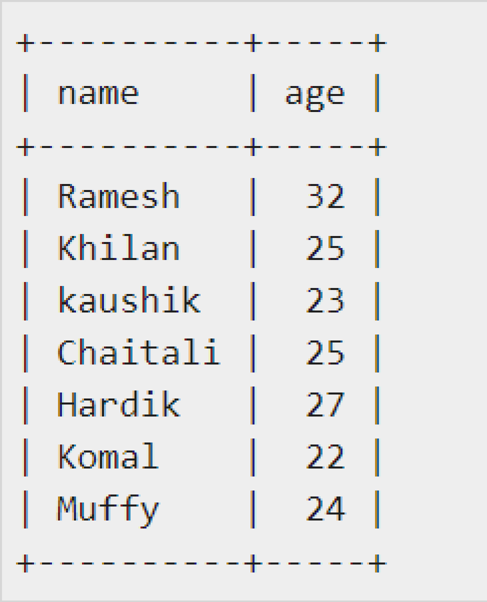
```
+----------+-----+
| name     | age |
+----------+-----+
| Ramesh   |  32 |
| Khilan   |  25 |
| kaushik  |  23 |
| Chaitali |  25 |
| Hardik   |  27 |
| Komal    |  22 |
| Muffy    |  24 |
+----------+-----+
```

Updating view

- There are certain conditions needed to be satisfied to update a view. If any one of these conditions is not met, then we will not be allowed to update the view.

1. The SELECT statement which is used to create the view should not include GROUP BY clause or ORDER BY clause.
2. The SELECT statement should not have the DISTNCT keyword.
3. The view should not be created using nested queries or complex queries.
4. The view should be created from a single table. If the view is created using multiple tables then we will not be allowed to update the view.
5. All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

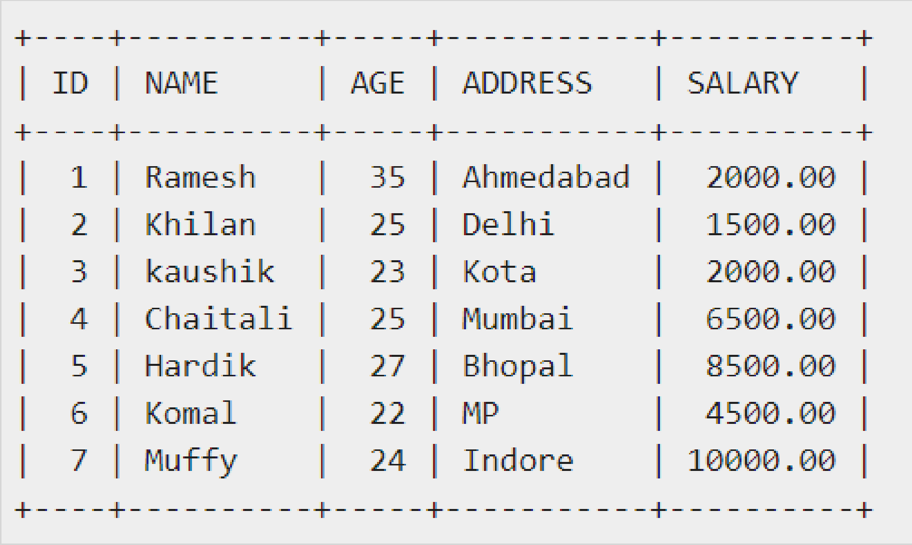
Updating view



```
+----------+-----+
| name     | age |
+----------+-----+
| Ramesh   |  32 |
| Khilan   |  25 |
| kaushik  |  23 |
| Chaitali |  25 |
| Hardik   |  27 |
| Komal    |  22 |
| Muffy    |  24 |
+----------+-----+
```

```
UPDATE CUSTOMERS_VIEW
   SET AGE = 35
   WHERE name = 'Ramesh';
```

```
SELECT * FROM CUSTOMERS;
```



```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Inserting rows in a view

- Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command.
- Here, we cannot insert rows in the CUSTOMERS_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in a similar way as you insert them in a table.

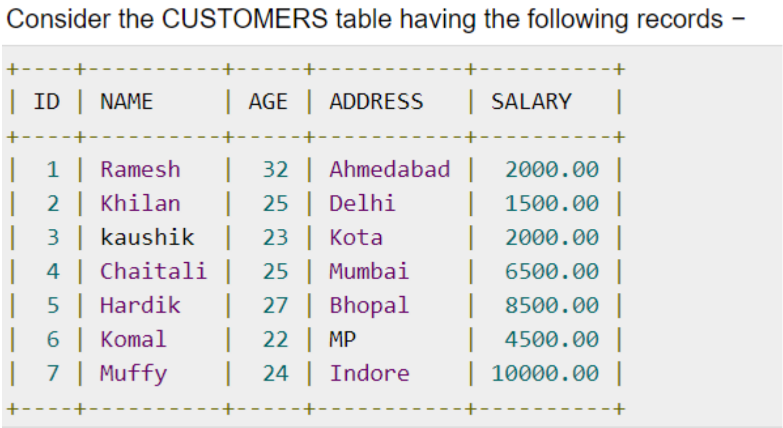
Deleting rows in a view

DELETE FROM CUSTOMERS_VIEW

WHERE age = 22;

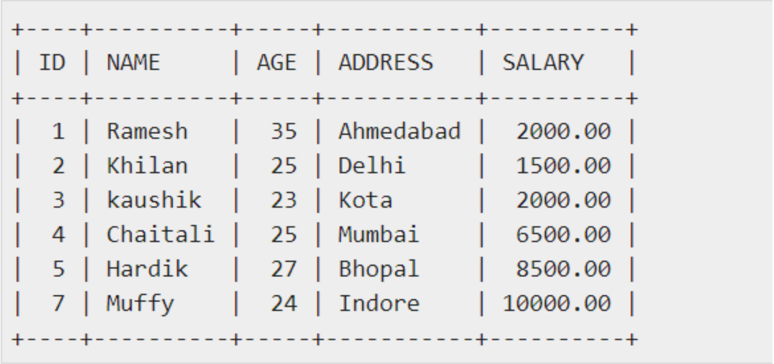
Dropping view

DROP VIEW CUSTOMERS_VIEW;



Consider the CUSTOMERS table having the following records –

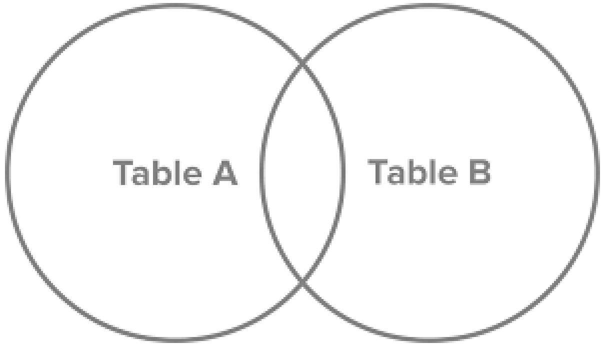
```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali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



```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

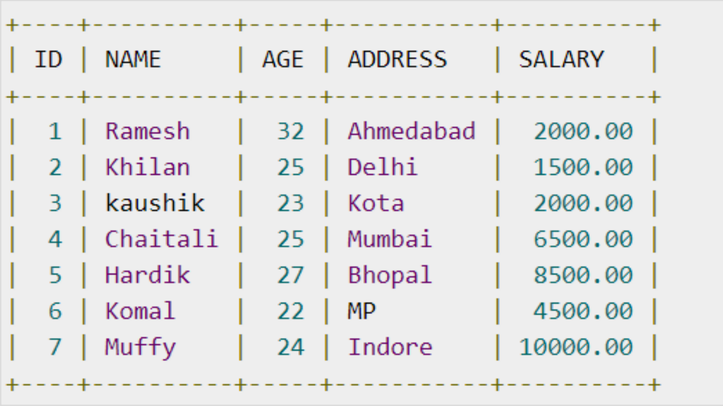
Joins in SQL

- It is used to combine records from two or more tables in a database.
- A JOIN is a means for combining fields from two tables by using values common to each.



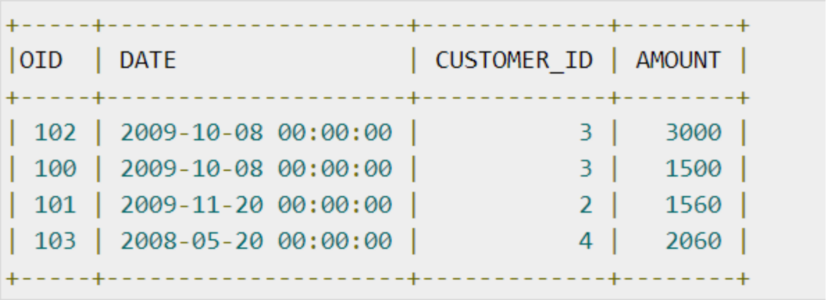
Equi Join Example

Table 1 – CUSTOMERS Table



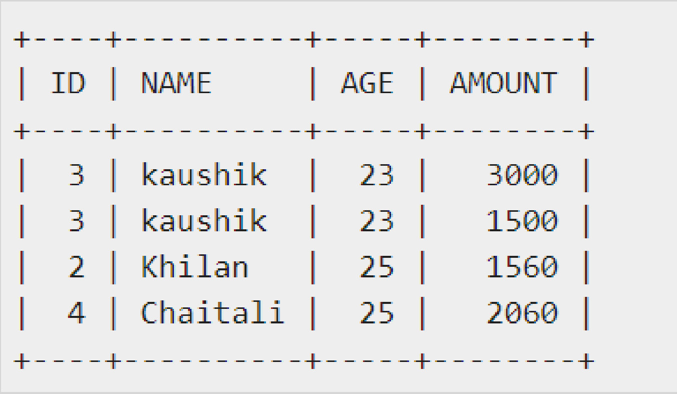
```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali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

