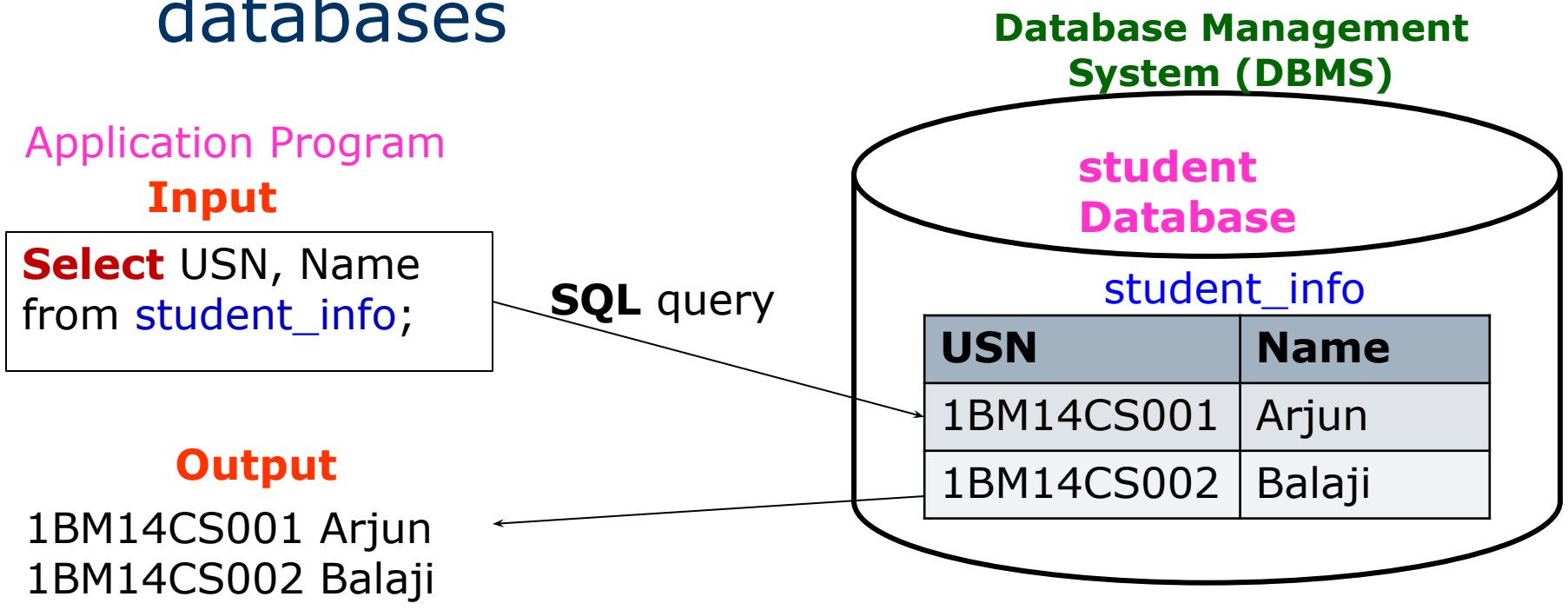
Topics Covered in Todays Class

Unit 1:

- SQL Data Definition and Data Types
- Specifying basic constraints in SQL

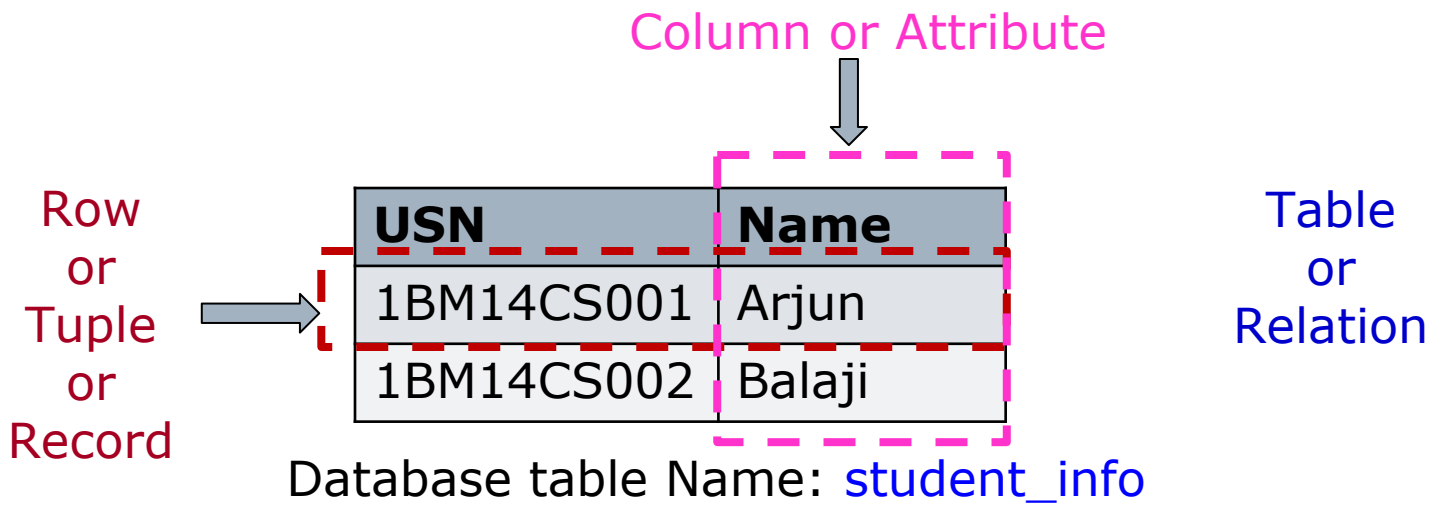
What is SQL ?

- SQL stands for **Structured Query Language**
- SQL lets you access and manipulate databases



RDBMS

- RDBMS stands for **Relational Database Management System**.
- RDBMS is the basis for SQL, and for all modern database systems such as Oracle, MySQL, MS SQL Server, IBM DB2, and Microsoft Access.
- The data in RDBMS is stored in database objects called **tables**.
- A **table is a collection** of related data entries and it consists of **columns and rows**.



SQL commands fro Data Definition

Command	Description
create	to create new table or database
alter	for alteration
truncate	delete data from table
drop	to drop a table
rename	to rename a table

SQL: create command

- **create** is a DDL (Data Definition) command used to create a table or a database.

Creating a Database

- Syntax: **create** database *database-name*;
- Example: create database student;

SQL: create command

- **create** is a DDL (Data Definition) command used to create a table or a database.

Creating a Database

- Syntax: **create** database *database-name*;
- Example: create database student;

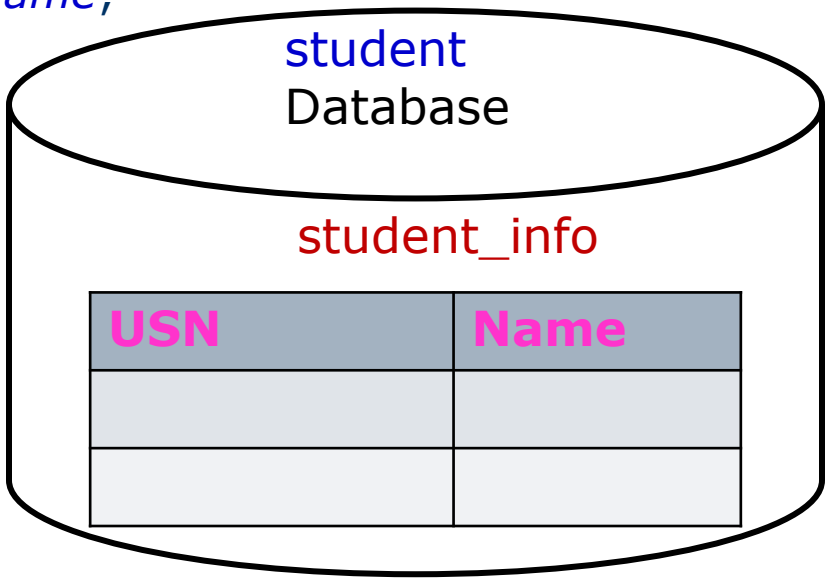
Creating a Table

- Syntax:

create table *table-name* (
column-name1 datatype1,
column-name2 datatype2);

- Example:

```
create table student.student_info(
USN char(10),
Name char(30)
);
```



student is database name or schema name
student_info is the table name

Here student_info will be created under the database student

SQL: create command

What is the difference between

```
create table student.student_info(
USN char(10),
Name char(30)
);
```

```
create table student_info(
USN char(10),
Name char(30);
);
```

Specifying Constraints in SQL

- Specifying Attributes constraints and Attribute values
- Specifying Key and Referential Integrity constraints
- Specifying constraints on Tuples using CHECK

Specifying Attributes constraints and Attribute values

```
create table student.student_info(
USN char(10),
Name char(30) NOT NULL,
DepName char(3) NOT NULL DEFAULT 'CSE',
Marks int NOT NULL CHECK (Marks > 0 AND Marks < 101)
);
```

Constraints on Attributes

- NOT NULL
- DEAFULT
- CHECK

Specifying Key and Referential Integrity constraints

PRIMARY KEY Constraint

```
create table student.student_info(
USN char(10),
Name char(30) NOT NULL,
DepName char(3) NOT NULL DEFAULT 'CSE',
Marks int NOT NULL CHECK (Marks > 0 AND Marks < 101)
PRIMARY KEY (USN)
);
```

Specifying Key and Referential Integrity constraints

PRIMARY KEY Constraint

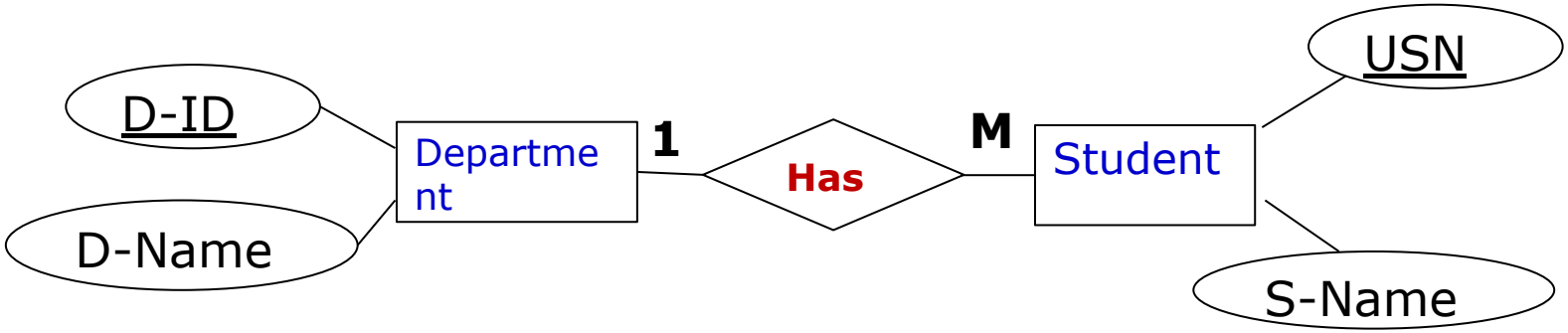
Question: What is the difference between following two create commands ?

```
create table student.student_info(
USN char(10),
Name char(30) NOT NULL,
DepName char(3) NOT NULL DEFAULT 'CSE',
Marks int NOT NULL CHECK (Marks > 0 AND Marks < 101)
PRIMARY KEY (USN)
);

create table student.student_info(
USN char(10),
Name char(30) NOT NULL,
DepName char(3) NOT NULL DEFAULT 'CSE',
Marks int NOT NULL CHECK (Marks > 0 AND Marks < 101)
CONSTRAINT usn_pk PRIMARY KEY (USN)
);
```

Specifying Key and Referential Integrity constraints

- Referential Integrity constraint or Foreign Key Constraint




```
create  table Department(
D_ID int,
D_Name char(3),
PRIMARY KEY (D_ID)
);
```

```
create table Student(
USN char(10),
S_Name char(20),
Dep_Num int,
PRIMARY KEY (USN),
FOREIGN KEY(Dep_Num) REFERENCES Department(D_ID)
);
```

Specifying Key and Referential Integrity constraints

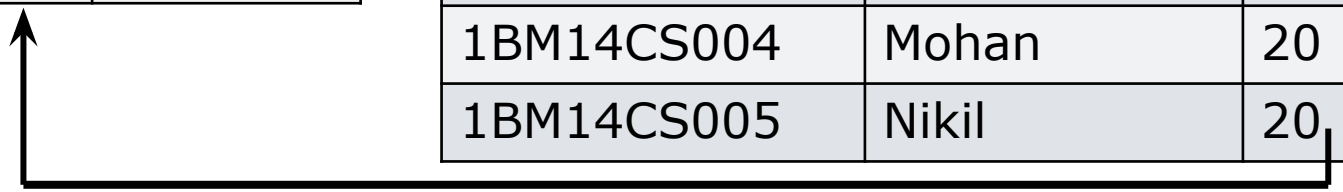
- Referential Integrity constraint or Foreign Key Constraint

Department Table

D_ID	D-Name
10	CSE
20	ISE

Student Table

USN	S-Name	Dep_ Num
1BM14CS001	Akash	10
1BM14CS002	Bharath	10
1BM14CS003	Ragu	10
1BM14CS004	Mohan	20
1BM14CS005	Nikil	20



Question

1. Assume, D_ID value of **CSE** department in Department table has been updated to **50**. Then how the values of **Dep_Num** column in Student table should be affected
2. Assume, Department table has been deleted. Then how the values of **Dep_Num** column in Student table should be affected

Specifying Key and Referential Integrity constraints

- Referential Integrity constraint or Foreign Key Constraint

```
create  table Department(
D_ID int,
D_Name int(3),
PRIMARY KEY (D_ID)
);
```

```
create table Student(
USN char(10),
S_Name char(20),
Dep_Num int,
PRIMARY KEY (USN),
FOREIGN KEY(Dep_Num) REFERENCES Department(D_ID)
ON DELETE CASCADE ON UPDATE CASCADE
);
```

Specifying Key and Referential Integrity constraints

- Referential Integrity constraint or Foreign Key Constraint

```
create  table Department(
D_ID int,
D_Name int(3),
PRIMARY KEY (D_ID)
);
```

```
create table Student(
USN char(10),
S_Name char(20),
Dep_Num int,
PRIMARY KEY (USN),
FOREIGN KEY(Dep_Num) REFERENCES Department(D_ID)
ON DELETE SET NULL ON UPDATE CASCADE
);
```