Table 2 – ORDERS Table



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

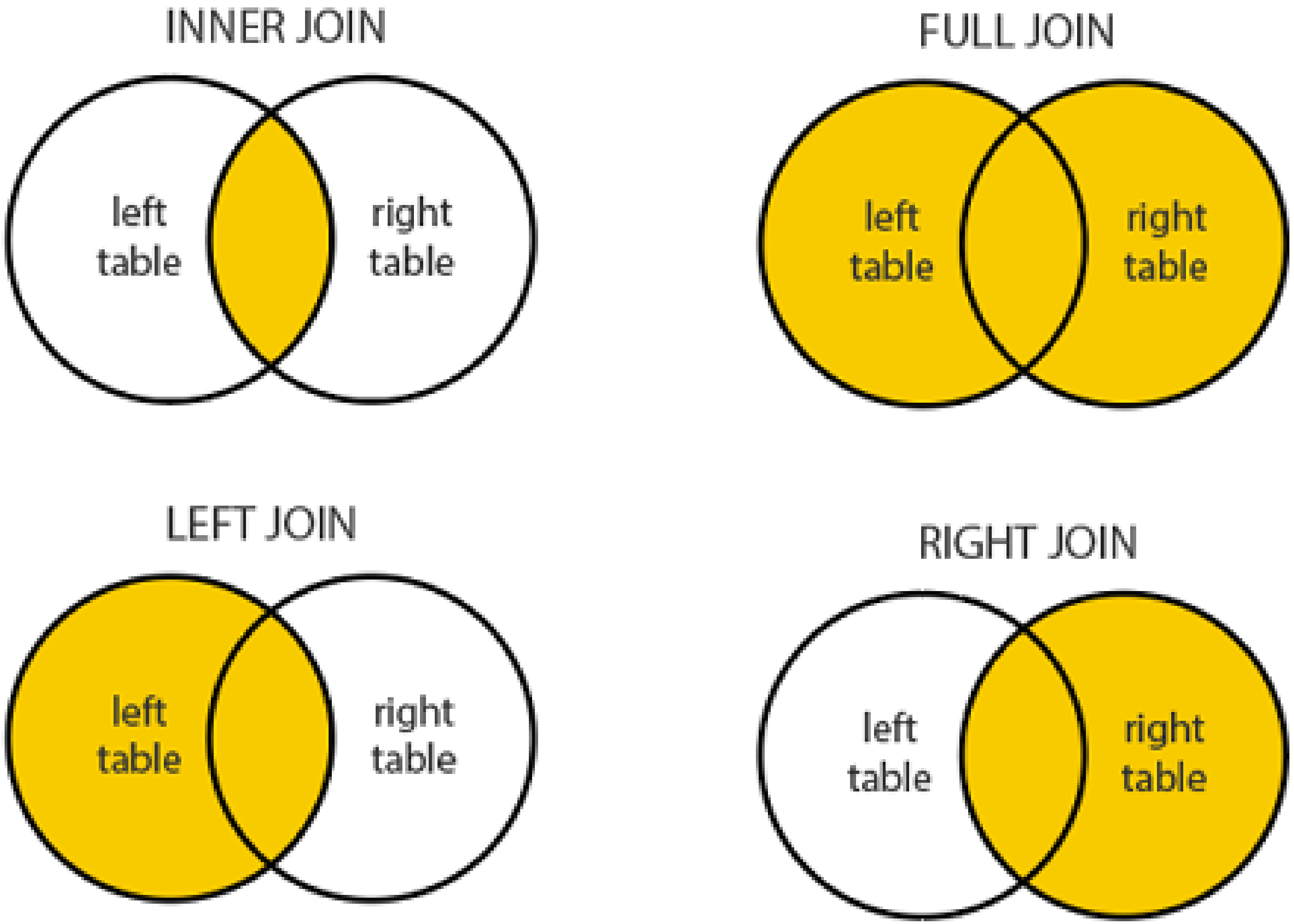
```
SELECT ID, NAME, AGE, AMOUNT
  FROM CUSTOMERS, ORDERS
  WHERE  CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```



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

Note: Several operators can be used to join tables, such as =,**Non-Equi Join**(<, >, <=, >=, !=, BETWEEN, LIKE, and NOT); they can all be used to join tables. However, the most common operator is the equal to symbol.

Types of Join



INNER JOIN

- Also referred as EQUI JOIN

Syntax:

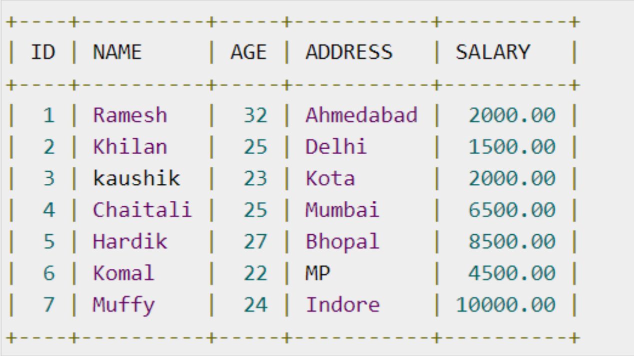
```
SELECT table1.column1, table2.column2...
FROM table1
INNER JOIN table2
ON table1.common_field = table2.common_field;
```

LEFT JOIN

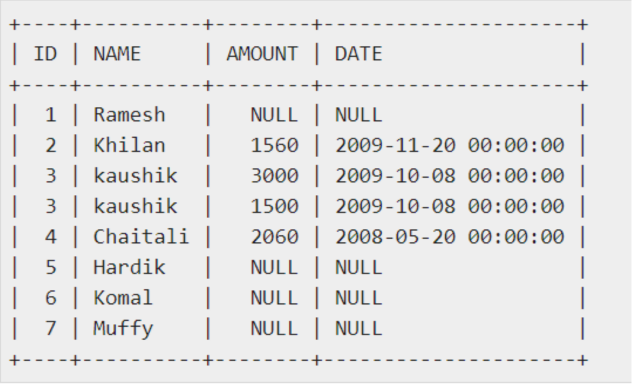
- Returns all rows from the left table, even if there are no matches in the right table.
- It returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

```
SELECT  ID, NAME, AMOUNT, DATE
  FROM CUSTOMERS
  LEFT JOIN ORDERS
  ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

Table 1 – CUSTOMERS Table

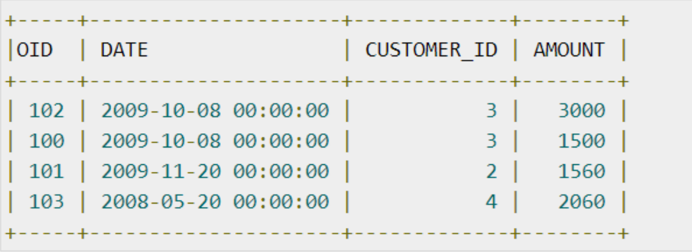


```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali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



```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
|  1 | Ramesh   |   NULL | NULL                |
|  2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|  3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|  3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|  4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|  5 | Hardik   |   NULL | NULL                |
|  6 | Komal    |   NULL | NULL                |
|  7 | Muffy    |   NULL | NULL                |
+----+----------+--------+---------------------+
```

Table 2 – ORDERS Table



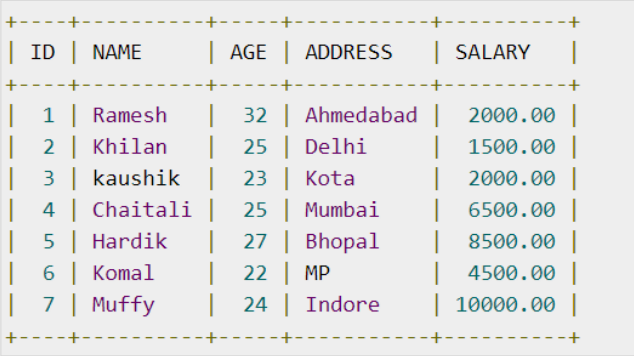
```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |  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

RIGHT JOIN

- Returns all rows from the right table, even if there are no matches in the right table.
- It returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

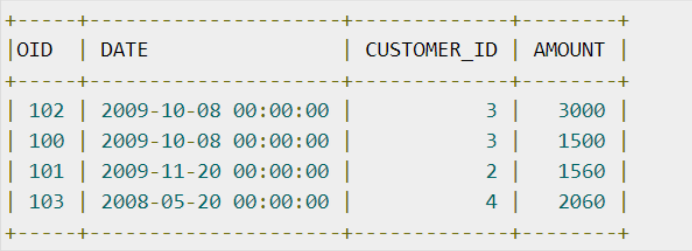
```
SELECT  ID, NAME, AMOUNT, DATE
  FROM CUSTOMERS
  RIGHT JOIN ORDERS
  ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

Table 1 – CUSTOMERS Table

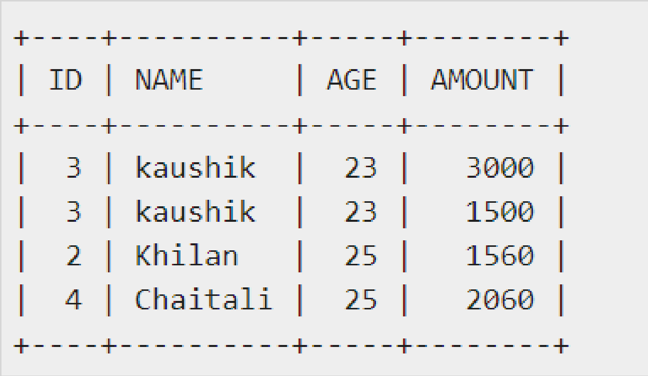


```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali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

Table 2 – ORDERS Table



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

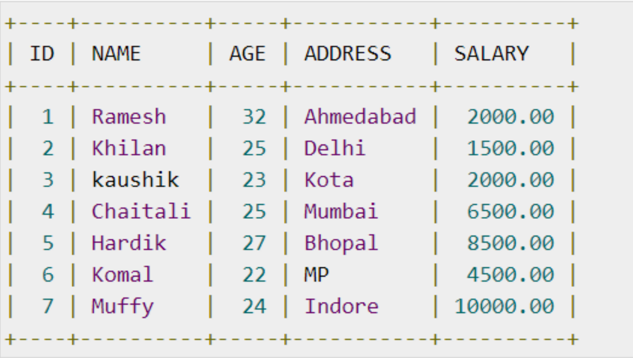


```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
|  3 | kaushik  |  23 |   3000 |
|  3 | kaushik  |  23 |   1500 |
|  2 | Khilan   |  25 |   1560 |
|  4 | Chaitali |  25 |   2060 |
+----+----------+-----+--------+
```

FULL JOIN

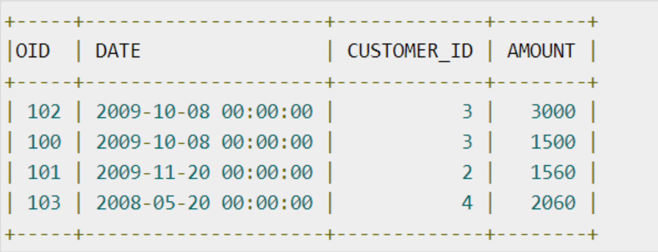
- Combines the results of both left and right outer joins.
- The joined table will contain all records from both the tables and fill in NULLs for missing matches on either side.

Table 1 – CUSTOMERS Table

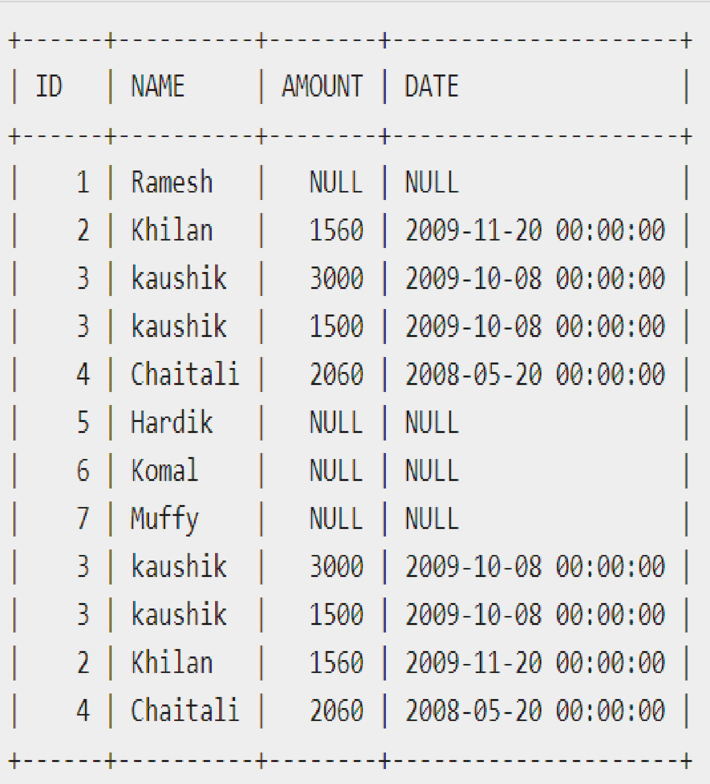


```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali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

Table 2 – ORDERS Table



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



```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
|    1 | Ramesh   |   NULL | NULL                |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|    5 | Hardik   |   NULL | NULL                |
|    6 | Komal    |   NULL | NULL                |
|    7 | Muffy    |   NULL | NULL                |
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

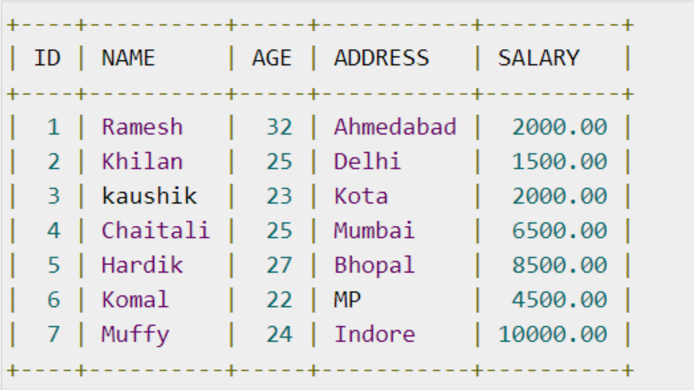
```
SELECT  ID, NAME, AMOUNT, DATE
   FROM CUSTOMERS
   FULL JOIN ORDERS
   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

SELF JOIN

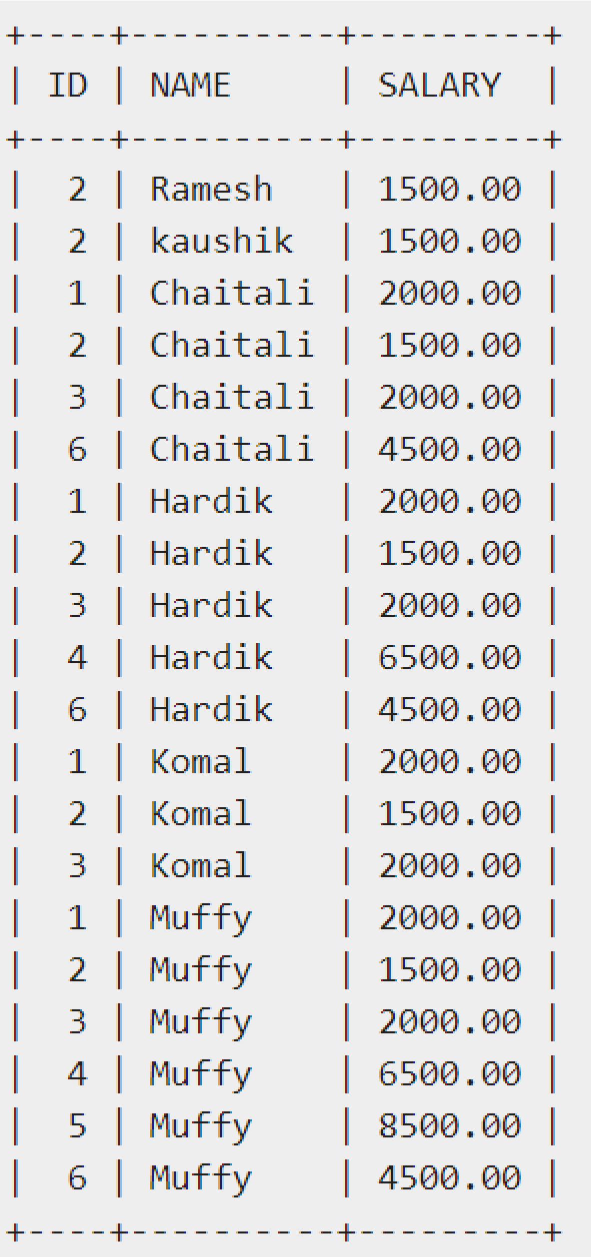
- SQL SELF JOIN is used to join a table to itself as if the table were two tables

```
SELECT  a.ID, b.NAME, a.SALARY
  FROM CUSTOMERS a, CUSTOMERS b
  WHERE a.SALARY < b.SALARY;
```

Table 1 – CUSTOMERS Table



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



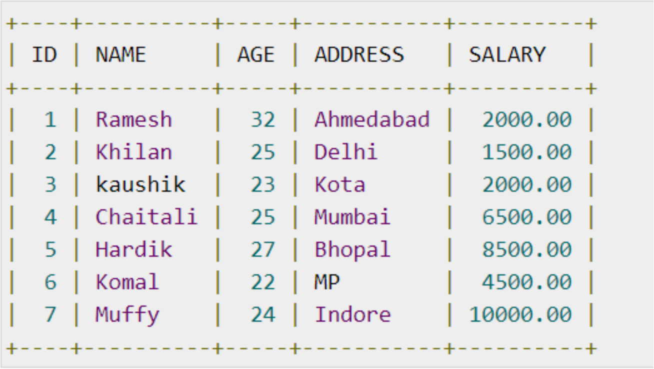
```
+----+----------+---------+
| ID | NAME     | SALARY  |
+----+----------+---------+
|  2 | Ramesh   | 1500.00 |
|  2 | kaushik  | 1500.00 |
|  1 | Chaitali | 2000.00 |
|  2 | Chaitali | 1500.00 |
|  3 | Chaitali | 2000.00 |
|  6 | Chaitali | 4500.00 |
|  1 | Hardik   | 2000.00 |
|  2 | Hardik   | 1500.00 |
|  3 | Hardik   | 2000.00 |
|  4 | Hardik   | 6500.00 |
|  6 | Hardik   | 4500.00 |
|  1 | Komal    | 2000.00 |
|  2 | Komal    | 1500.00 |
|  3 | Komal    | 2000.00 |
|  1 | Muffy    | 2000.00 |
|  2 | Muffy    | 1500.00 |
|  3 | Muffy    | 2000.00 |
|  4 | Muffy    | 6500.00 |
|  5 | Muffy    | 8500.00 |
|  6 | Muffy    | 4500.00 |
+----+----------+---------+
```

CROSS/CARTESIAN JOIN

- Returns the Cartesian product of the sets of records from two or more joined tables.
- Thus, it equates to an inner join where the join-condition always evaluates to either True or where the join-condition is absent from the statement.

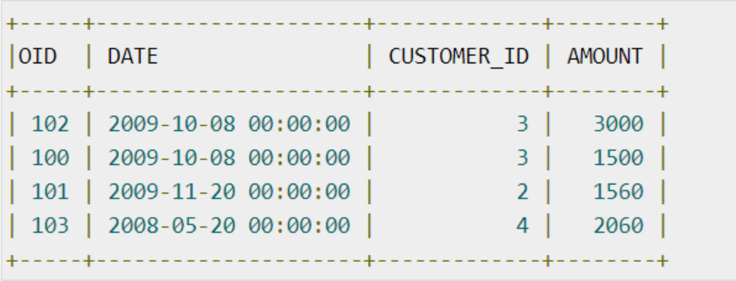
```
SELECT  ID, NAME, AMOUNT, DATE
FROM CUSTOMERS, ORDERS;
```

Table 1 – CUSTOMERS Table

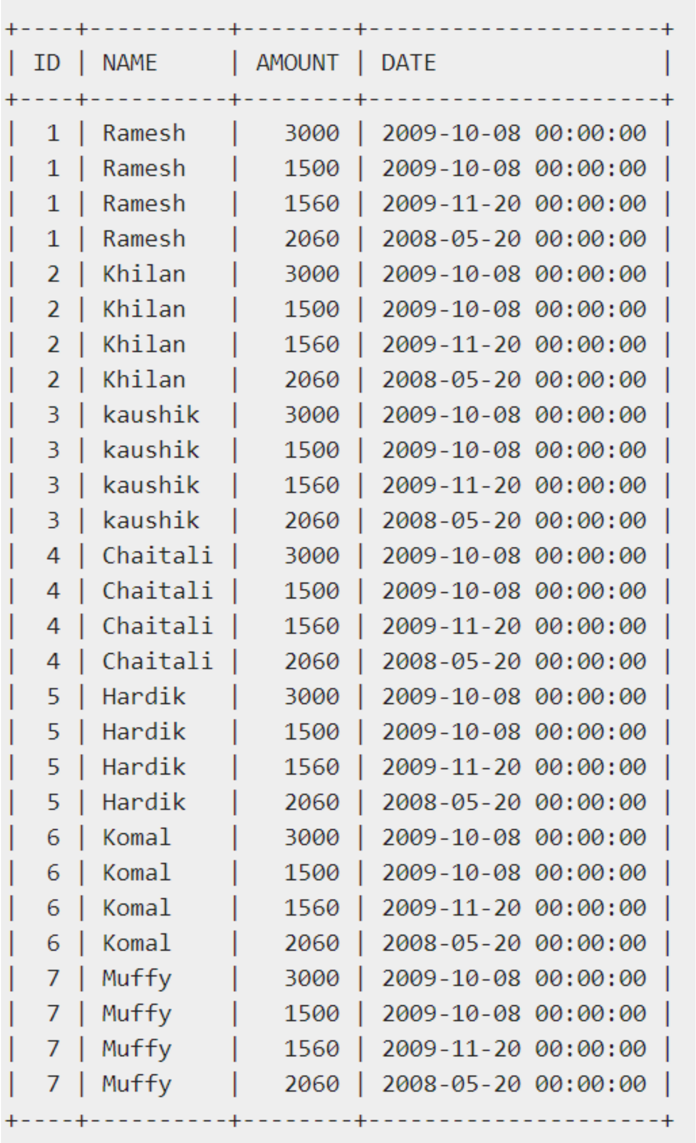


ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Table 2 – ORDERS Table



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



ID	NAME	AMOUNT	DATE
1	Ramesh	3000	2009-10-08 00:00:00
1	Ramesh	1500	2009-10-08 00:00:00
1	Ramesh	1560	2009-11-20 00:00:00
1	Ramesh	2060	2008-05-20 00:00:00
2	Khilan	3000	2009-10-08 00:00:00
2	Khilan	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
2	Khilan	2060	2008-05-20 00:00:00
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
3	kaushik	1560	2009-11-20 00:00:00
3	kaushik	2060	2008-05-20 00:00:00
4	Chaitali	3000	2009-10-08 00:00:00
4	Chaitali	1500	2009-10-08 00:00:00
4	Chaitali	1560	2009-11-20 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	3000	2009-10-08 00:00:00
5	Hardik	1500	2009-10-08 00:00:00
5	Hardik	1560	2009-11-20 00:00:00
5	Hardik	2060	2008-05-20 00:00:00
6	Komal	3000	2009-10-08 00:00:00
6	Komal	1500	2009-10-08 00:00:00
6	Komal	1560	2009-11-20 00:00:00
6	Komal	2060	2008-05-20 00:00:00
7	Muffy	3000	2009-10-08 00:00:00
7	Muffy	1500	2009-10-08 00:00:00
7	Muffy	1560	2009-11-20 00:00:00
7	Muffy	2060	2008-05-20 00:00:00

Nested Queries

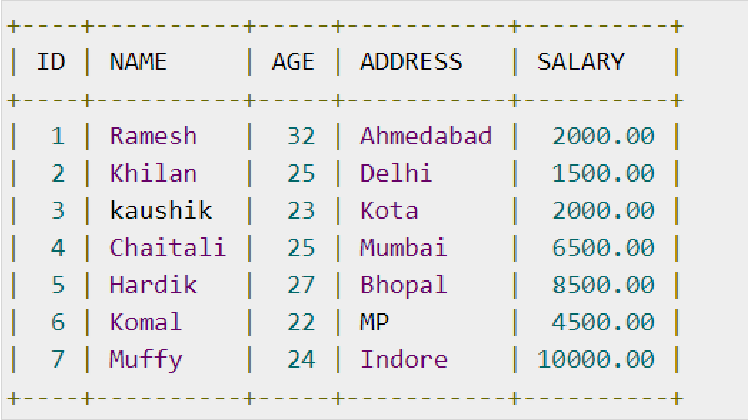
- A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.
- A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.
- Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows

```
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
   (SELECT column_name [, column_name ]
   FROM table1 [, table2 ]
   [WHERE])
```

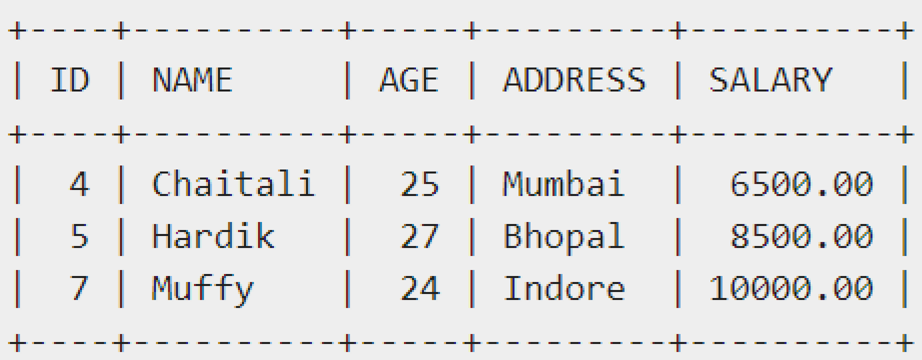
Example

```
SELECT *
  FROM CUSTOMERS
  WHERE ID IN (SELECT ID
        FROM CUSTOMERS
        WHERE SALARY > 4500) ;
```

Table 1 – CUSTOMERS Table



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



```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  5 | Hardik   |  27 | Bhopal  |  8500.00 |
|  7 | Muffy    |  24 | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
```

Subqueries with INSERT statement

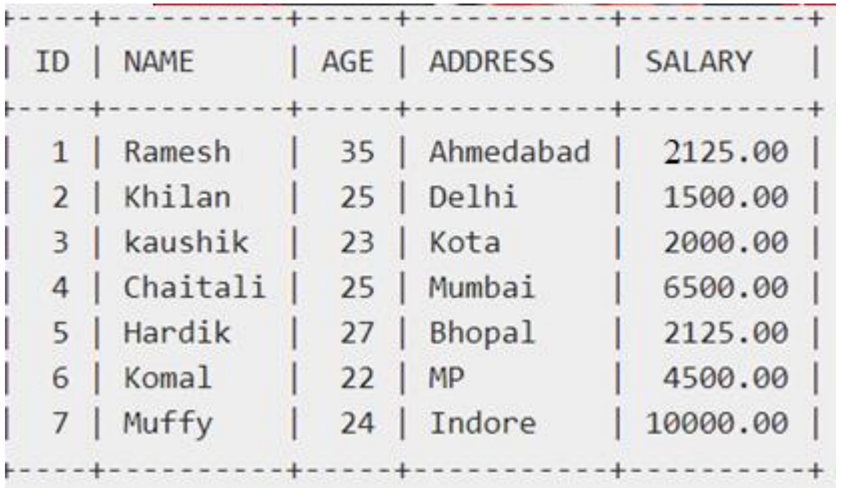
- Consider a table CUSTOMERS_BKP with similar structure as CUSTOMERS table. Now to copy the complete CUSTOMERS table into the CUSTOMERS_BKP table, you can use the following syntax

```
INSERT INTO CUSTOMERS_BKP
  SELECT * FROM CUSTOMERS
  WHERE ID IN (SELECT ID
  FROM CUSTOMERS) ;
```

Subqueries with the UPDATE Statement

- Assuming, we have CUSTOMERS_BKP table available which is backup of CUSTOMERS table. The following example updates SALARY by 0.25 times in the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

```
UPDATE CUSTOMERS
 SET SALARY = SALARY * 0.25
 WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
   WHERE AGE >= 27 );
```

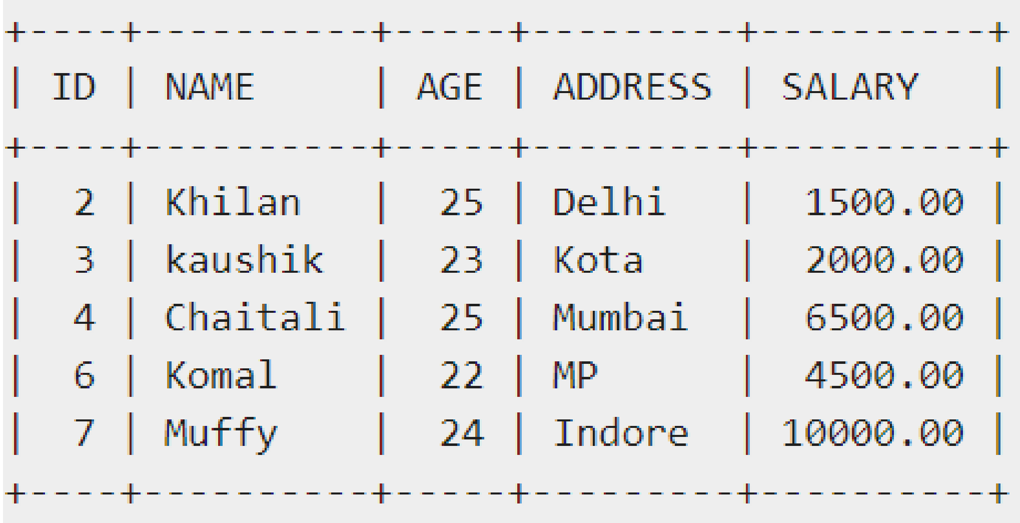


```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2125.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  2125.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Subqueries with the DELETE Statement