Specifying Key and Referential Integrity constraints

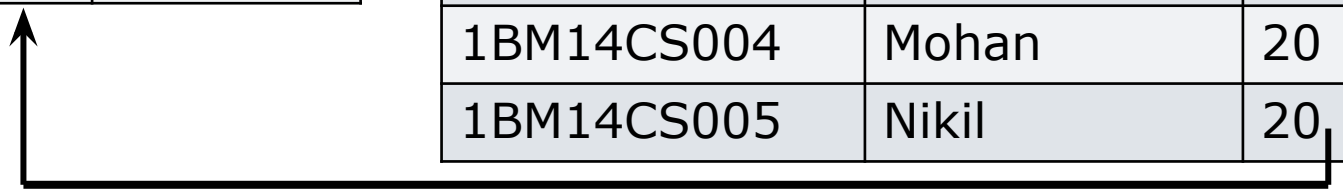
- Referential Integrity constraint or Foreign Key Constraint

Department Table

D_ID	D-Name
10	CSE
20	ISE

Student Table

USN	S-Name	Dep_ Num
1BM14CS001	Akash	10
1BM14CS002	Bharath	10
1BM14CS003	Ragu	10
1BM14CS004	Mohan	20
1BM14CS005	Nikil	20



Question

1. When the following SQL Command is executed, what will be the status of Student table contents?

Update Department set D_ID=50 where D_ID=10;

Specifying Key and Referential Integrity constraints

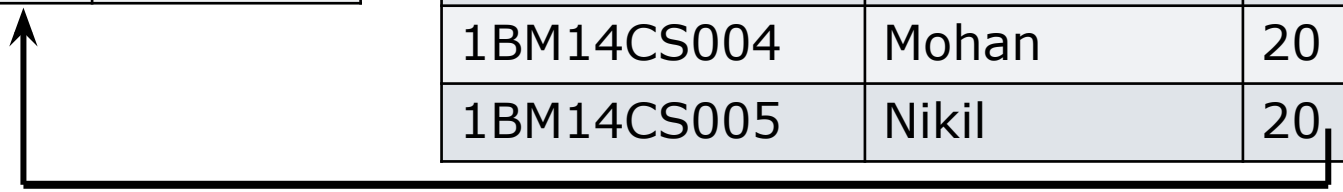
- Referential Integrity constraint or Foreign Key Constraint

Department Table

D_ID	D-Name
10	CSE
20	ISE

Student Table

USN	S-Name	Dep_ Num
1BM14CS001	Akash	10
1BM14CS002	Bharath	10
1BM14CS003	Ragu	10
1BM14CS004	Mohan	20
1BM14CS005	Nikil	20



Question

1. When the following SQL Command is executed, what will be the status of Student table contents?

Delete from Department where D_ID=10;

Specifying Key and Referential Integrity constraints

- Following SQL commands will be executed successfully ? YES / NO

```
create  table Department(
D_ID int,
D_Name char(3),
PRIMARY KEY (D_ID)
);
```

```
create table Student(
USN char(10),
S_Name char(20),
Dep_Num char(3),
PRIMARY KEY (USN),
FOREIGN KEY(Dep_Num) REFERENCES Department(D_ID)
ON DELETE SET NULL ON UPDATE CASCADE
);
```

Specifying Key and Referential Integrity constraints

- ON DELETE CASCADE
- ON DELETE SET NULL
- ON DELETE SET DEFAULT

- ON UPDATE CASCADE
- ON UPDATE SET NULL
- ON UPDATE SET DEFAULT

Specifying Key and Referential Integrity constraints

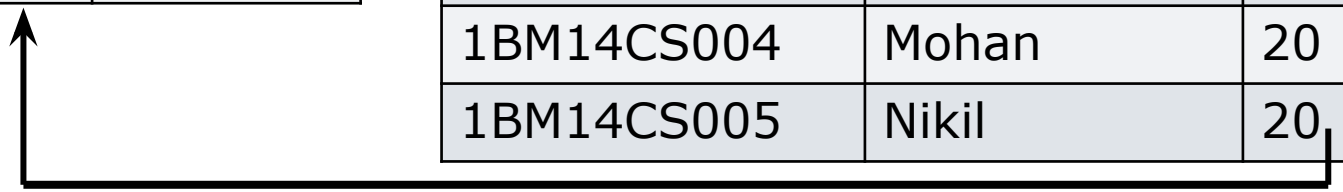
- Referential Integrity constraint or Foreign Key Constraint

Department Table

D_ID	D-Name
10	CSE
20	ISE

Student Table

USN	S-Name	Dep_ Num
1BM14CS001	Akash	10
1BM14CS002	Bharath	10
1BM14CS003	Ragu	10
1BM14CS004	Mohan	20
1BM14CS005	Nikil	20



Question

1. When the following SQL Command is executed, what will be the status of Student table contents?

Delete from Department where D_ID=10;

Topics Covered in Todays Class

Unit 1:
- Schema change statements in SQL
- Basic SQL queries

Schema change statements in SQL

- Drop command can be used to drop tables or constraints.

- The DROP TABLE Statement

General Syntax: DROP TABLE table_name

- The DROP DATABASE Statement

General Syntax: DROP DATABASE database_name

- The TRUNCATE TABLE Statement

What if we only want to delete the data inside the table, and not the table itself ? Then, use the TRUNCATE TABLE statement:

General Syntax: TRUNCATE TABLE table_name

Schema change statements in SQL

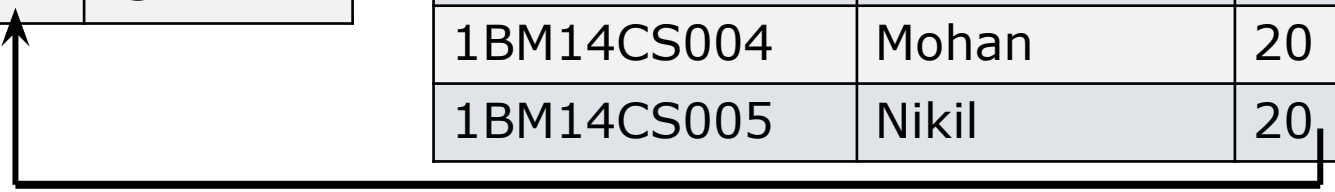
- Drop command

department Table

D_ID	Dep_name
10	CSE
20	ISE

student Table

USN	Name	Dep_num
1BM14CS001	Avinash	10
1BM14CS002	Balaji	10
1BM14CS003	Ram	10
1BM14CS004	Mohan	20
1BM14CS005	Nikil	20



Assume the above two tables were created using following create commands

create table department (D_ID integer, Dep_name varchar(3),primary key (D_ID));

create table student (USN varchar(10),Name varchar(20),Dep_num integer, primary key(usn),foreign key (Dep_num) references department(D_ID));

Note: Assume after above two table creation, values shown above are inserted into tables

Schema change statements in SQL

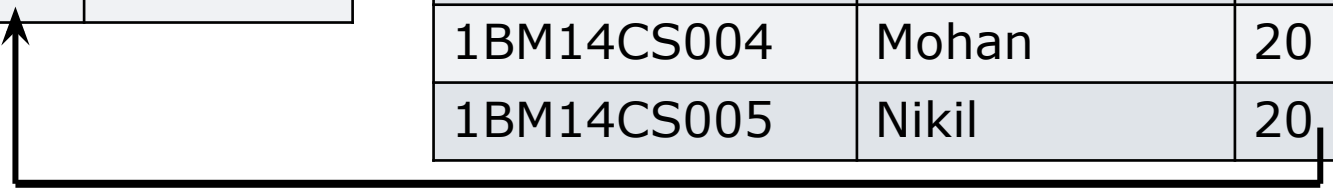
- Drop command

department Table

D_ID	Dep_name
10	CSE
20	ISE

student Table

USN	Name	Dep_num
1BM14CS001	Avinash	10
1BM14CS002	Balaji	10
1BM14CS003	Ram	10
1BM14CS004	Mohan	20
1BM14CS005	Nikil	20



Question

1. When the following SQL Command is executed, what will be the output

insert into student values ('1BM14CS006','Patel',50);

Schema change statements in SQL

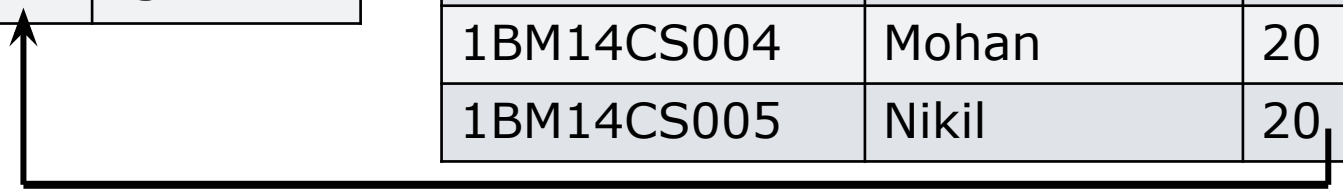
- Drop command

department Table

D_ID	Dep_name
10	CSE
20	ISE

student Table

USN	Name	Dep_num
1BM14CS001	Avinash	10
1BM14CS002	Balaji	10
1BM14CS003	Ram	10
1BM14CS004	Mohan	20
1BM14CS005	Nikil	20



Question

1. When the following SQL Command is executed, what will be the output

insert into student values ('1BM14CS006','Patel',50);

Cannot add a row: a foreign key constraint fails

Schema change statements in SQL

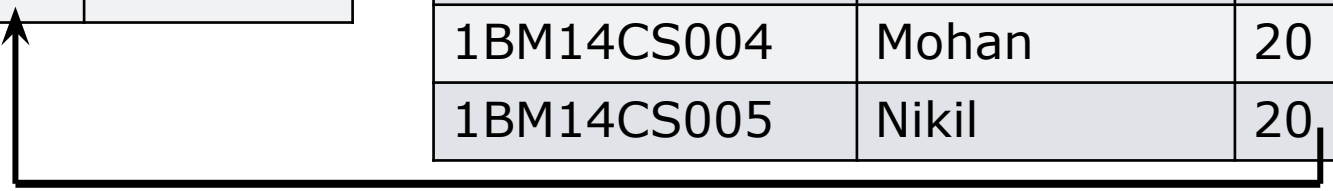
- Drop command

department Table

D_ID	Dep_name
10	CSE
20	ISE

student Table

USN	Name	Dep_num
1BM14CS001	Avinash	10
1BM14CS002	Balaji	10
1BM14CS003	Ram	10
1BM14CS004	Mohan	20
1BM14CS005	Nikil	20



Question

1. When the following SQL Command is executed, what will be the output

drop table department;

Schema change statements in SQL

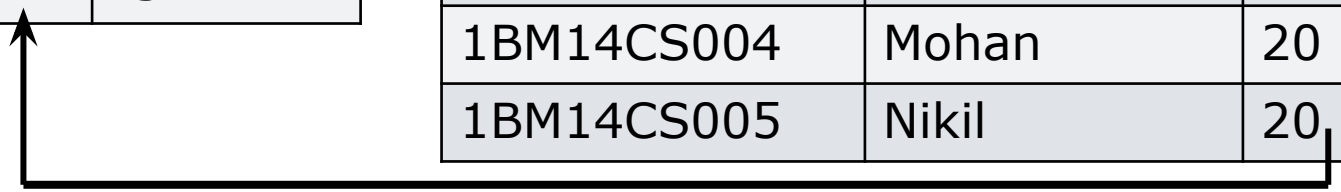
- Drop command

department Table

D_ID	Dep_name
10	CSE
20	ISE

student Table

USN	Name	Dep_num
1BM14CS001	Avinash	10
1BM14CS002	Balaji	10
1BM14CS003	Ram	10
1BM14CS004	Mohan	20
1BM14CS005	Nikil	20



Question

1. When the following SQL Command is executed, what will be the output

drop table department;

Department Table will not be deleted because of foreign key constraint

Schema change statements in SQL

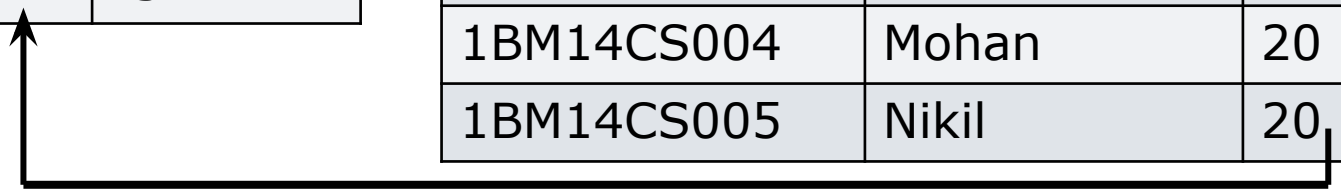
- Drop command

department Table

D_ID	Dep_name
10	CSE
20	ISE

student Table

USN	Name	Dep_num
1BM14CS001	Avinash	10
1BM14CS002	Balaji	10
1BM14CS003	Ram	10
1BM14CS004	Mohan	20
1BM14CS005	Nikil	20



Question

1. When the following SQL Command is executed, what will be the output

drop table student;

Schema change statements in SQL

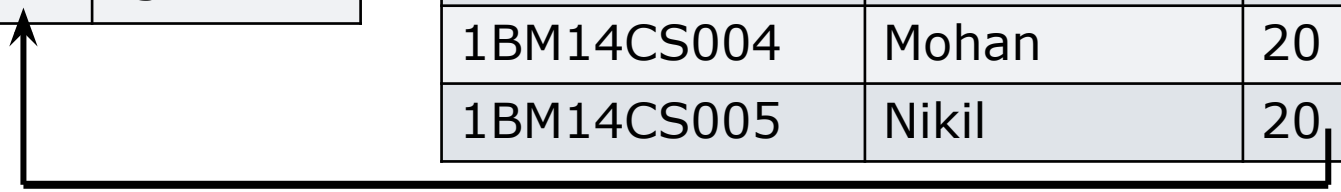
- Drop command

department Table

D_ID	Dep_name
10	CSE
20	ISE

student Table

USN	Name	Dep_num
1BM14CS001	Avinash	10
1BM14CS002	Balaji	10
1BM14CS003	Ram	10
1BM14CS004	Mohan	20
1BM14CS005	Nikil	20