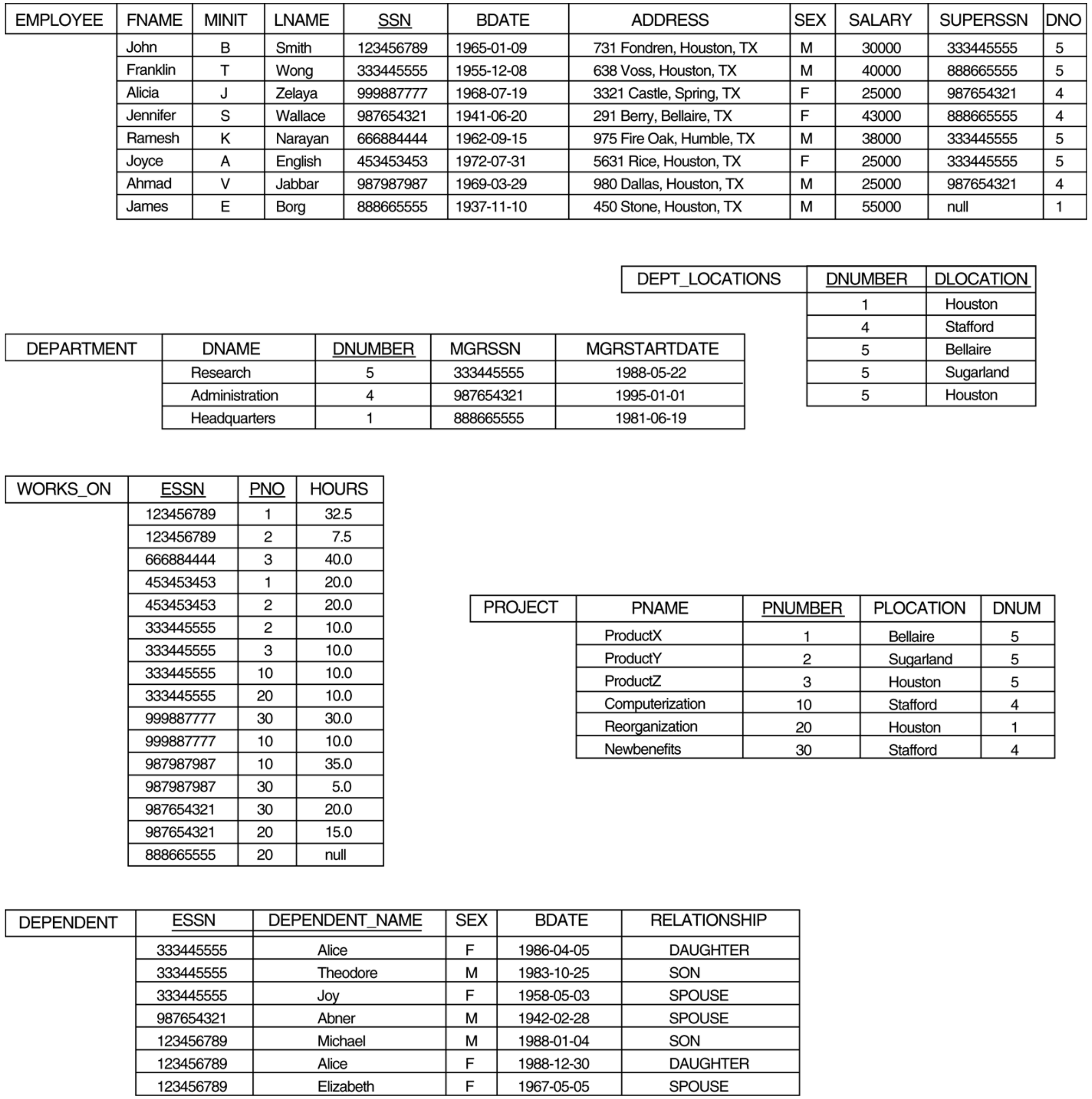
- Assuming, we have a CUSTOMERS_BKP table available which is a backup of the CUSTOMERS table. The following example deletes the records from the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

```
DELETE FROM CUSTOMERS
  WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
    WHERE AGE >= 27 );
```



```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  2 | Khilan   |  25 | Delhi   |  1500.00 |
|  3 | kaushik  |  23 | Kota    |  2000.00 |
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  6 | Komal    |  22 | MP      |  4500.00 |
|  7 | Muffy    |  24 | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
```

Populated Database



EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

DEPT_LOCATIONS	DNUMBER	DLOCATION
	1	Houston
	4	Stafford
	5	Bellaire
	5	Sugarland
	5	Houston

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

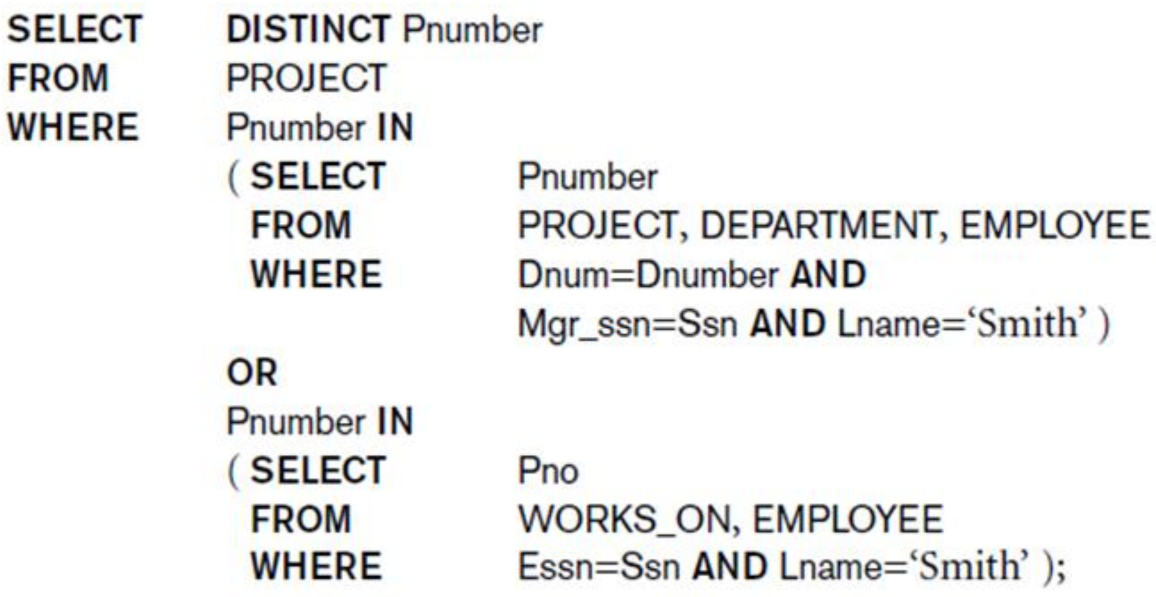
WORKS_ON	ESSN	PNO	HOURS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40.0
	453453453	1	20.0
	453453453	2	20.0
	333445555	2	10.0
	333445555	3	10.0
	333445555	10	10.0
	333445555	20	10.0
	999887777	30	30.0
	999887777	10	10.0
	987987987	10	35.0
	987987987	30	5.0
	987654321	30	20.0
	987654321	20	15.0
	888665555	20	null

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	ProductX	1	Bellaire	5
	ProductY	2	Sugarland	5
	ProductZ	3	Houston	5
	Computerization	10	Stafford	4
	Reorganization	20	Houston	1
	Newbenefits	30	Stafford	4

DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
	333445555	Alice	F	1986-04-05	DAUGHTER
	333445555	Theodore	M	1983-10-25	SON
	333445555	Joy	F	1958-05-03	SPOUSE
	987654321	Abner	M	1942-02-28	SPOUSE
	123456789	Michael	M	1988-01-04	SON
	123456789	Alice	F	1988-12-30	DAUGHTER
	123456789	Elizabeth	F	1967-05-05	SPOUSE

Query

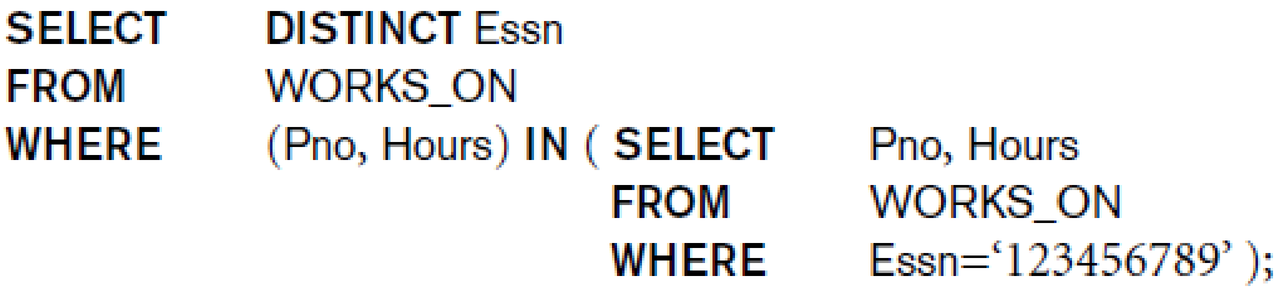
Make a list of project numbers for projects that involve an employee whose last name is ‘Smith’, either as a worker or as manager of the controlling department for the project.



```
SELECT      DISTINCT Pnumber
FROM        PROJECT
WHERE       Pnumber IN
            ( SELECT        Pnumber
              FROM          PROJECT, DEPARTMENT, EMPLOYEE
              WHERE         Dnum=Dnumber AND
                            Mgr_ssn=Ssn AND Lname=‘Smith’ )
            OR
            Pnumber IN
            ( SELECT        Pno
              FROM          WORKS_ON, EMPLOYEE
              WHERE         Essn=Ssn AND Lname=‘Smith’ );
```

Nested Queries (cont’d.)

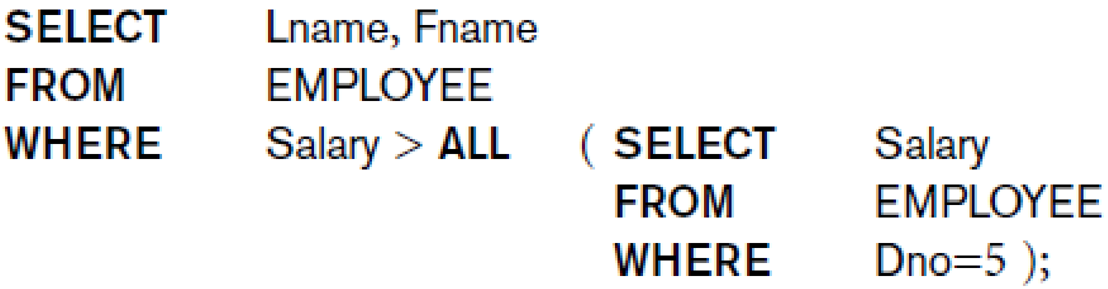
- Use tuples of values in comparisons
    - Place them within parentheses



```
SELECT      DISTINCT Essn
FROM        WORKS_ON
WHERE       (Pno, Hours) IN ( SELECT      Pno, Hours
                              FROM        WORKS_ON
                              WHERE       Essn=‘123456789’ );
```

Nested Queries (cont'd.)

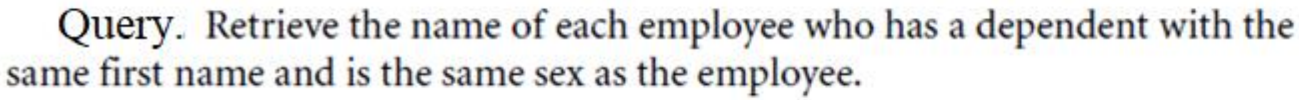
- Use other comparison operators to compare a single value *v*
  - `= ANY` (or `= SOME`) operator
    - Returns `TRUE` if the value *v* is equal to some value in the set *V* and is hence equivalent to `IN`
  - Other operators that can be combined with `ANY` (or `SOME`): >, >=, <, <=, and <>
  - `ALL:` value must exceed all values from nested query



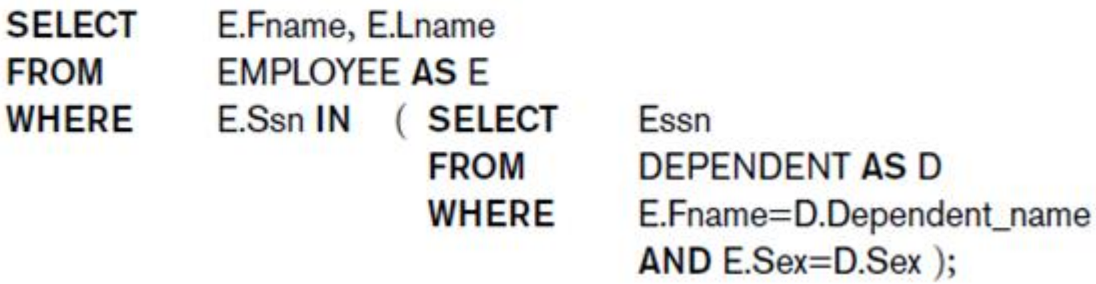
```
SELECT      Lname, Fname
FROM        EMPLOYEE
WHERE       Salary > ALL     ( SELECT      Salary
                               FROM        EMPLOYEE
                               WHERE       Dno=5 );
```

Nested Queries (cont’d.)

- Avoid potential errors and ambiguities
  - Create tuple variables (aliases) for all tables referenced in SQL query



Query. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.



```
SELECT   E.Fname, E.Lname
FROM     EMPLOYEE AS E
WHERE    E.Ssn IN  ( SELECT    Essn
                     FROM      DEPENDENT AS D
                     WHERE     E.Fname=D.Dependent_name
                               AND E.Sex=D.Sex );
```

EXISTS statement

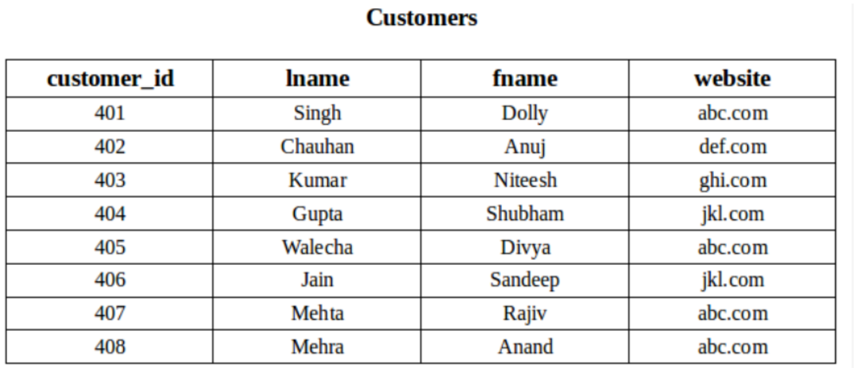
- EXISTS operator is used to test for the existence of any record in a subquery.
- EXISTS operator returns TRUE if the subquery returns one or more records.

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

Example

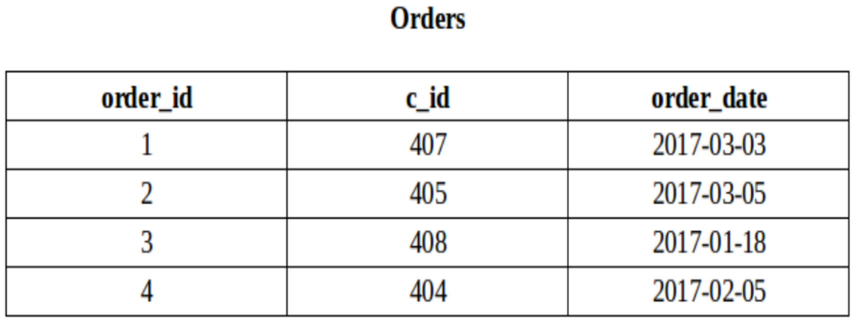
Query: To fetch the first and last name of the customers who placed atleast one order.

```
SELECT fname, lname

FROM Customers

WHERE EXISTS (SELECT *

          FROM Orders

          WHERE Customers.customer_id = Orders.c_id);
```



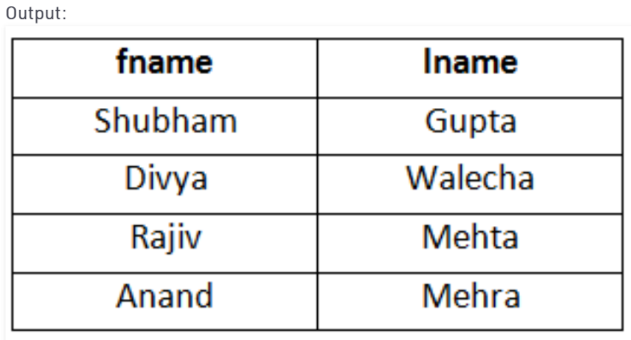
Customers

customer_id	lname	fname	website
401	Singh	Dolly	abc.com
402	Chauhan	Anuj	def.com
403	Kumar	Niteesh	ghi.com
404	Gupta	Shubham	jkl.com
405	Walecha	Divya	abc.com
406	Jain	Sandeep	jkl.com
407	Mehta	Rajiv	abc.com
408	Mehra	Anand	abc.com



Orders

order_id	c_id	order_date
1	407	2017-03-03
2	405	2017-03-05
3	408	2017-01-18
4	404	2017-02-05



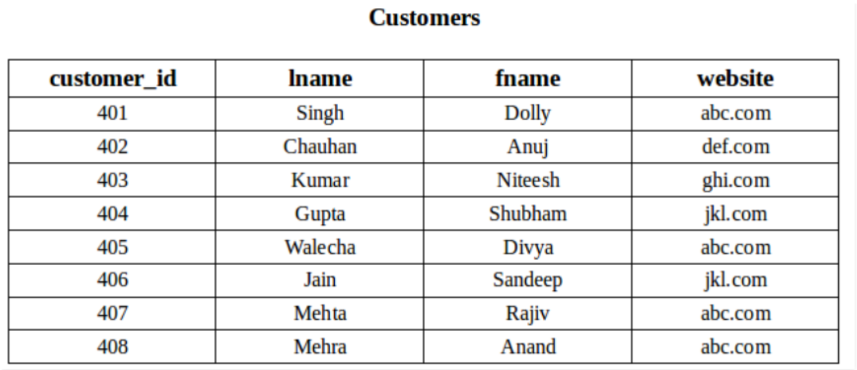
Output:

fname	lname
Shubham	Gupta
Divya	Walecha
Rajiv	Mehta
Anand	Mehra

Using NOT with EXISTS

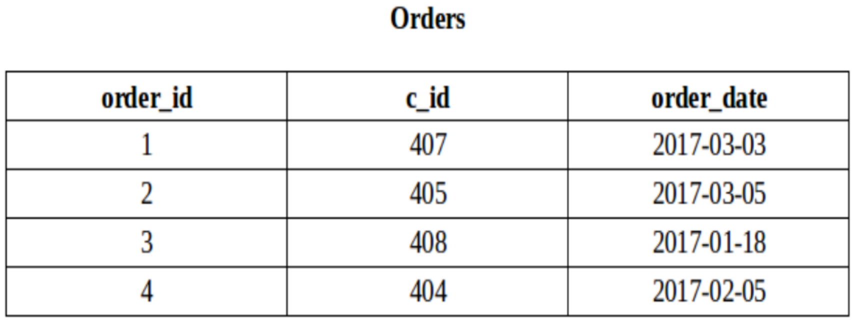
Query: To fetch the first and last name of the customers who has not placed any order.

```
SELECT fname, lname

FROM Customers

WHERE NOT EXISTS (SELECT *

          FROM Orders

          WHERE Customers.customer_id = Orders.c_id);
```



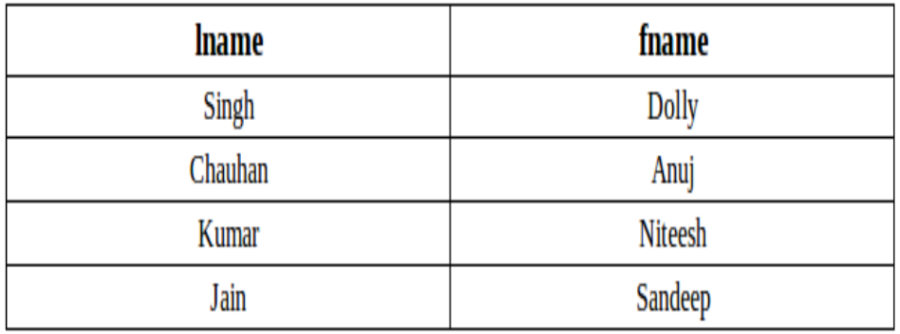
Customers

customer_id	lname	fname	website
401	Singh	Dolly	abc.com
402	Chauhan	Anuj	def.com
403	Kumar	Niteesh	ghi.com
404	Gupta	Shubham	jkl.com
405	Walecha	Divya	abc.com
406	Jain	Sandeep	jkl.com
407	Mehta	Rajiv	abc.com
408	Mehra	Anand	abc.com



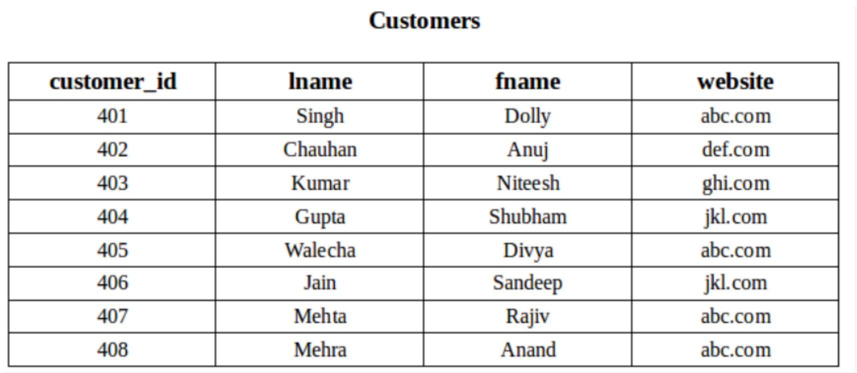
Orders

order_id	c_id	order_date
1	407	2017-03-03
2	405	2017-03-05
3	408	2017-01-18
4	404	2017-02-05



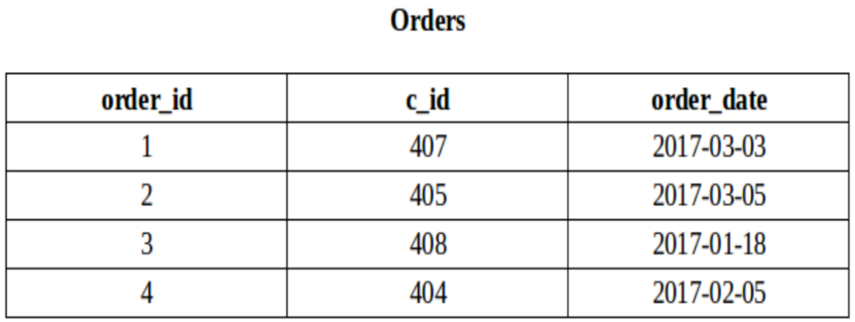
lname	fname
Singh	Dolly
Chauhan	Anuj
Kumar	Niteesh
Jain	Sandeep