Question

1. When the following SQL Command is executed, what will be the output

drop table student;

student Table will be deleted

Schema change statements in SQL

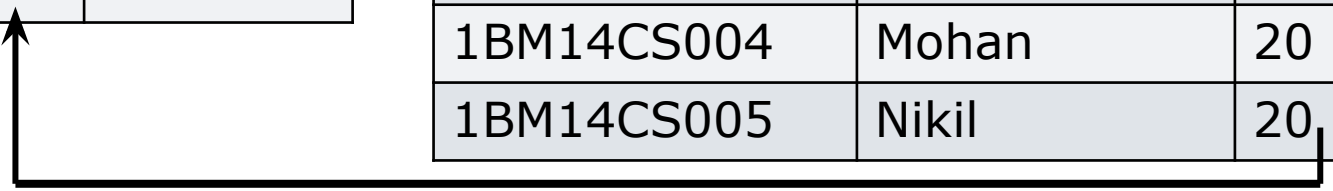
- Drop command

department Table

D_ID	Dep_name
10	CSE
20	ISE

student Table

USN	Name	Dep_ Num
1BM14CS001	Avinash	10
1BM14CS002	Balaji	10
1BM14CS003	Ram	10
1BM14CS004	Mohan	20
1BM14CS005	Nikil	20



Question

1. When the following SQL Command is executed, what will be the output

drop table department cascade constraints;

Schema change statements in SQL

- Drop command

department Table

D_ID	Dep_name
10	CSE
20	ISE

student Table

USN	Name	Dep_ Num
1BM14CS001	Avinash	10
1BM14CS002	Balaji	10
1BM14CS003	Ram	10
1BM14CS004	Mohan	20
1BM14CS005	Nikil	20

Question

1. When the following SQL Command is executed, what will be the output

drop table department cascade constraints;

Department Table will be deleted and foreign key constraint to student Table will be dropped

Schema change statements in SQL

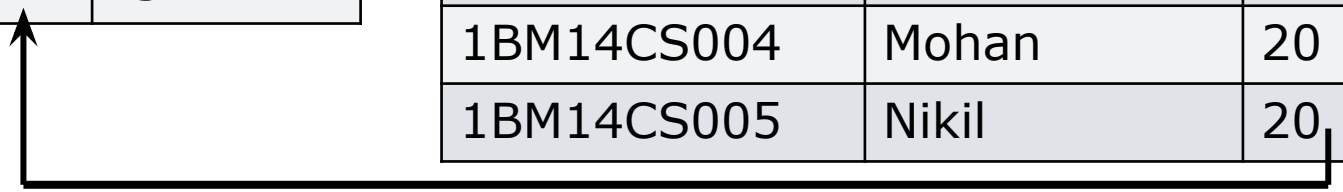
- Drop command

department Table

D_ID	Dep_name
10	CSE
20	ISE

Student Table

USN	Name	Dep_ Num
1BM14CS001	Avinash	10
1BM14CS002	Balaji	10
1BM14CS003	Ram	10
1BM14CS004	Mohan	20
1BM14CS005	Nikil	20



Question

1. Whether the following two SQL commands will be executed successfully

drop table department cascade constraints;
insert into student values ('1BM14CS006','Patel',50);

Schema change statements in SQL

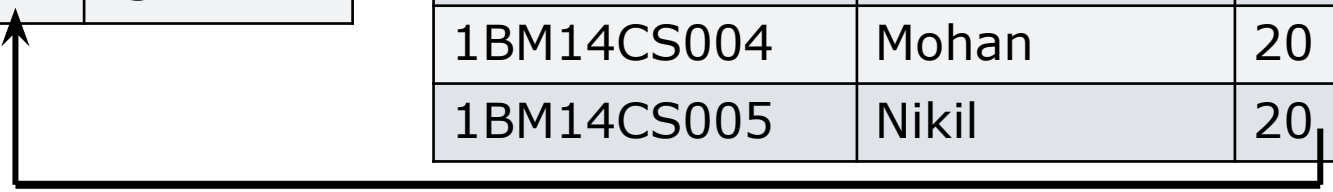
- Drop command

department Table

D_ID	Dep_name
10	CSE
20	ISE

student Table

USN	Name	Dep_ Num
1BM14CS001	Avinash	10
1BM14CS002	Balaji	10
1BM14CS003	Ram	10
1BM14CS004	Mohan	20
1BM14CS005	Nikil	20



Question

1. Whether the following two SQL commands will be executed successfully

drop table department cascade constraints;
insert into student values ('1BM14CS006','Patel',50);

Into the student table new row will be inserted with dep_num 50

Schema change statements in SQL

- Drop Command

General syntax:

drop table table_name [drop_behavior]

There are two drop behavior options

1. **Cascade**: All **constraints** that references the table are **dropped automatically** along with the table itself.
2. **Restrict**: Table is dropped only it is **not referenced** in any constraints.

Schema change statements in SQL

- **Alter** command

General Syntax:

alter table table_name [add|drop|alter] column column_name;

Adding or dropping column

Changing column definition

Adding or dropping table constraints

Schema change statements in SQL

- Alter command

Student Table

USN	**Name**	**Dep_ Num**
1BM14CS001	Avinash	10
1BM14CS002	Balaji	10
1BM14CS003	Ram	10
1BM14CS004	Mohan	20
1BM14CS005	Nikil	20

```
SQL> alter table student drop column Dep_num;
```

Schema change statements in SQL

- Alter command

Student Table

USN	**Name**	**Dep_ Num**
1BM14CS001	Avinash	10
1BM14CS002	Balaji	10
1BM14CS003	Ram	10
1BM14CS004	Mohan	20
1BM14CS005	Nikil	20

```
SQL> alter table student add (Email_ID varchar(40));
```

Schema change statements in SQL

- Alter command

Student Table

USN	**Name**	**Dep_ Num**
1BM14CS001	Akash	10
1BM14CS002	Bharath	10
1BM14CS003	Ragu	10
1BM14CS004	Mohan	20
1BM14CS005	Nikil	20

Question

1. What will be the contents of the student table when the following SQL commands are executed ?

```
SQL> alter table student add email varchar(20);
SQL> insert into student values ('1BM14CS006','Patel',10,'patel@gmail.com');
SQL> select * from student;
```

Schema change statements in SQL

- Alter command

Student Table

USN	S-Name	Dep_Num	email
1BM14CS001	Akash	10	
1BM14CS002	Bharath	10	
1BM14CS003	Ragu	10	
1BM14CS004	Mohan	20	
1BM14CS005	Nikil	20	
1BM14CS006	Patel	10	patel@gmail.com

```
SQL> alter table student add email varchar(20);
SQL> insert into student values ('1BM14CS006','Patel',10,'patel@gmail.com');
SQL> select * from student;
```

Basic queries in SQL

- To retrieve data from database table, basic SQL statement is **SELECT**
- Syntax:

```
select column_name_list
from table_name
where condition;
```

SQL select statement

student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

```
SQL> select * from student;
```

SQL select statement

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

```
SQL> select * from student;
```

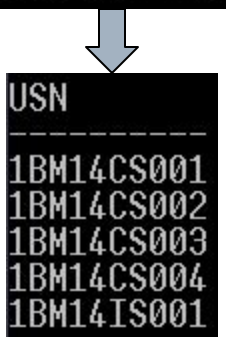
```
USN         NAME                 DEP_NUM      MARKS
----------  -------------------- ---------- ----------
1BM14CS001  Avinash                      10        100
1BM14CS002  Balaji                       10         80
1BM14CS003  Chandan                      10         45
1BM14CS004  Dinesh                       10         60
1BM14IS001  Avinash                      20         90
```

SQL select statement

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

```
SQL> select usn from student;
```



SQL select statement

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

```
SQL> select * from student where name='Avinash';
```

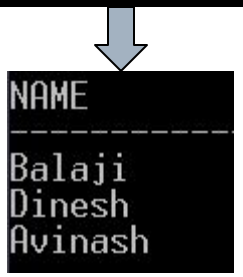
```
USN         NAME                 DEP_NUM      MARKS
---------- -------------------- ---------- ----------
1BM14CS001 Avinash                      10        100
1BM14IS001 Avinash                      20         90
```

SQL select statement

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

```
SQL> select name from student where marks >=60 and marks <=90;
```



Select statement

- Where clause conditions

=

!=

<

>

<=

>=

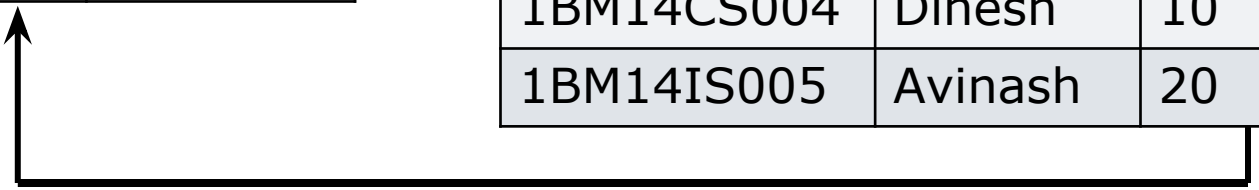
SQL select statement

student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS005	Avinash	20	90

department Table

D_ID	D-Name
10	CSE
20	ISE



1. List out the USN's of the students who belong to department number 10?

SQL select statement

student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS005	Avinash	20	90

department Table

D_ID	D-Name
10	CSE
20	ISE

1. List out the USN's of the students who belong to department number 10?

```
SQL> select usn from student where dep_num=10;

USN
----------
1BM14CS001
1BM14CS002
1BM14CS003
1BM14CS004
```

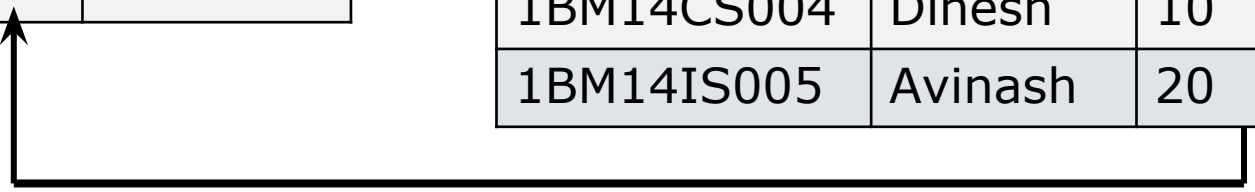
SQL select statement

department Table

d_id	dep_name
10	CSE
20	ISE

student Table

usn	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS005	Avinash	20	90



Question

1. List out the USN's of the students who belong to CSE department ?

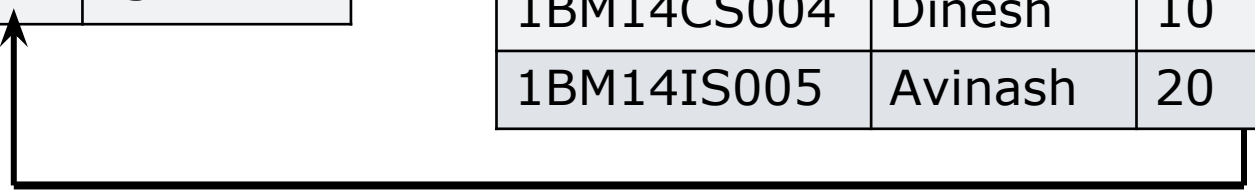
SQL select statement

student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS005	Avinash	20	90

department Table

d_id	dep_na me
10	CSE
20	ISE



Question

1. List out the USN's of the students who belong to CSE department ?

```
SQL> select usn from student,department where student.dep_num=department.d_id
     and department.dep_name='CSE';
```

```
USN
----------
1BM14CS001
1BM14CS002
1BM14CS003
1BM14CS004
```

SQL select statement

student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS005	Avinash	20	90

department Table

d_id	dep_na me
10	CSE
20	ISE

```
SQL> select usn from student,department where dep_num=d_id and dep_name='CSE';
```

```
USN
----------
1BM14CS001
1BM14CS002
1BM14CS003
1BM14CS004
```

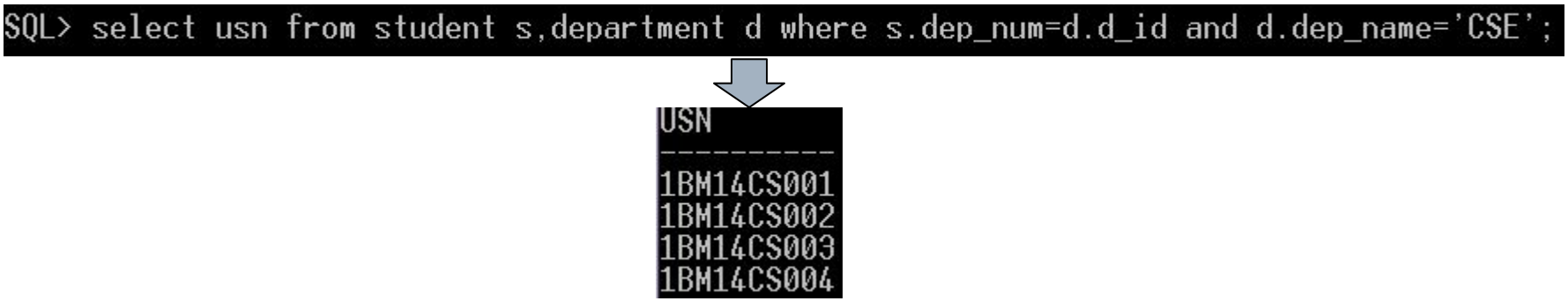
SQL select statement: Aliasing

department Table

d_id	dep_name
10	CSE
20	ISE

student Table

usn	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS005	Avinash	20	90



SQL select statement: Distinct

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

```
SQL> select name from student;
```

Avinash
Balaji
Chandan
Dinesh
Avinash

SQL select statement: Distinct

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

```
SQL> select distinct name from student;
```

Avinash
Balaji
Chandan
Dinesh

SQL select statement: Order By

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

```
SQL> select usn,name,marks from student order by marks asc;
```

```
USN        NAME                 MARKS
---------- ------------------ ----------
1BM14CS003 Chandan                  45
1BM14CS004 Dinesh                   60
1BM14CS002 Balaji                   80
1BM14IS001 Avinash                  90
1BM14CS001 Avinash                 100
```

SQL select statement: Order By

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

```
SQL> select usn,name,marks from student order by marks desc;
```

```
USN         NAME                 MARKS
----------  -------------------- ----------
1BM14CS001  Avinash                 100
1BM14IS001  Avinash                  90
1BM14CS002  Balaji                   80
1BM14CS004  Dinesh                   60
1BM14CS003  Chandan                  45
```

SQL select statement: Order By

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

Question

List usn, name of the students who belong to department number 10 ordered by ascending order of their marks ?