Using EXISTS condition with DELETE statement



Customers

customer_id	lname	fname	website
401	Singh	Dolly	abc.com
402	Chauhan	Anuj	def.com
403	Kumar	Niteesh	ghi.com
404	Gupta	Shubham	jkl.com
405	Walecha	Divya	abc.com
406	Jain	Sandeep	jkl.com
407	Mehta	Rajiv	abc.com
408	Mehra	Anand	abc.com

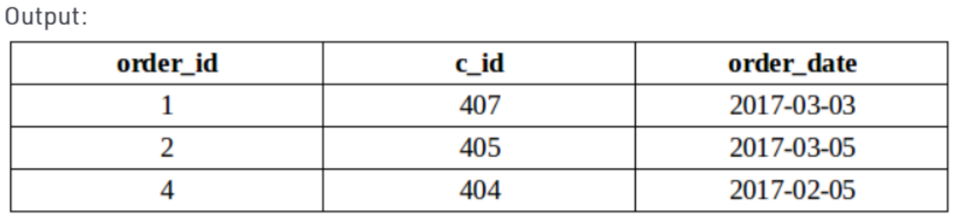


Orders

order_id	c_id	order_date
1	407	2017-03-03
2	405	2017-03-05
3	408	2017-01-18
4	404	2017-02-05

Query: Delete the record of all the customer from Order Table whose last name is ‘Mehra’.

```
DELETE
FROM Orders
WHERE EXISTS (SELECT *
          FROM customers
          WHERE Customers.customer_id = Orders.cid
          AND Customers.lname = 'Mehra');
```



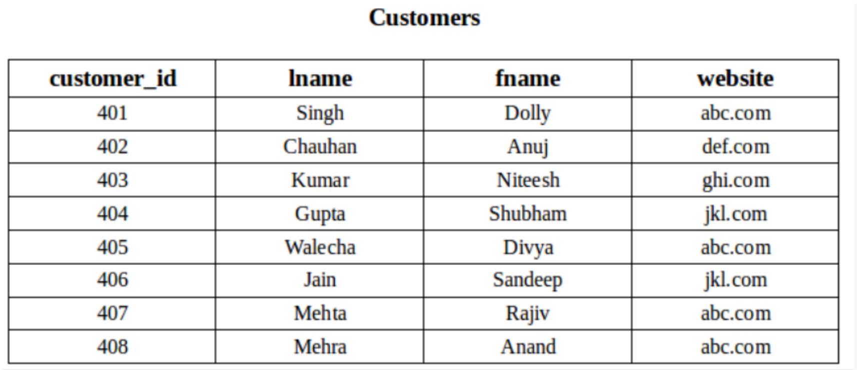
Output:

order_id	c_id	order_date
1	407	2017-03-03
2	405	2017-03-05
4	404	2017-02-05

Using EXISTS condition with UPDATE statement

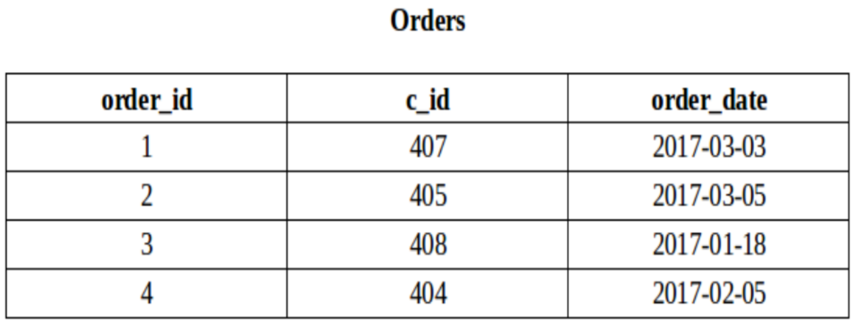
Query: Update the lname as ‘Kumari’ of customer in Customer Table whose customer_id is 401.

```
UPDATE Customers

SET lname = 'Kumari'

WHERE EXISTS (SELECT *

          FROM Customers

          WHERE customer_id = 401);
```



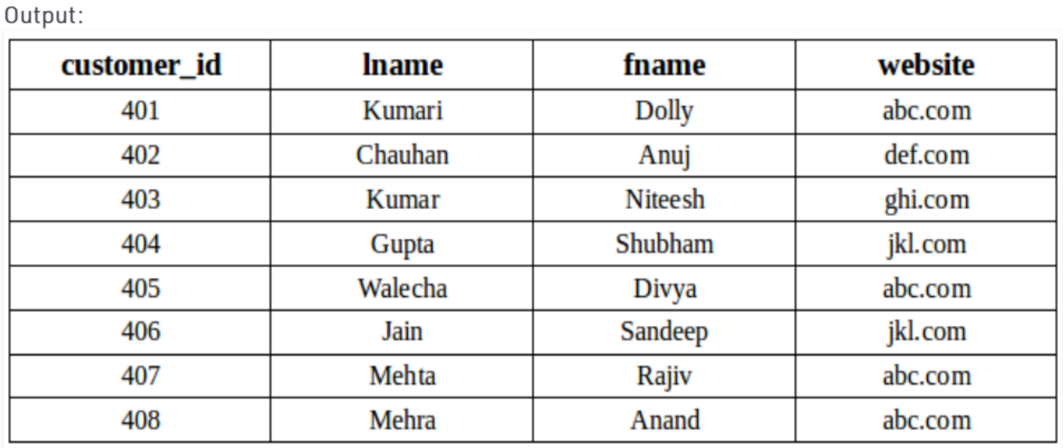
Customers

customer_id	lname	fname	website
401	Singh	Dolly	abc.com
402	Chauhan	Anuj	def.com
403	Kumar	Niteesh	ghi.com
404	Gupta	Shubham	jkl.com
405	Walecha	Divya	abc.com
406	Jain	Sandeep	jkl.com
407	Mehta	Rajiv	abc.com
408	Mehra	Anand	abc.com



Orders

order_id	c_id	order_date
1	407	2017-03-03
2	405	2017-03-05
3	408	2017-01-18
4	404	2017-02-05



Output:

customer_id	lname	fname	website
401	Kumari	Dolly	abc.com
402	Chauhan	Anuj	def.com
403	Kumar	Niteesh	ghi.com
404	Gupta	Shubham	jkl.com
405	Walecha	Divya	abc.com
406	Jain	Sandeep	jkl.com
407	Mehta	Rajiv	abc.com
408	Mehra	Anand	abc.com

Triggers

- **Triggers** are executed when a specified condition occurs during insert/delete/update
    - Triggers are action that fire automatically based on these conditions

Event-Condition-Action (ECA) Model

- Triggers follow an Event-condition-action (ECA) model
  - **Event**:
    - Database modification
      - E.g., insert, delete, update
  - **Condition**:
    - Any true/false expression
      - Optional: If no condition is specified then condition is always true
  - **Action**:
    - Sequence of SQL statements that will be automatically executed

Example: Trigger Definition

Can be CREATE or ALTER

```
CREATE TRIGGER Total_sal1
   AFTER INSERT ON Employee
   FOR EACH ROW
   WHEN (NEW.Dno is NOT NULL)
      UPDATE DEPARTMENT
      SET Total_sal = Total_sal + NEW. Salary
      WHERE Dno = NEW.Dno;
```

Can be FOR, AFTER, INSTEAD OF

Can be INSERT, UPDATE, DELETE

The condition

The action

Note: In oracle, to reference a pseudorecord, put a colon before its name—:OLD or :NEW

CREATE or ALTER TRIGGER

- CREATE TRIGGER <name>
  - Creates a trigger
- ALTER TRIGGER <name>
  - Alters a trigger (assuming one exists)
- CREATE OR ALTER TRIGGER <name>
  - Creates a trigger if one does not exist
  - Alters a trigger if one does exist
  - Works in both cases, whether a trigger exists or not

Note: In oracle, use replace instead of alter.

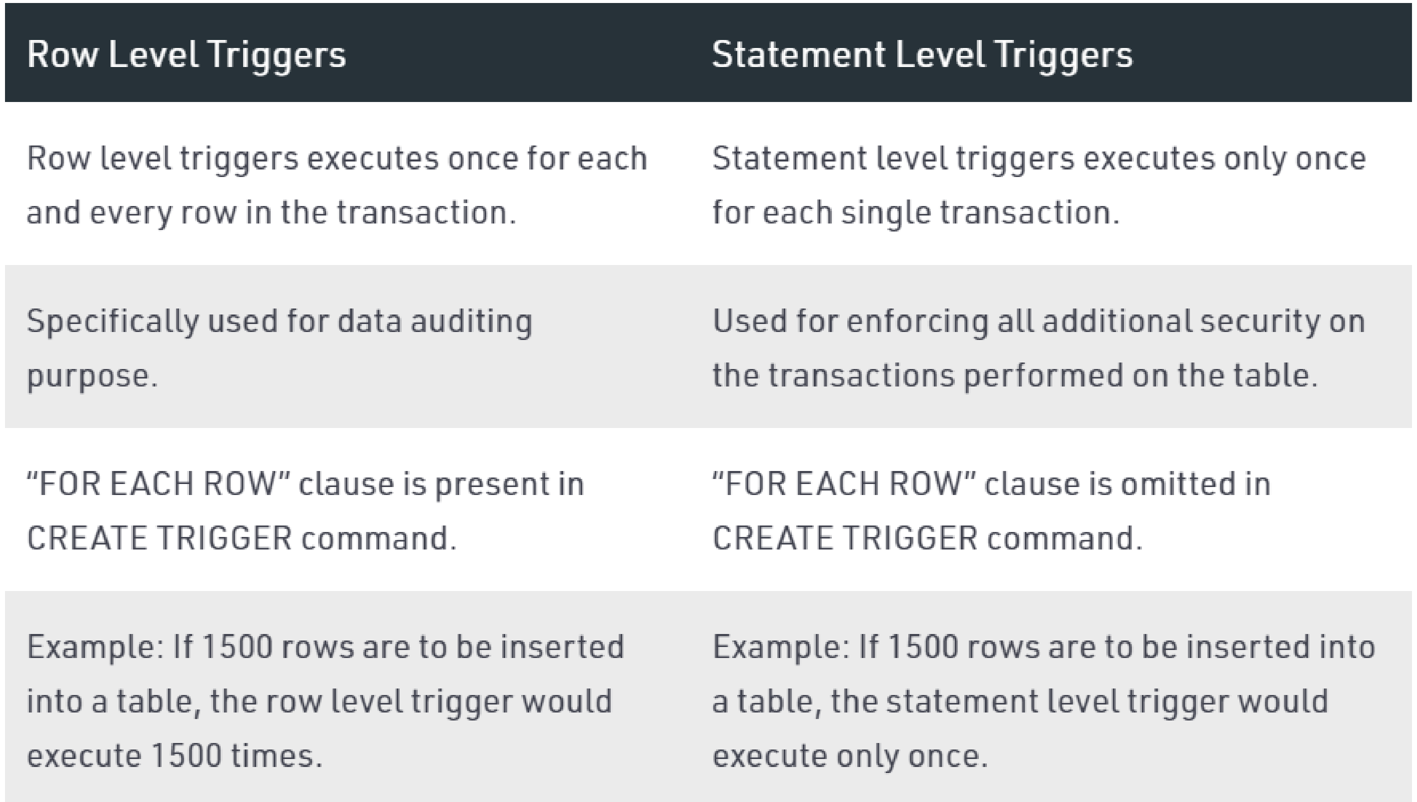
Conditions

- AFTER
  - Executes after the event
- BEFORE
  - Executes before the event
- INSTEAD OF
  - Executes **instead of** the event
    - Note that event does not execute in this case