SQL select statement: Order By

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

Question

List usn, name of the students who belong to department number 10 ordered by ascending order of their marks ?

```
SQL> select usn,name,marks from student where dep_num=10 order by marks asc;
```

```
USN        NAME                 MARKS
---------- -------------------- ----------
1BM14CS003 Chandan                  45
1BM14CS004 Dinesh                   60
1BM14CS002 Balaji                   80
1BM14CS001 Avinash                 100
```

Activity: To do

Employee table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-Dec-80	800		20
7499	ALLEN	SALESMAN	7698	20-Feb-81	1600	300	30
7521	WARD	SALESMAN	7698	22-Feb-81	1250	500	30
7566	JONES	MANAGER	7839	02-Apr-81	2975		20
7654	MARTIN	SALESMAN	7698	28-Sep-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-May-81	2850		30
7782	CLARK	MANAGER	7839	09-Jun-81	2450		10
7788	SCOTT	ANALYST	7566	09-Dec-82	3000		20
7839	KING	PRESIDENT		17-Nov-81	5000		10
7844	TURNER	SALESMAN	7698	08-Sep-81	1500	0	30
7876	ADAMS	CLERK	7788	12-Jan-83	1100		20
7900	JAMES	CLERK	7698	03-Dec-81	950		30
7902	FORD	ANALYST	7566	03-Dec-81	3000		20
7934	MILLER	CLERK	7782	23-Jan-82	1300		10

Write SQL queries for the following

1. Display all the information of the Employee table ?
2. Display unique Jobs from Employee table?
3. List names of the employees in the ascending order of their Salaries ?

Topics Covered in Today's class

Unit 1: Basic queries in SQL

w.r.t SELECT statement
SQL operators: LIKE, IN, BETWEEN

Structured Query Language (SQL)

SQL can be divided into two parts:

1. The Data Manipulation Language (DML) and
2. The Data Definition Language (DDL)

1. DDL statements in SQL are:

- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table
- DROP TABLE - deletes a table

2. DML part of SQL:

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database

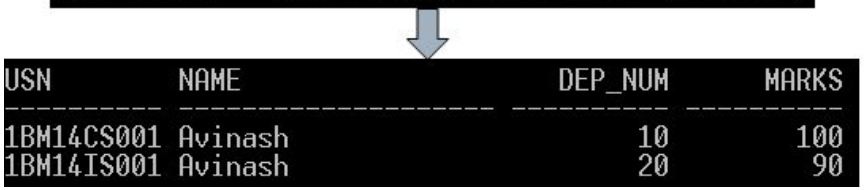
The SQL SELECT Statement

- The SELECT statement is used to select data from a database.
- The result is stored in a result table, called the result-set.

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

```
SQL> select * from student where name='Avinash';
```



```
USN         NAME                 DEP_NUM      MARKS
----------  -------------------- ---------- ----------
1BM14CS001  Avinash                      10        100
1BM14IS001  Avinash                      20         90
```

- Note: **SQL is not case sensitive**. SELECT is the same as select.

w.r.t to SELECT SQL statement

1. SQL Alias: We can give a table or a column another name by using an alias.

- **SQL Alias Syntax for Tables**

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

- **SQL Alias Syntax for Columns**

```
SELECT column_name AS alias_name
FROM table_name;
```

2. The **DISTINCT** keyword can be used to return only distinct (different) values.

- **SQL SELECT DISTINCT Syntax**

```
SELECT DISTINCT column_name(s)
FROM table_name
```

w.r.t to SELECT sql statement

3. The **ORDER BY** keyword is used to sort the result-set by a specified column.

- **SQL ORDER BY Syntax**

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC | DESC;
```

4. The **WHERE** clause is used to filter records. The WHERE clause is used to extract only those records that fulfill a specified criterion.

- **SQL WHERE Syntax**

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator value
```

- With the WHERE clause, the following operators can be used:

Operator	Description
=	Equal
!=	Not Equal
>	Greater than
<	Less than
>=	Greater than or equal
<	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	If you know the exact value you want to return for at least one of the columns

Substring Pattern Matching: SQL LIKE Operator

- The LIKE operator is used in a WHERE clause **to search** for a **specified pattern** in a column.
- SQL LIKE Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern
```

SQL LIKE Operator

- The **LIKE** operator is used in a WHERE clause **to search** for a **specified pattern** in a column.
- SQL LIKE Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern
```

Example: Write SQL statement to list the names starting with letter "A" from following student table.

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Avinash	20	90

SQL LIKE Operator

Write SQL statement to list the names starting with letter "A" from the following student table.

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

```
SQL> select name from student where name like 'A%';
```

SQL LIKE Operator

Write SQL statement to list the names starting with letter "A" from the following student table.

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

```
SQL> select name from student where name like 'A%';
```

```
NAME
--------
Avinash
Arvind
```

Note:
Wildcard character **%**, A substitute for **zero or more characters**

SQL LIKE Operator

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question
Write SQL statement to list name of the students whose names **end** with letter '**h**'?

Note:
Wildcard character **%**, A substitute for **zero or more characters**

SQL LIKE Operator

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question

Write SQL statement to list name of the students whose names end with letter '**h**'?

```
SQL> select name from student where name like '%h';

NAME
--------
Avinash
Dinesh
```

SQL LIKE Operator

Student Table

usn	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question
Write SQL statement to list name of the students whose names are having the substring '**in**'?

Note:
Wildcard character **%**, A substitute for **zero or more characters**

SQL LIKE Operator

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question

Write SQL statement to list name of the students whose names are having the substring '**in**'?

```
SQL> select name from student where name like '%in%';

NAME
---------
Avinash
Dinesh
Arvind
```

SQL LIKE Operator

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question
Write SQL statement to list name of the students whose names start with letter 'A' or 'B'

SQL LIKE Operator

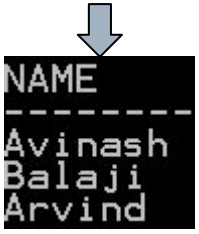
Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question

Write SQL statement to list name of the students whose names start with letter 'A' or 'B'

```
SQL> select name from student where name like 'A%' or name like 'B%';
```



SQL LIKE Operator

Student Table

usn	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question

Write SQL statement to list name of the students whose names start with letter 'A' or 'D' but end with letter 'h'

SQL LIKE Operator

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question

Write SQL statement to list name of the students whose names start with letter 'A' or 'D' but end with letter 'h'

```
SQL> select name from student where name like 'A%h' or name like 'D%h';
```

```
NAME
---------
Avinash
Dinesh
```

SQL LIKE Operator

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question
Write SQL statement to list name of the students whose third letter in the name is '**a**' .

SQL LIKE Operator

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Question

Write SQL statement to list name of the students whose third letter in the name is '**a**' .

```
SQL> select name from student where name like '__a%';
```

```
NAME
--------
Chandan
```

Note:

An **underscore** (_) in the pattern matches exactly **one character**

A **percent sign** (%) in the pattern can match **zero or more characters**

underscore (_) and **percent sign** (%) are referred as wildcard characters

SQL Wildcard Characters

- In SQL, wildcard characters are used with the SQL LIKE operator.
- SQL wildcards are used to search for data within a table.

With SQL, the wildcards are:

Wildcard	**Description**
%	A substitute for zero or more characters
_	A substitute for a single character
[*charlist*]	Sets and ranges of characters to match
[^*charlist*] or [!*charlist*]	Matches only a character NOT specified within the brackets

SQL IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.

SQL IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...)
```

SQL IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.

SQL IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...)
```

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Example: List USN's of the students with name equal to "Avinash" or "Dinesh" from the table above.

SQL IN Operator

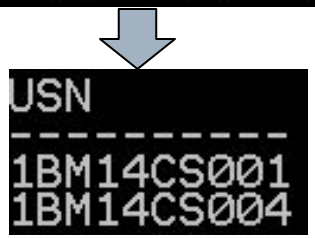
- The IN operator allows you to specify multiple values in a WHERE clause.

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Example: List USN's of the students with name equal to "Avinash" or "Dinesh" from the table above.

```
SQL> select usn from student where name in ('Avinash','Dinesh');
```



SQL BETWEEN operator

- The BETWEEN operator selects a range of data between two values. The values can be numbers, text, or dates.

SQL BETWEEN Syntax

SELECT column_name(s)

FROM table_name

WHERE column_name **BETWEEN** value1 AND value2

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Example: List USN's and Names of students whose marks is in between 40 and 80

SQL BETWEEN operator

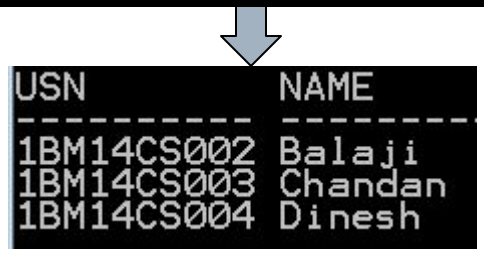
- The BETWEEN operator selects a range of data between two values. The values can be numbers, text, or dates.

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Example: List USN's and Names of students whose marks is in between 40 and 80

```
SQL> select usn,name from student where marks between 40 and 80;
```



```
USN         NAME
----------  --------
1BM14CS002  Balaji
1BM14CS003  Chandan
1BM14CS004  Dinesh
```

Activity To Do

Consider table of Nobel prize winners:
nobel(yr, subject, winner)

nobel

yr	subject	winner
1960	Chemistry	Willard F. Libby
1960	Literature	Saint-John Perse
1960	Medicine	Sir Frank Macfarlane Burnet
1960	Medicine	Peter Medawar
1960	Physics	Donald A. Glaser
1960	Peace	Albert Lutuli
...		

Answer the following

1. Pick the code which shows the name of winner's names beginning with C and ending in n

```
SELECT name FROM nobel
 WHERE winner LIKE '%C%' AND winner LIKE '%n%'
```

```
SELECT name FROM nobel
 WHERE winner LIKE '%C' AND winner LIKE 'n%'
```

```
SELECT name FROM nobel
 WHERE winner LIKE 'C%' AND winner LIKE '%n'
```

```
SELECT winner FROM nobel
 WHERE winner LIKE '%C' AND winner LIKE 'n%'
```

```
SELECT winner FROM nobel
 WHERE winner LIKE 'C%' AND winner LIKE '%n'
```

Activity To Do

Consider table of Nobel prize winners:
nobel(yr, subject, winner)

nobel

yr	subject	winner
1960	Chemistry	Willard F. Libby
1960	Literature	Saint-John Perse
1960	Medicine	Sir Frank Macfarlane Burnet
1960	Medicine	Peter Medawar
1960	Physics	Donald A. Glaser
1960	Peace	Albert Lutuli
...		

Answer the following

1. Pick the code which shows the name of winner's names beginning with C and ending in n

```
SELECT name FROM nobel
 WHERE winner LIKE '%C%' AND winner LIKE '%n%'
```

```
SELECT name FROM nobel
 WHERE winner LIKE '%C' AND winner LIKE 'n%'
```

```
SELECT name FROM nobel
 WHERE winner LIKE 'C%' AND winner LIKE '%n'
```

```
SELECT winner FROM nobel
 WHERE winner LIKE '%C' AND winner LIKE 'n%'
```

```
SELECT winner FROM nobel
 WHERE winner LIKE 'C%' AND winner LIKE '%n'
```

Activity To Do

Consider table of Nobel prize winners:
nobel(yr, subject, winner)

Answer the following

nobel

yr	subject	winner
1960	Chemistry	Willard F. Libby
1960	Literature	Saint-John Perse
1960	Medicine	Sir Frank Macfarlane Burnet
1960	Medicine	Peter Medawar
1960	Physics	Donald A. Glaser
1960	Peace	Albert Lutuli
...		

2. Select the code that shows how many Chemistry awards were given between 1950 and 1960

```
SELECT COUNT(subject) FROM nobel
 WHERE subject = 'Chemistry'
   AND BETWEEN 1950 AND 1960
```

```
SELECT COUNT(subject) FROM nobel
 WHERE subject = 'Chemistry'
   AND yr BETWEEN (1950, 1960)
```

```
SELECT COUNT(subject) FROM nobel
 WHERE subject = 'Chemistry'
   AND yr BETWEEN 1950 AND 1960
```

```
SELECT subject FROM nobel
 WHERE subject = 'Chemistry'
   AND yr BETWEEN 1950 AND 1960
```

```
SELECT subject FROM nobel
 WHERE subject = 'Chemistry'
   AND yr BETWEEN (1950, 1960)
```

Activity To Do

Consider table of Nobel prize winners:
nobel(yr, subject, winner)

nobel

yr	subject	winner
1960	Chemistry	Willard F. Libby
1960	Literature	Saint-John Perse
1960	Medicine	Sir Frank Macfarlane Burnet
1960	Medicine	Peter Medawar
1960	Physics	Donald A. Glaser
1960	Peace	Albert Lutuli
...		

Answer the following

2. Select the code that shows how many Chemistry awards were given between 1950 and 1960

```
SELECT COUNT(subject) FROM nobel
 WHERE subject = 'Chemistry'
   AND BETWEEN 1950 AND 1960
```

```
SELECT COUNT(subject) FROM nobel
 WHERE subject = 'Chemistry'
   AND yr BETWEEN (1950, 1960)
```

```
SELECT COUNT(subject) FROM nobel
 WHERE subject = 'Chemistry'
   AND yr BETWEEN 1950 AND 1960
```

```
SELECT subject FROM nobel
 WHERE subject = 'Chemistry'
   AND yr BETWEEN 1950 AND 1960
```

```
SELECT subject FROM nobel
 WHERE subject = 'Chemistry'
   AND yr BETWEEN (1950, 1960)
```

Aggregate Functions in SQL

- Why we need aggregate functions ??
- Example say we want find maximum marks scored by the students in the class.

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

- We use aggregate function to group multiple rows together to form a single value output.

Topics Covered in Todays Class

Unit 1: Basic queries in SQL

Aggregate functions
Group BY Clause
Having Clause

Arithmetic operators in Queries

- Arithmetic operators for addition (+), subtraction (-), multiplication(*), and division (/) can be applied to numeric values or attributes with numeric domain.

- Example:

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	arvind	20	90

```
SQL> select usn,name,10+marks as total from student;
```

```
USN         NAME                 TOTAL
----------  -------------------- ----------
1BM14CS001  Avinash                 110
1BM14CS002  Balaji                   90
1BM14CS003  Chandan                  55
1BM14CS004  Dinesh                   70
1BM14IS001  arvind                  100
```

Aggregate Functions in SQL

- SQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

- AVG() - Returns the average value
- COUNT() - Returns the number of rows
- MAX() - Returns the largest value
- MIN() - Returns the smallest value
- SUM() - Returns the sum

Aggregate Functions in SQL

The AVG() Function

- The AVG() function returns the average value of a numeric column.
- **SQL AVG() Syntax**

SELECT AVG(column_name) FROM table_name

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Find average marks of all the students in the class.

```
SQL> select avg(marks) from student;

AVG(MARKS)
----------
        75
```

Aggregate Functions in SQL

- **SQL COUNT() Function**

The COUNT() function returns the number of rows that matches a specified criteria.

- **SQL COUNT(column_name) Syntax**

The COUNT(column_name) function returns the number of values (NULL values will not be counted) of the specified column:

```
SELECT COUNT(column_name) FROM table_name
```

- **SQL COUNT(*) Syntax**

The COUNT(*) function returns the number of records in a table:

```
SELECT COUNT(*) FROM table_name
```

- **SQL COUNT(DISTINCT column_name) Syntax**

The COUNT(DISTINCT column_name) function returns the number of distinct values of the specified column:

```
SELECT COUNT(DISTINCT column_name) FROM table_name
```

Aggregate Functions in SQL

- **SQL COUNT(*) Syntax**

The COUNT(*) function returns the number of records in a table.

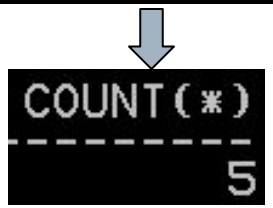
.

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Find total number of records in the above student table

```
SQL> select count(*) from student;
```



```
  COUNT(*)
----------
         5
```

Aggregate Functions in SQL

- **SQL COUNT(*) Syntax**

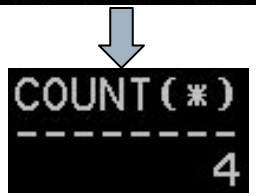
The COUNT(*) function returns the number of records in a table.

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Find total number of students who belong to department number **10**

```
SQL> select count(*) from student where dep_num=10;
```



```
COUNT(*)
--------
       4
```

Aggregate Functions in SQL

- **SQL COUNT(column_name) Syntax**

The COUNT(column_name) function returns the number of values (NULL values will not be counted) of the specified column.

student1 Table

usn	name	dep_ num	marks	emailid
1BM14CS001	Avinash	10	100	avinash@gmail.com
1BM14CS002	Balaji	10	80	
1BM14CS003	Chandan	10	45	chandan@gmail.com
1BM14CS004	Dinesh	10	60	
1BM14IS001	Arvind	20	90	arvind@gmail.com

Find total number of students who have Email IDs

```
SQL> select count(emailid) from student1;
```

```
COUNT(EMAILID)
--------------
             3
```

Aggregate Functions in SQL

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

What will be the out put of following SQL queries ?

```
SQL> select count(distinct dep_num) from student;
```

```
SQL> select count(dep_num) from student;
```

```
SQL> select min(*) from student;
```

```
SQL> select min(marks) from student;
```

Aggregate Functions in SQL

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

```
SQL> select count(distinct dep_num) from student;

COUNT(DISTINCTDEP_NUM)
----------------------
                     2

SQL> select count(dep_num) from student;

COUNT(DEP_NUM)
--------------
             5

SQL> select min(*) from student;
select min(*) from student
           *
ERROR at line 1:
ORA-00936: missing expression


SQL> select min(marks) from student;

MIN(MARKS)
----------
        45
```

SQL GROUP BY Statement

- Why we need Group by Statement
- Say we want to find total number of students by department wise.

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

- Partition the set of records into groups based on certain criteria.
- Groups are formed on the basis of certain attribute.
- Aggregate functions are calculated for each group.

SQL GROUP BY Statement

- The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.
- **SQL GROUP BY Syntax**

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
```

SQL GROUP BY Statement

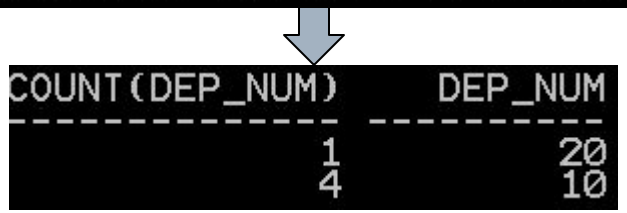
- The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Find total number of students in each department

```
SQL> select count(dep_num),dep_num from student group by dep_num;
```



```
COUNT(DEP_NUM)    DEP_NUM
--------------  ----------
             1          20
             4          10
```

SQL GROUP BY Statement

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Whether the following SQL queries will "Find total number of students in each department"

```
SQL> select count(*),dep_num from student;
```

```
SQL> select count(*),dep_num from student group by dep_num;
```

SQL GROUP BY Statement

Student Table

usn	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

Whether the following SQL queries will "Find total number of students in each department"

```
SQL> select count(*),dep_num from student;
select count(*),dep_num from student
                *
ERROR at line 1:
ORA-00937: not a single-group group function


SQL> select count(*),dep_num from student group by dep_num;

  COUNT(*)    DEP_NUM
---------- ----------
         1         20
         4         10
```

SQL GROUP BY Statement

- The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

What will be the output of following SQL statement

```
SQL> select count(dep_num),dep_num from student
     group by dep_num
     order by dep_num asc;
```

SQL GROUP BY Statement

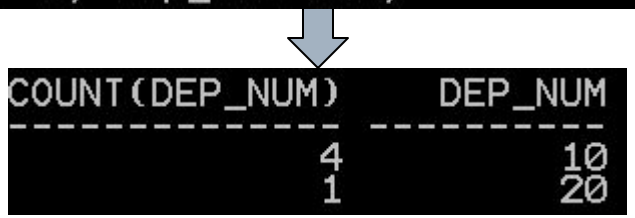
- The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

What will be the output of following SQL statement

```
SQL> select count(dep_num),dep_num from student
     group by dep_num
     order by dep_num asc;
```



Answer the following

7. Pick the result that would be obtained from the following code:

```
SELECT subject, COUNT(subject)
  FROM nobel
 WHERE yr ='1960'
 GROUP BY subject
```

1
1
2
1
1

Chemistry	6

Chemistry	3
Literature	1
Medicine	2
Peace	0
Physics	2

Chemistry	1
Literature	1
Medicine	2
Peace	1
Physics	1

nobel

yr	subject	winner
1960	Chemistry	Willard F. Libby
1960	Literature	Saint-John Perse
1960	Medicine	Sir Frank Macfarlane Burnet
1960	Medicine	Peter Medawar
1960	Physics	Donald A. Glaser
1960	Peace	Albert Lutuli
...		

Answer the following

7. Pick the result that would be obtained from the following code:

```
SELECT subject, COUNT(subject)
  FROM nobel
 WHERE yr ='1960'
 GROUP BY subject
```

1
1
2
1
1

Chemistry	6

Chemistry	3
Literature	1
Medicine	2
Peace	0
Physics	2

Chemistry	1
Literature	1
Medicine	2
Peace	1
Physics	1

nobel

yr	subject	winner
1960	Chemistry	Willard F. Libby
1960	Literature	Saint-John Perse
1960	Medicine	Sir Frank Macfarlane Burnet
1960	Medicine	Peter Medawar
1960	Physics	Donald A. Glaser
1960	Peace	Albert Lutuli
...		

SQL HAVING Clause

- Why we need Having clause ??
- Say we want to find USNs of the students who have registered for atleast two courses

Courses_registered1 table

usn	student_ name	c_id
1BM14CS001	Avinash	10
1BM14CS002	Balaji	10
1BM14CS003	Chandan	20
1BM14CS001	Avinash	20
1BM14CS002	Balaji	30
1BM14CS001	Avinash	30
1BM14CS004	Dinesh	20
1BM14CS004	Dinesh	30
1BM14CS005	Mahesh	30

Course table

C_id	C_name
10	DBMS
20	Java
30	OS

SQL HAVING Clause

- The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

SQL HAVING Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value
```

SQL HAVING Clause

Courses_registered1 table

usn	student_ name	c_id
1BM14CS001	Avinash	10
1BM14CS002	Balaji	10
1BM14CS003	Chandan	20
1BM14CS001	Avinash	20
1BM14CS002	Balaji	30
1BM14CS001	Avinash	30
1BM14CS004	Dinesh	20
1BM14CS004	Dinesh	30
1BM14CS005	Mahesh	30

Course table

C_id	C_name
10	DBMS
20	Java
30	OS

List USNs of the students who have registered for atleast two courses

SQL HAVING Clause

Courses_registered1 table

usn	student_ name	c_id
1BM14CS001	Avinash	10
1BM14CS002	Balaji	10
1BM14CS003	Chandan	20
1BM14CS001	Avinash	20
1BM14CS002	Balaji	30
1BM14CS001	Avinash	30
1BM14CS004	Dinesh	20
1BM14CS004	Dinesh	30
1BM14CS005	Mahesh	30

Course table

C_id	C_name
10	DBMS
20	Java
30	OS

List USNs of the students who have registered for atleast two courses

```
SQL> select usn from courses_registered1 group by usn having count(c_id) >=2;
```

SQL HAVING Clause

Courses_registered table

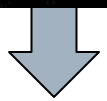
usn	c_id
1BM14CS001	10
1BM14CS002	10
1BM14CS003	20
1BM14CS001	20
1BM14CS002	30
1BM14CS001	30

Course table

C_id	C_name
10	DBMS
20	Java
30	OS

What will be the out put of the following SQL query

```
SQL> select usn,count(c_id) from courses_registered
     group by usn
     having count(c_id) >= 2;
```



SQL HAVING Clause

Courses_registered table

usn	c_id
1BM14CS001	10
1BM14CS002	10
1BM14CS003	20
1BM14CS001	20
1BM14CS002	30
1BM14CS001	30

Course table

C_id	C_name
10	DBMS
20	Java
30	OS

What will be the out put of the following SQL query

```
SQL> select usn,count(c_id) from courses_registered
     group by usn
     having count(c_id) >= 2;
```

```
USN         COUNT(C_ID)
---------- -----------
1BM14CS001            3
1BM14CS002            2
```

SQL HAVING Clause

Courses_registered1 table

usn	student_ name	c_id
1BM14CS001	Avinash	10
1BM14CS002	Balaji	10
1BM14CS003	Chandan	20
1BM14CS001	Avinash	20
1BM14CS002	Balaji	30
1BM14CS001	Avinash	30
1BM14CS004	Dinesh	20
1BM14CS004	Dinesh	30
1BM14CS005	Mahesh	30

Course table

C_id	C_name
10	DBMS
20	Java
30	OS

- List number of courses registered and Names of the student whose names end with ‘h’ and having registered for at least two courses

SQL HAVING Clause

Courses_registered1 table

usn	student_ name	c_id
1BM14CS001	Avinash	10
1BM14CS002	Balaji	10
1BM14CS003	Chandan	20
1BM14CS001	Avinash	20
1BM14CS002	Balaji	30
1BM14CS001	Avinash	30
1BM14CS004	Dinesh	20
1BM14CS004	Dinesh	30
1BM14CS005	Mahesh	30

Course table

C_id	C_name
10	DBMS
20	Java
30	OS

- List number of courses registered and Names of the student whose names end with 'h' and having registered for at least two courses

```
SQL> select student_name,count(c_id) from courses_registered1
     where student_name like '%h'
     group by student_name
     having count(c_id) >=2;
```

```
STUDENT_NAME         COUNT(C_ID)
-------------------- -----------
Avinash                        3
Dinesh                         2
```

SQL HAVING Clause

Courses_registered1 table

usn	student_ name	c_id
1BM14CS001	Avinash	10
1BM14CS002	Balaji	10
1BM14CS003	Chandan	20
1BM14CS001	Avinash	20
1BM14CS002	Balaji	30
1BM14CS001	Avinash	30
1BM14CS004	Dinesh	20
1BM14CS004	Dinesh	30
1BM14CS005	Mahesh	30

Course table

C_id	C_name
10	DBMS
20	Java
30	OS

What will be the out put of the following SQL query

```
SQL> select student_name,count(c_id) from courses_registered1
     where student_name like '%h'
     group by student_name
     having count(c_id) >=2
     order by student_name desc;
```

```
STUDENT_NAME         COUNT(C_ID)
-------------------- -----------
Dinesh                         2
Avinash                        3
```

Remember the following steps to a complete understanding of SQL:

- FROM generates the data set
- WHERE filters the generated data set
- GROUP BY aggregates the filtered data set
- HAVING filters the aggregated data set
- SELECT transforms the filters aggregated data set
- ORDER BY sorts the transformed data set

SQL practice Questions

Sample table: employees

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	1987-06-17	AD_PRES	24000.00	0.00	0	90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	1987-06-18	AD_VP	17000.00	0.00	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	1987-06-19	AD_VP	17000.00	0.00	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	1987-06-20	IT_PROG	9000.00	0.00	102	60
104	Bruce	Ernst	BERNST	590.423.4568	1987-06-21	IT_PROG	6000.00	0.00	103	60

- Write a query to get the total salaries payable to employees.

SQL practice Questions

Sample table: employees

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	1987-06-17	AD_PRES	24000.00	0.00	0	90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	1987-06-18	AD_VP	17000.00	0.00	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	1987-06-19	AD_VP	17000.00	0.00	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	1987-06-20	IT_PROG	9000.00	0.00	102	60
104	Bruce	Ernst	BERNST	590.423.4568	1987-06-21	IT_PROG	6000.00	0.00	103	60

- Write a query to get the total salaries payable to employees.

SELECT SUM(salary) **FROM** employees;

SQL practice Questions

Sample table: employees

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	1987-06-17	AD_PRES	24000.00	0.00	0	90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	1987-06-18	AD_VP	17000.00	0.00	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	1987-06-19	AD_VP	17000.00	0.00	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	1987-06-20	IT_PROG	9000.00	0.00	102	60
104	Bruce	Ernst	BERNST	590.423.4568	1987-06-21	IT_PROG	6000.00	0.00	103	60

- Write a query to list the number of job ids available in the employees table.

SQL practice Questions

Sample table: employees

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	1987-06-17	AD_PRES	24000.00	0.00	0	90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	1987-06-18	AD_VP	17000.00	0.00	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	1987-06-19	AD_VP	17000.00	0.00	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	1987-06-20	IT_PROG	9000.00	0.00	102	60
104	Bruce	Ernst	BERNST	590.423.4568	1987-06-21	IT_PROG	6000.00	0.00	103	60

- Write a query to list the number of job ids available in the employees table.

SELECT COUNT(**DISTINCT** job_id)
FROM employees;

SQL practice Questions

Sample table: employees

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	1987-06-17	AD_PRES	24000.00	0.00	0	90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	1987-06-18	AD_VP	17000.00	0.00	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	1987-06-19	AD_VP	17000.00	0.00	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	1987-06-20	IT_PROG	9000.00	0.00	102	60
104	Bruce	Ernst	BERNST	590.423.4568	1987-06-21	IT_PROG	6000.00	0.00	103	60

Write a query to get the maximum salary of an employee working as a IT_PROG.