Trigger types

- Triggers can be
  - **Row-level**
    - FOR EACH ROW specifies a row-level trigger
  - **Statement-level**
    - Default (when FOR EACH ROW is not specified)
- Row level triggers
  - Executed separately for each affected row
- Statement-level triggers
  - Execute once for the SQL statement,

Row-Level versus Statement-level

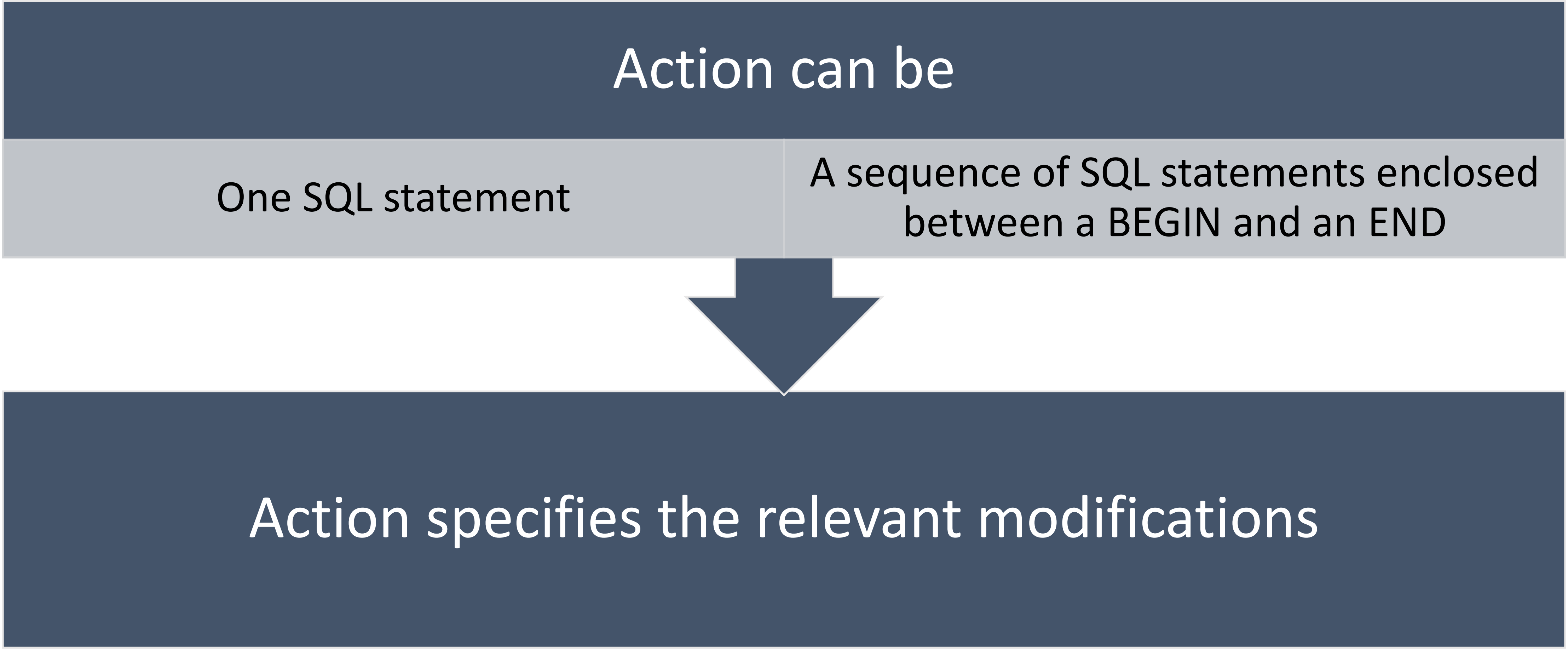


Row Level Triggers	Statement Level Triggers
Row level triggers executes once for each and every row in the transaction.	Statement level triggers executes only once for each single transaction.
Specifically used for data auditing purpose.	Used for enforcing all additional security on the transactions performed on the table.
“FOR EACH ROW” clause is present in CREATE TRIGGER command.	“FOR EACH ROW” clause is omitted in CREATE TRIGGER command.
Example: If 1500 rows are to be inserted into a table, the row level trigger would execute 1500 times.	Example: If 1500 rows are to be inserted into a table, the statement level trigger would execute only once.

Condition

- Any true/false condition to control whether a trigger is activated on not
- Absence of condition means that the trigger will always execute.
- Otherwise, condition is evaluated
    - before the event for BEFORE trigger
    - after the event for AFTER trigger

Action



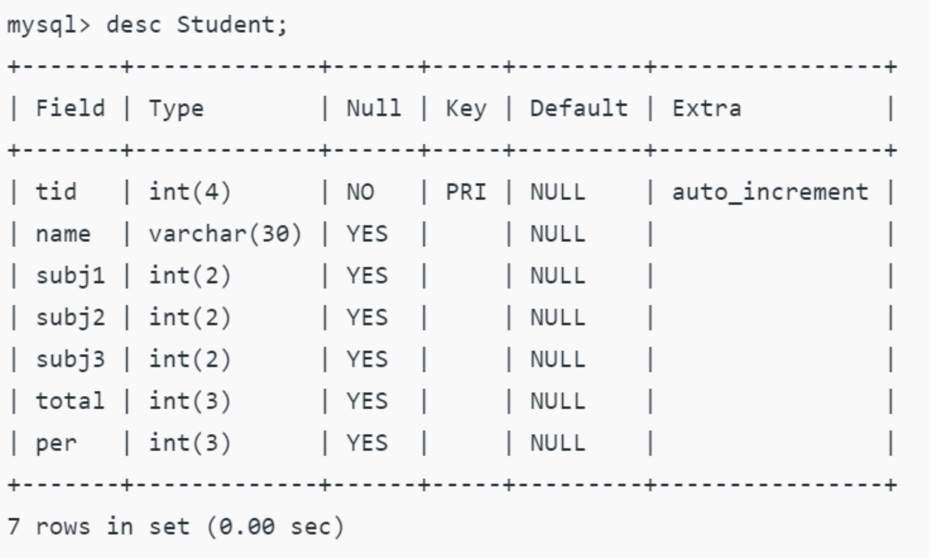
Action can be

One SQL statement

A sequence of SQL statements enclosed between a BEGIN and an END

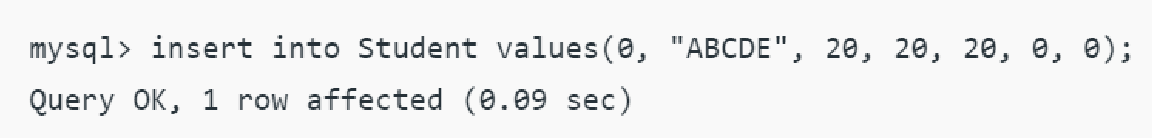
Action specifies the relevant modifications

Example

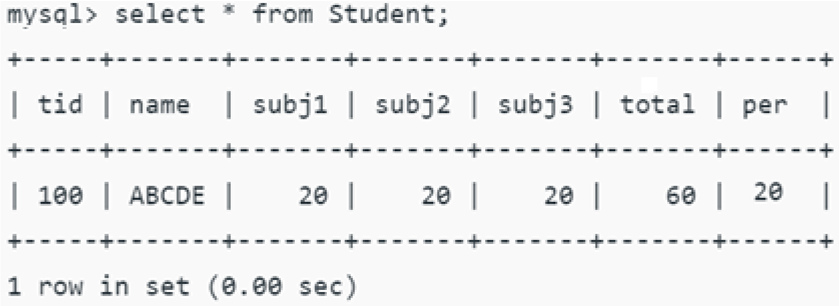


```
mysql> desc Student;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| tid   | int(4)      | NO   | PRI | NULL    | auto_increment |
| name  | varchar(30) | YES  |     | NULL    |                |
| subj1 | int(2)      | YES  |     | NULL    |                |
| subj2 | int(2)      | YES  |     | NULL    |                |
| subj3 | int(2)      | YES  |     | NULL    |                |
| total | int(3)      | YES  |     | NULL    |                |
| per   | int(3)      | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

```
create trigger stud_marks
after INSERT  on  Student
for each row
update student
set total =subj1 + subj2 + subj3, per=(total*1000)/300
```



```
mysql> insert into Student values(0, "ABCDE", 20, 20, 20, 0, 0);
Query OK, 1 row affected (0.09 sec)
```



```
mysql> select * from Student;
+-----+-------+-------+-------+-------+-------+------+
| tid | name  | subj1 | subj2 | subj3 | total | per  |
+-----+-------+-------+-------+-------+-------+------+
| 100 | ABCDE |    20 |    20 |    20 |    60 |  20  |
+-----+-------+-------+-------+-------+-------+------+
1 row in set (0.00 sec)
```

Trigger operations

- Viewing all triggers details
  - Select * from user_triggers
- Dropping triggers
  - Drop trigger <name>
  - Eg: Drop trigger stud_marks
- Enabling/Disabling triggers
  - Alter trigger <name> {disable|enable}
  - Eg: Alter trigger stud_marks disable
- Note: All queries working in oracle