SQL practice Questions

Sample table: employees

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	1987-06-17	AD_PRES	24000.00	0.00	0	90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	1987-06-18	AD_VP	17000.00	0.00	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	1987-06-19	AD_VP	17000.00	0.00	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	1987-06-20	IT_PROG	9000.00	0.00	102	60
104	Bruce	Ernst	BERNST	590.423.4568	1987-06-21	IT_PROG	6000.00	0.00	103	60

Write a query to get the maximum salary of an employee working as a IT_PROG.

```
SELECT MAX(salary)
FROM employees
WHERE job_id = 'IT_PROG';
```

SQL practice Questions

Sample table: employees

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	1987-06-17	AD_PRES	24000.00	0.00	0	90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	1987-06-18	AD_VP	17000.00	0.00	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	1987-06-19	AD_VP	17000.00	0.00	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	1987-06-20	IT_PROG	9000.00	0.00	102	60
104	Bruce	Ernst	BERNST	590.423.4568	1987-06-21	IT_PROG	6000.00	0.00	103	60

Write a query to get the department ID and the total salary payable in each department.

SQL practice Questions

Sample table: employees

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	1987-06-17	AD_PRES	24000.00	0.00	0	90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	1987-06-18	AD_VP	17000.00	0.00	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	1987-06-19	AD_VP	17000.00	0.00	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	1987-06-20	IT_PROG	9000.00	0.00	102	60
104	Bruce	Ernst	BERNST	590.423.4568	1987-06-21	IT_PROG	6000.00	0.00	103	60

Write a query to get the department ID and the total salary payable in each department.

```
SELECT department_id, SUM(salary)
FROM employees
GROUP BY department_id;
```

SQL practice Questions

Sample table: employees

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	1987-06-17	AD_PRES	24000.00	0.00	0	90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	1987-06-18	AD_VP	17000.00	0.00	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	1987-06-19	AD_VP	17000.00	0.00	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	1987-06-20	IT_PROG	9000.00	0.00	102	60
104	Bruce	Ernst	BERNST	590.423.4568	1987-06-21	IT_PROG	6000.00	0.00	103	60

Write a query to get the average salary for each job ID excluding IT_PROG.

SQL practice Questions

Sample table: employees

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	1987-06-17	AD_PRES	24000.00	0.00	0	90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	1987-06-18	AD_VP	17000.00	0.00	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	1987-06-19	AD_VP	17000.00	0.00	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	1987-06-20	IT_PROG	9000.00	0.00	102	60
104	Bruce	Ernst	BERNST	590.423.4568	1987-06-21	IT_PROG	6000.00	0.00	103	60

Write a query to get the average salary for each job ID excluding IT_PROG.

```
SELECT job_id, AVG(salary)
FROM employees
WHERE job_id != 'IT_PROG'
GROUP BY job_id;
```

SQL practice Questions

Sample table: employees

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	1987-06-17	AD_PRES	24000.00	0.00	0	90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	1987-06-18	AD_VP	17000.00	0.00	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	1987-06-19	AD_VP	17000.00	0.00	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	1987-06-20	IT_PROG	9000.00	0.00	102	60
104	Bruce	Ernst	BERNST	590.423.4568	1987-06-21	IT_PROG	6000.00	0.00	103	60

Write a query to get the average salary for all departments employing more than 10 employees.

SQL practice Questions

Sample table: employees

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	1987-06-17	AD_PRES	24000.00	0.00	0	90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	1987-06-18	AD_VP	17000.00	0.00	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	1987-06-19	AD_VP	17000.00	0.00	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	1987-06-20	IT_PROG	9000.00	0.00	102	60
104	Bruce	Ernst	BERNST	590.423.4568	1987-06-21	IT_PROG	6000.00	0.00	103	60

Write a query to get the average salary for all departments employing more than 10 employees.

```
SELECT AVG(salary), COUNT(employee_id)
FROM employees
GROUP BY department_id
HAVING COUNT(employee_id) > 10;
```

DML statements

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database

The INSERT INTO Statement

- The INSERT INTO statement is used to insert a new row in a table.
- **SQL INSERT INTO Syntax**

It is possible to write the INSERT INTO statement in two forms.

The first form doesn't specify the column names where the data will be inserted, only their values:

```
INSERT INTO table_name
VALUES (value1, value2, value3,...)
```

The second form specifies both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

SQL UPDATE Statement

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

SQL UPDATE Syntax

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

student1 Table

usn	name	dep_ num	marks	emailid
1BM14CS001	Avinash	10	100	avinash@gmail.com
1BM14CS002	Balaji	10	80	
1BM14CS003	Chandan	10	45	chandan@gmail.com
1BM14CS004	Dinesh	10	60	
1BM14IS001	Arvind	20	90	arvind@gmail.com

```
update student1
set emailid='dinesh@gmail.com'
where name='Dinesh';
```

SQL DELETE Statement

- The DELETE statement is used to delete rows in a table.

SQL DELETE Syntax

```
DELETE FROM table_name
WHERE some_column=some_value
```

student1 Table

usn	name	dep_ num	marks	emailid
1BM14CS001	Avinash	10	100	avinash@gmail.com
1BM14CS002	Balaji	10	80	
1BM14CS003	Chandan	10	45	chandan@gmail.com
1BM14CS004	Dinesh	10	60	
1BM14IS001	Arvind	20	90	arvind@gmail.com

```
delete from student1
where usn='1BM14IS001';
```

Topics Covered in Todays Class

Unit 1: More Complex SQL queries

Nested Queries, Tuples and Set/Mutiset Comparison

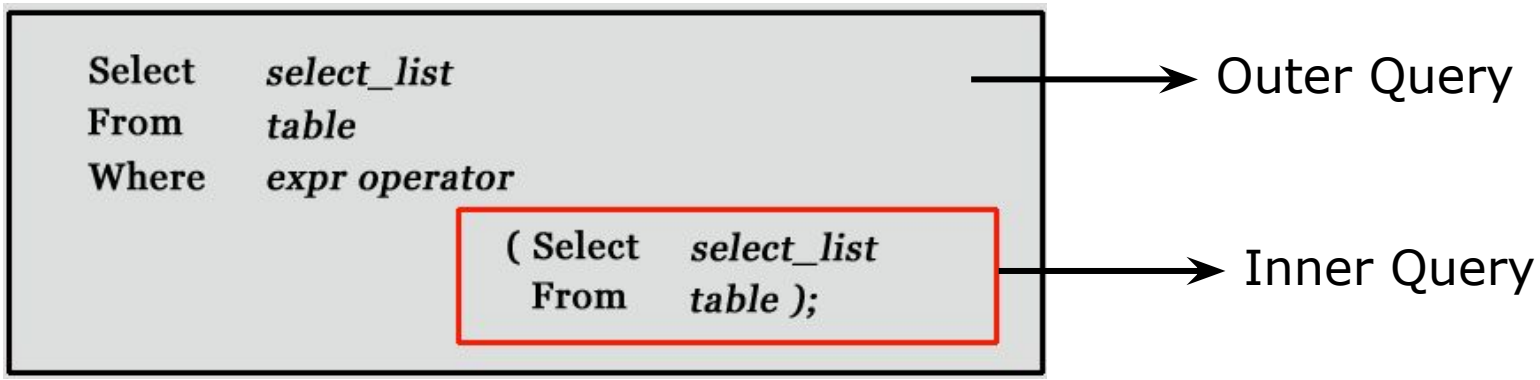
Subqueries or Nested Queries

- A subquery is a SQL query nested inside a larger query.

  SELECT colname1, colname2……. (SELECT …….)

  Note: Subqueries must be enclosed with in parenthesis

- A subquery is usually added within the WHERE Clause of another SQL SELECT statement.
  - We can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, SOME, or ALL.
- A subquery can be treated as an inner query, which is a SQL query placed as a part of another query called as outer query.
- The inner query executes first before its parent query so that the results of inner query can be passed to the outer query.



Subqueries or Nested Queries

- A subquery is a SQL query nested inside a larger query.

  SELECT colname1, colname2……. (SELECT …….)

  Note: Subqueries must be enclosed with in parenthesis

- A subquery is usually added within the WHERE Clause of another SQL SELECT statement.
  - We can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, SOME, or ALL.
- A subquery can be treated as an inner query, which is a SQL query placed as a part of another query called as outer query.
- The inner query executes first before its parent query so that the results of inner query can be passed to the outer query.

```
SELECT column-names
  FROM table-name1
 WHERE value IN (SELECT column-name
                   FROM table-name2
                  WHERE condition)
```

→ Outer Query

→ Inner Query

Subqueries

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

- List names of the students whose marks score is greater than 'Balaji'

Subqueries

Student Table

usn	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

- List names of the students whose marks score is greater than 'Balaji'

Outer Query

Inner Query

```
SQL> select name
     from student
     where marks > (select marks from student where name='Balaji');
```

Subqueries

Student Table

usn	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

- List names of the students whose marks score is greater than 'Balaji'

Outer Query

```
SQL> select name
     from student
     where marks > (select marks from student where name='Balaji');
```

80

Inner Query

Subqueries

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

- List names of the students whose marks score is greater than 'Balaji'

```
SQL> select name
     from student
     where marks > (select marks from student where name='Balaji');
```

```
NAME
---------
Avinash
Arvind
```

Subqueries

Student Table

usn	name	dep_num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

- List name of the student whose marks score is second highest

Subqueries

Student Table

usn	name	dep_ num	marks
1BM14CS001	Avinash	10	100
1BM14CS002	Balaji	10	80
1BM14CS003	Chandan	10	45
1BM14CS004	Dinesh	10	60
1BM14IS001	Arvind	20	90

- List name of the student whose marks score is second highest

```
SQL> select max(marks)
     from student
     where marks != (select max(marks) from student);
```

```
MAX(MARKS)
----------
        90
```

Subqueries : Using Comparisons

- A subquery can be used before or after any of the comparison operators.
- The subquery can return at most one value. The value can be the result of an arithmetic expression or a column function.
- SQL then compares the value that results from the subquery with the value on the other side of the comparison operator. You can use the following comparison operators:

Operator	Description
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
!=	Not equal to

Activity To Do

movie_director(movie_title,pid)

movie_title	pid
Guru	P1
Dasvathaaram	P2
Raavan	P1

person(pid,name)

pid	name
P1	ManiRathnam
P2	KamalHassan

Write Query to,
List down the movie titles directed by 'ManiRathnam'

Activity To Do

movie_director(movie_title,pid)

movie_title	pid
Guru	P1
Dasvathaaram	P2
Raavan	P1

person(pid,name)

pid	name
P1	ManiRathnam
P2	KamalHassan

Write Query to,
List down the movie titles directed by 'ManiRathnam'

```
select movie_title
from movie_director
where pid=(select pid from person where name='ManiRatnam');
```

Activity To Do

movie_director(movie_title,pid)

movie_title	pid
Guru	P1
Dasvathaaram	P2
Raavan	P1

person(pid,name)

pid	name
P1	ManiRathnam
P2	KamalHassan

Write Query to,
List down the movie titles directed by 'ManiRathnam'

```
SQL> select movie_title
     from movie_director m, person p
     where m.pid=p.pid and p.name='ManiRathnam';
```

Subqueries with ALL, ANY, IN, or SOME

- We can use a subquery after a comparison operator, followed by the keyword ALL, ANY, or SOME.
- The ALL operator compares value to every value returned by the subquery. Therefore ALL operator (which must follow a comparison operator) returns TRUE if the comparison is TRUE for ALL of the values in the column that the subquery returns.
- **Syntax :**

operand comparison_operator **ALL** (subquery)

```
SELECT column-names
  FROM table-name
 WHERE column-name operator ALL
       (SELECT column-name
          FROM table-name
         WHERE condition)
```

```
SELECT column-names
  FROM table-name
 WHERE column-name operator ANY
       (SELECT column-name
          FROM table-name
         WHERE condition)
```

Subquery: ALL operator

- **> all** means greater than every value, or greater than the maximum value.

For example, **> all** (1, 2, 3) means greater than 3.

Subquery: ALL operator

- **> all** means greater than every value, or greater than the maximum value.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL
7369	SMITH	CLERK	7902	17-DEC-1980 00:00:00	800
7499	ALLEN	SALESMAN	7698	20-FEB-1981 00:00:00	1600
7521	WARD	SALESMAN	7698	22-FEB-1981 00:00:00	1250
7566	JONES	MANAGER	7839	02-APR-1981 00:00:00	2975
7654	MARTIN	SALESMAN	7698	28-SEP-1981 00:00:00	1250
7698	BLAKE	MANAGER	7839	01-MAY-1981 00:00:00	2850
7782	CLARK	MANAGER	7839	09-JUN-1981 00:00:00	2450
7788	SCOTT	ANALYST	7566	19-APR-1987 00:00:00	3000
7839	KING	PRESIDENT		17-NOV-1981 00:00:00	5000
7844	TURNER	SALESMAN	7698	08-SEP-1981 00:00:00	1500
7876	ADAMS	CLERK	7788	23-MAY-1987 00:00:00	1100
7900	JAMES	CLERK	7698	03-DEC-1981 00:00:00	950
7902	FORD	ANALYST	7566	03-DEC-1981 00:00:00	3000
7934	MILLER	CLERK	7782	23-JAN-1982 00:00:00	1300

```
SELECT empno, sal
FROM   emp
WHERE  sal > ALL (2000, 3000, 4000);
```

EMPNO	SAL
7839	5000

Subquery: ALL operator

- **> all** means greater than every value, or greater than the maximum value.

```
SELECT empno, sal
FROM   emp
WHERE  sal > ALL (2000, 3000, 4000);

     EMPNO        SAL
---------- ----------
      7839       5000
```

```
-- Transformed to equivalent statement without ALL.

SELECT empno, sal
FROM   emp
WHERE  sal > 2000 AND sal > 3000 AND sal > 4000;

     EMPNO        SAL
---------- ----------
      7839       5000
```

Activity To Do

- **List EMPNO and Salary of the employees whose salary is greater than the salary of all employees in the department 20**

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-1980 00:00:00	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-1981 00:00:00	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-1981 00:00:00	1250	500	30
7566	JONES	MANAGER	7839	02-APR-1981 00:00:00	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-1981 00:00:00	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-1981 00:00:00	2850		30
7782	CLARK	MANAGER	7839	09-JUN-1981 00:00:00	2450		10
7788	SCOTT	ANALYST	7566	19-APR-1987 00:00:00	3000		20
7839	KING	PRESIDENT		17-NOV-1981 00:00:00	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-1981 00:00:00	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-1987 00:00:00	1100		20
7900	JAMES	CLERK	7698	03-DEC-1981 00:00:00	950		30
7902	FORD	ANALYST	7566	03-DEC-1981 00:00:00	3000		20
7934	MILLER	CLERK	7782	23-JAN-1982 00:00:00	1300		10

Activity To Do

- **List EMPNO and salary of the employees whose salary is greater than the salary of all employees in the department 20**

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-1980 00:00:00	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-1981 00:00:00	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-1981 00:00:00	1250	500	30
7566	JONES	MANAGER	7839	02-APR-1981 00:00:00	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-1981 00:00:00	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-1981 00:00:00	2850		30
7782	CLARK	MANAGER	7839	09-JUN-1981 00:00:00	2450		10
7788	SCOTT	ANALYST	7566	19-APR-1987 00:00:00	3000		20
7839	KING	PRESIDENT		17-NOV-1981 00:00:00	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-1981 00:00:00	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-1987 00:00:00	1100		20
7900	JAMES	CLERK	7698	03-DEC-1981 00:00:00	950		30
7902	FORD	ANALYST	7566	03-DEC-1981 00:00:00	3000		20
7934	MILLER	CLERK	7782	23-JAN-1982 00:00:00	1300		10

```
SELECT e1.empno, e1.sal
FROM   emp e1
WHERE  e1.sal > ALL (SELECT e2.sal
                     FROM   emp e2
                     WHERE  e2.deptno = 20);

     EMPNO        SAL
---------- ----------
      7839       5000
```

Subquery: ALL operator

- **> all means greater than all values**
- The **> all** operator means that, for a row to satisfy the condition in the outer query, the value in the column that introduces the subquery must be greater than each of the values returned by the subquery.

- "x = ALL (...)": The value must match all the values in the list to evaluate to TRUE.
- "x != ALL (...)": The value must not match any values in the list to evaluate to TRUE.
- "x > ALL (...)": The value must be greater than the biggest value in the list to evaluate to TRUE.
- "x < ALL (...)": The value must be smaller than the smallest value in the list to evaluate to TRUE.
- "x >= ALL (...)": The value must be greater than or equal to the biggest value in the list to evaluate to TRUE.
- "x <= ALL (...)": The value must be smaller than or equal to the smallest value in the list to evaluate to TRUE.

Subquery: ANY operator

- **> any** means greater than at least one value, or greater than the minimum value.
- Therefore, **> any** (1, 2, 3) means greater than 1.

Subquery: ANY operator

- **> any** means greater than at least one value, or greater than the minimum value.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL
7369	SMITH	CLERK	7902	17-DEC-1980 00:00:00	800
7499	ALLEN	SALESMAN	7698	20-FEB-1981 00:00:00	1600
7521	WARD	SALESMAN	7698	22-FEB-1981 00:00:00	1250
7566	JONES	MANAGER	7839	02-APR-1981 00:00:00	2975
7654	MARTIN	SALESMAN	7698	28-SEP-1981 00:00:00	1250
7698	BLAKE	MANAGER	7839	01-MAY-1981 00:00:00	2850
7782	CLARK	MANAGER	7839	09-JUN-1981 00:00:00	2450
7788	SCOTT	ANALYST	7566	19-APR-1987 00:00:00	3000
7839	KING	PRESIDENT		17-NOV-1981 00:00:00	5000
7844	TURNER	SALESMAN	7698	08-SEP-1981 00:00:00	1500
7876	ADAMS	CLERK	7788	23-MAY-1987 00:00:00	1100
7900	JAMES	CLERK	7698	03-DEC-1981 00:00:00	950
7902	FORD	ANALYST	7566	03-DEC-1981 00:00:00	3000
7934	MILLER	CLERK	7782	23-JAN-1982 00:00:00	1300

```
SELECT empno, sal
FROM   emp
WHERE  sal > ANY (2000, 3000, 4000);
```

EMPNO	SAL
7566	2975
7698	2850
7782	2450
7788	3000
7839	5000
7902	3000

Subquery: ANY operator

- **> any** means greater than at least one value, or greater than the minimum value.

```
SELECT empno, sal
FROM   emp
WHERE  sal > ANY (2000, 3000, 4000);

     EMPNO        SAL
---------- ----------
      7566       2975
      7698       2850
      7782       2450
      7788       3000
      7839       5000
      7902       3000
```

```
-- Transformed to equivalent statement without ANY.

SELECT empno, sal
FROM   emp
WHERE  sal > 2000 OR sal > 3000 OR sal > 4000;

     EMPNO        SAL
---------- ----------
      7566       2975
      7698       2850
      7782       2450
      7788       3000
      7839       5000
      7902       3000
```

Activity To Do

- **List EMPNO and salary of the employees whose salary is greater than the salary of any employees in the department 10**

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-1980 00:00:00	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-1981 00:00:00	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-1981 00:00:00	1250	500	30
7566	JONES	MANAGER	7839	02-APR-1981 00:00:00	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-1981 00:00:00	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-1981 00:00:00	2850		30
7782	CLARK	MANAGER	7839	09-JUN-1981 00:00:00	2450		10
7788	SCOTT	ANALYST	7566	19-APR-1987 00:00:00	3000		20
7839	KING	PRESIDENT		17-NOV-1981 00:00:00	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-1981 00:00:00	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-1987 00:00:00	1100		20
7900	JAMES	CLERK	7698	03-DEC-1981 00:00:00	950		30
7902	FORD	ANALYST	7566	03-DEC-1981 00:00:00	3000		20
7934	MILLER	CLERK	7782	23-JAN-1982 00:00:00	1300		10

Activity To Do

- **List EMPNO and salary of the employees whose salary is greater than the salary of any employees in the department 10**

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-1980 00:00:00	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-1981 00:00:00	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-1981 00:00:00	1250	500	30
7566	JONES	MANAGER	7839	02-APR-1981 00:00:00	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-1981 00:00:00	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-1981 00:00:00	2850		30
7782	CLARK	MANAGER	7839	09-JUN-1981 00:00:00	2450		10
7788	SCOTT	ANALYST	7566	19-APR-1987 00:00:00	3000		20
7839	KING	PRESIDENT		17-NOV-1981 00:00:00	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-1981 00:00:00	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-1987 00:00:00	1100		20
7900	JAMES	CLERK	7698	03-DEC-1981 00:00:00	950		30
7902	FORD	ANALYST	7566	03-DEC-1981 00:00:00	3000		20
7934	MILLER	CLERK	7782	23-JAN-1982 00:00:00	1300		10

```
SELECT e1.empno, e1.sal
FROM   emp e1
WHERE  e1.sal > ANY (SELECT e2.sal
                     FROM   emp e2
                     WHERE  e2.deptno = 10);
```

EMPNO	SAL
7839	5000
7902	3000
7788	3000
7566	2975
7698	2850
7782	2450
7499	1600
7844	1500

Subquery: ANY operator

- **> any means greater than at least one value**
- **> any** means that, for a row to satisfy the outer query, the value in the column that introduces the subquery must be greater than at least one of the values in the list returned by the subquery.

- "x = ANY (...)": The value must match one or more values in the list to evaluate to TRUE.
- "x != ANY (...)": The value must not match one or more values in the list to evaluate to TRUE.
- "x > ANY (...)": The value must be greater than the smallest value in the list to evaluate to TRUE.
- "x < ANY (...)": The value must be smaller than the biggest value in the list to evaluate to TRUE.
- "x >= ANY (...)": The value must be greater than or equal to the smallest value in the list to evaluate to TRUE.
- "x <= ANY (...)": The value must be smaller than or equal to the biggest value in the list to evaluate to TRUE.

Subquery: SOME operator

- The SOME and ANY comparison conditions do exactly the same thing and are completely interchangeable.

Subquery: IN Operator

- Syntax

```
SELECT "column_name"
FROM "table_name"
WHERE "column_name" IN ('value1', 'value2', ...);
```

The number of values in the parenthesis can be one or more, with each values separated by comma. Values can be numerical or characters. If there is only one value inside the parenthesis, this command is equivalent to, `WHERE "column_name" = 'value1'`

```
SELECT column-names
  FROM table-name1
 WHERE value IN (SELECT column-name
                   FROM table-name2
                  WHERE condition)
```

Subquery: IN Operator

- Syntax

```
SELECT "column_name"
FROM "table_name"
WHERE "column_name" IN ('value1', 'value2', ...);
```

The number of values in the parenthesis can be one or more, with each values separated by comma. Values can be numerical or characters. If there is only one value inside the parenthesis, this command is equivalent to, `WHERE "column_name" = 'value1'`

```
SELECT column-names
  FROM table-name1
 WHERE value IN (SELECT column-name
                   FROM table-name2
                  WHERE condition)
```

Subquery: IN Operator

Find the names of the publishers who have published CSE books:

publisher

pid	pub_name
1	Pearson
2	Tata
3	ABP
4	GitaPress

titles

pid	type
1	CSE
2	ISE
3	CSE
4	EC

Subquery: IN Operator

Find the names of the publishers who have published CSE books:

publisher

pid	pub_name
1	Pearson
2	Tata
3	ABP
4	GitaPress

titles

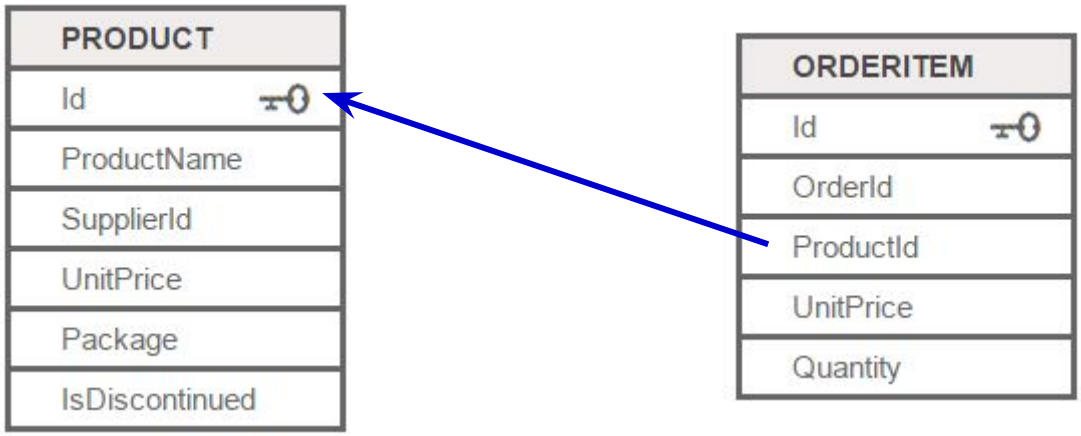
pid	type
1	CSE
2	ISE
3	CSE
4	EC

```
select pub_name
from publisher
where pid in  (select pid
                from titles
                where type = 'CSE')
```

```
pub_name
-----------
Pearson
ABP
```

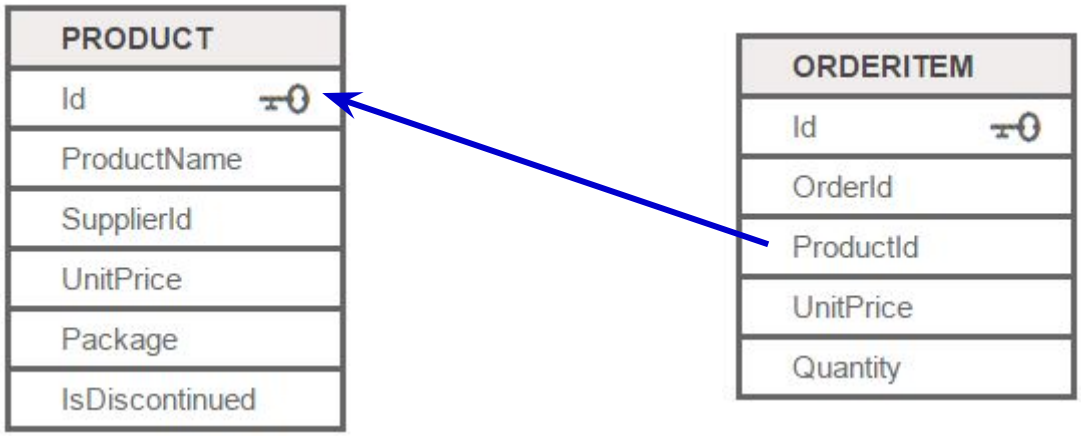
Activity To Do

- **List products with order quantities greater than 100.**



Activity To Do

- **List products with order quantities greater than 100.**




```
SELECT ProductName
  FROM Product
 WHERE Id IN (SELECT ProductId
                FROM OrderItem
               WHERE Quantity > 100)
```

Topics Covered in Todays Class

Unit 1: More Complex SQL queries

Correlated SubQueries
EXISTS operator

Illustration why we need to construct Correlated Sub Queries

- List the details of the item which is having maximum unitprice in **each class**

ITEMID	ITEMNAME	UNITPRICE	CLASS
B1	Long Book	60	Stationary
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

Illustration why we need to construct Correlated Sub Queries

- List the details of the item which is having maximum unitprice in **each class**

Item

ITEMID	ITEMNAME	UNITPRICE	CLASS
B1	Long Book	60	Stationary
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

Class Group 1

Class Group 2

Output

ITEMID	ITEMNAME	UNITPRICE	CLASS
B1	Long Book	60	Stationary
C2	Chocolate IceCream	150	Food

Illustration: Why we need Correlated Sub Queries

- List the maximum unitprice in **each class**

Item

ITEMID	ITEMNAME	UNITPRICE	CLASS
B1	Long Book	60	Stationary
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

Class Group 1

Class Group 2

```
SQL> select max(unitprice) from item group by class;

MAX(UNITPRICE)
--------------
            60
           150
```

Illustration: Why we need Correlated Sub Queries

- List the class name and maximum unitprice in **each class**

Item

ITEMID	ITEMNAME	UNITPRICE	CLASS
B1	Long Book	60	Stationary
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

Class Group 1

Class Group 2

```
SQL> select class, max(unitprice) from item group by class;

CLASS                MAX(UNITPRICE)
-------------------- --------------
Stationary                       60
Food                            150
```

Illustration: Why we need Correlated Sub Queries

- List the class name and maximum unitprice in **each class**

Item

ITEMID	ITEMNAME	UNITPRICE	CLASS
B1	Long Book	60	Stationary
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

Class Group 1

Class Group 2

```
SQL> select class, max(unitprice) from item group by class;

CLASS                MAX(UNITPRICE)
-------------------- --------------
Stationary                       60
Food                            150
```

Illustration: Why we need Correlated Sub Queries

- List the Item name, maximum unitprice and class name in **each class**

Item table

ITEMID	ITEMNAME	UNITPRICE	CLASS
B1	Long Book	60	Stationary
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

Class Group 1

Class Group 2

Output should be

ItemName	max(unitPrice)	Class
Long Book	60	Stationary
Chocolate IceCream	150	Food

Illustration: Why we need Correlated Sub Queries

- List the Item name, maximum unitprice and class name in **each class**

Item table

ITEMID	ITEMNAME	UNITPRICE	CLASS	
B1	Long Book	60	Stationary	**Class Group** 1
B2	Short Book	35	Stationary	
P1	Natraja Pencil	10	Stationary	
P2	Apsara Pencil	5	Stationary	
C1	Vanila IceCream	75	Food	**Class Group** 2
C2	Chocolate IceCream	150	Food	
R1	Fried Rice	80	Food	
R2	Curd Rice	50	Food	

```
SQL> select itemname, class, max(unitprice)
     from item group by class;
```

```
ERROR at line 1:
ORA-00979: not a GROUP BY expression
```

Illustration: Why we need Correlated Sub Queries

- List the Item name, maximum unitprice and class name of the item which is having maximum unitprice in **each class**

Item

ITEMID	ITEMNAME	UNITPRICE	CLASS
B1	Long Book	60	Stationary
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

```
select itemname,
max(unitprice)  unitprice,class
from item
group by  class,itemname;
```

ITEMNAME	UNITPRICE	CLASS
Long Book	60	Stationary
Short Book	35	Stationary
Natraja Pencil	10	Stationary
Apsara Pencil	5	Stationary
Vanila IceCream	75	Food
Chocolate IceCream	150	Food
Fried Rice	80	Food
Curd Rice	50	Food

SQL GROUP BY Statement

- The GROUP BY statement is **used in conjunction with the aggregate functions** to group the result-set by one or more columns.
- **SQL GROUP BY Syntax**

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
[Where .............]
GROUP BY column_name
```

One purpose of grouping is to **display count, averages, sum, minimum, maximum values** for each group. Note that in the result of such a query, we see only **one line per group**.

Illustration: Why we need Correlated Queries

- List the class name and max,min,sum, average Unitprice and number of items in **each class**

Item

ITEMID	ITEMNAME	UNITPRICE	CLASS	
B1	Long Book	60	Stationary	Class Group 1
B2	Short Book	35	Stationary	
P1	Natraja Pencil	10	Stationary	
P2	Apsara Pencil	5	Stationary	
C1	Vanila IceCream	75	Food	Class Group 2
C2	Chocolate IceCream	150	Food	
R1	Fried Rice	80	Food	
R2	Curd Rice	50	Food	

```
SQL> select class,max(unitprice), min(unitprice),
     sum(unitprice),avg(unitprice),count(*)
     from item
     group by class;
```

Class	max(unitprice)	min(unitprice)	sum(unitprice)	avg(unitprice)	count(*)
Stationary	60	5	110	27.5	4
Food	150	50	355	88.75	4

Say, we want to

- List the details of the item which is having maximum unitprice in **each class**

Item

ITEMID	ITEMNAME	UNITPRICE	CLASS
B1	Long Book	60	Stationary
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

Class Group 1

Class Group 2

Output

ITEMID	ITEMNAME	UNITPRICE	CLASS
B1	Long Book	60	Stationary
C2	Chocolate IceCream	150	Food

What is Independent Sub Query or Non Correlated Sub Query ?

- A subquery is a SELECT statement within another statement.

Outer Query

```
SELECT t1.column1
FROM table t1 WHERE t1.column1 IN (SELECT t2.column1
                                   FROM table t2
                                   where t2.column2 >10 );
```

What is Independent Sub Query or Non Correlated Sub Query ?

- A subquery is a SELECT statement within another statement.

Outer Query

```
SELECT t1.column1
FROM table t1 WHERE t1.column1 IN (SELECT t2.column1
                                   FROM table t2
                                   where t2.column2 >10 );
```

Inner Query
Not dependent on columns of Outer Query

What is Correleated Sub Query ?

- A *correlated subquery* is a subquery that contains a reference to a table that also appears in the outer query.

Outer Query

```
SELECT t1.column1
FROM table t1 WHERE t1.column1 IN (SELECT t2.column1
                                   FROM table t2
                                   where t2.column2 >t1.column2);
```

What is Correleated Sub Query ?

- A *correlated subquery* is a subquery that contains a reference to a table that also appears in the outer query.

Outer Query

```
SELECT t1.column1
FROM table t1 WHERE t1.column1 IN (SELECT t2.column1
                                   FROM table t2
                                   where t2.column2 >t1.column2);
```

Inner Query
is dependent on columns of
Outer Query
i.e., t2.colum2=t1.column2

What is Correlated Sub Query

- A correlated subquery is a subquery that contains a reference to a table (in the parent query) that also appears in the outer query. SQL evaluates from inside to outside.
- **Correlated subquery syntax :**

```
SELECT column1, column2, ...
FROM   table1 outerr
WHERE  column1 operator
                    (SELECT   column1, column2
                     FROM     table2
                     WHERE    expr1 =
                                  outerr.expr2);
```

- **The inner Query is executed separately for each row of the outer query.SQL performs a subquery over and over again – once for each row of the main query.**

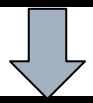
Correlated Queries

- List the details of the item which is having maximum unitprice in each class

Item

ITEMID	ITEMNAME	UNITPRICE	CLASS
B1	Long Book	60	Stationary
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

```
SQL> select outer.itemid, outer.itemname, outer.unitprice,outer.class
     from item outer
     where outer.unitprice = (select max(inner.unitprice) from item inner
     where inner.class=outer.class);
```



ITEMID	ITEMNAME	UNITPRICE	CLASS
B1	Long Book	60	Stationary
C2	Chocolate IceCream	150	Food

Logic of correlated sub queries

Outer item table

itemid	itemname	unitprice	class
B1	Long Book	60	**Stationary**
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

Inner item table

itemid	itemname	unitprice	class
B1	Long Book	60	Stationary
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

Logic of correlated sub queries

Outer item table

itemid	itemname	unitprice	class
B1	Long Book	60	**Stationary**
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

Step 1
For every record of outer query
Pick the class name
Stationary

Inner item table

itemid	itemname	unitprice	class
B1	Long Book	60	Stationary
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

Logic of correlated sub queries

Outer item table

itemid	itemname	unitprice	class
B1	Long Book	60	**Stationary**
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

Step 1
For every record of outer query
Pick the class name
Stationary

Step 2
In the inner query
Find maximum unitprice of class name picked from outer query
i.e compare **class name** of **outer** with **inner**

Inner item table

itemid	itemname	unitprice	class
B1	Long Book	60	Stationary
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

Max
60

Logic of correlated sub queries

Outer item table

itemid	itemname	unitprice	class
B1	Long Book	60	**Stationary**
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

Step 1
For every record of outer query
Pick the class name
Stationary

Step 2
In the inner query
Find maximum unitprice of class name picked from outer query
i.e compare **class name** of **outer** with **inner**

Step 3
Check if equal, unit price of outer query with maximum unit price of inner query

Max 60

Inner item table

itemid	itemname	unitprice	class
B1	Long Book	60	Stationary
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

Logic of correlated sub queries

Outer item table

itemid	itemname	unitprice	class
B1	Long Book	60	Stationary
B2	Short Book	35	**Stationary**
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

Step 1
For every record of outer query
Pick the class name
Stationary

Step 2
In the inner query
Find maximum unitprice of class name picked from outer query
i.e compare **class name** of **outer** with **inner**

Step 3
Check if equal, unit price of outer query with maximum unit price of inner query

Max 60

Inner item table

itemid	itemname	unitprice	class
B1	Long Book	60	Stationary
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

Logic of correlated sub queries

Outer item table

itemid	itemname	unitprice	class
B1	Long Book	60	Stationary
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	**Food**
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

Step 1
For every record of outer query
Pick the class name
Food

Step 2
In the inner query
Find maximum unitprice of class name picked from outer query
i.e compare **class name** of **outer** with **inner**

Step 3
Check if equal, unit price of outer query with maximum unit price of inner query

Max 150

Inner item table

itemid	itemname	unitprice	class
B1	Long Book	60	Stationary
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

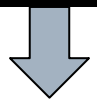
Non Correlated Sub Query

- Using IN Operator

Item

ITEMID	ITEMNAME	UNITPRICE	CLASS
B1	Long Book	60	Stationary
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

```
SQL> select * from item outer
     where outer.unitprice IN (select max(inner.unitprice) from item inner
     group by inner.class);
```



ITEMID	ITEMNAME	UNITPRICE	CLASS
B1	Long Book	60	Stationary
C2	Chocolate IceCream	150	Food

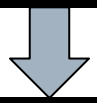
Observation

- **Question : Why the Error is getting generated when we run the following Query ?**

Item

ITEMID	ITEMNAME	UNITPRICE	CLASS
B1	Long Book	60	Stationary
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

```
SQL> select * from item outer
     where outer.unitprice = (select max(inner.unitprice) from item inner
     group by inner.class);
```



```
ERROR at line 2:
ORA-01427: single-row subquery returns more than one row
```

Observation

- **Question: Justify why we are getting the following output when we run the query as shown below**

Item

ITEMID	ITEMNAME	UNITPRICE	CLASS
B1	Long Book	60	Stationary
B2	Short Book	35	Stationary
P1	Natraja Pencil	10	Stationary
P2	Apsara Pencil	5	Stationary
C1	Vanila IceCream	75	Food
C2	Chocolate IceCream	150	Food
R1	Fried Rice	80	Food
R2	Curd Rice	50	Food

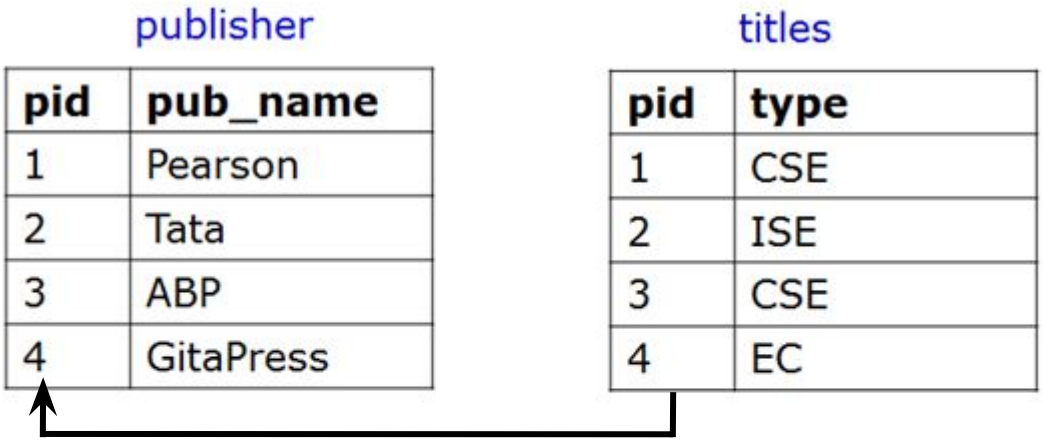
```
SQL> select * from item outer
     where outer.unitprice =(select max(inner.unitprice) from item inner
     group by outer.class);
```

Output

ITEMID	ITEMNAME	UNITPRICE	CLASS
C2	Chocolate IceCream	150	Food

Illustration of Subquery EXISTS Operator

Find the names of the publishers who have published CSE books:



pid	pub_name
1	Pearson
2	Tata
3	ABP
4	GitaPress

pid	type
1	CSE
2	ISE
3	CSE
4	EC

OUTPUT should be

pub_name
Pearson
ABP

Without using Subquery EXISTS Operator

Find the names of the publishers who have published CSE books:

- Non Correlated Sub Query: Using IN operator

publisher

pid	pub_name
1	Pearson
2	Tata
3	ABP
4	GitaPress

titles

pid	type
1	CSE
2	ISE
3	CSE
4	EC

```
select pub_name
from publisher
where pid in  (select pid
                from titles
                where type = 'CSE')
```

```
pub_name
-----------
Pearson
ABP
```

Without using Subquery EXISTS Operator

Find the names of the publishers who have published CSE books:

Using Cartesian product or Cross Join

publisher

pid	pub_name
1	Pearson
2	Tata
3	ABP
4	GitaPress

titles

pid	type
1	CSE
2	ISE
3	CSE
4	EC

```
select pub_name
from publisher p, titles t
where p.pid = t.pid and t.type='CSE';
```

```
pub_name
------------
Pearson
ABP
```

Without using Subquery EXISTS Operator

Find the names of the publishers who have published CSE books:

Using Correlated sub query

publisher

pid	pub_name
1	Pearson
2	Tata
3	ABP
4	GitaPress

titles

pid	type
1	CSE
2	ISE
3	CSE
4	EC

```
select pub_name
from publisher p
where p.pid = (select t.pid from titles t
                    where t.pid = p.pid and t.type='CSE';
```

```
pub_name
-----------
Pearson
ABP
```

Subquery: EXISTS Operator

- The EXISTS keyword is used to check whether a sub query produces any row(s) of result.
- If the query following the EXISTS return **at least one row**, then EXISTS returns **TRUE**
- If the query following the EXISTS returns no rows, then EXISTS returns FALSE

Subquery: EXISTS Operator

Find the names of the publishers who have published CSE books:

Using Exists Operator

publisher

pid	pub_name
1	Pearson
2	Tata
3	ABP
4	GitaPress

titles

pid	type
1	CSE
2	ISE
3	CSE
4	EC

```
select pub_name
from publisher p
where exists    (select  *  from titles t
                  where t.pid = p.pid
                  and type = 'CSE')
```

pub_name
Pearson
ABP

Subquery: EXISTS Operator

Why we will get the following output when we run the query as shown below ?

publisher

pid	pub_name
1	Pearson
2	Tata
3	ABP
4	GitaPress

titles

pid	type
1	CSE
2	ISE
3	CSE
4	EC

```
select pub_name
from publisher p
where exists    (select  *  from titles t
                  where type = 'CSE')
```

OUTPUT

```
pub_name
-----------
Pearson
Tata
ABP
GitaPress
```

Different ways of framing query for same output

publisher

pid	pub_name
1	Pearson
2	Tata
3	ABP
4	GitaPress

titles

pid	type
1	CSE
2	ISE
3	CSE
4	EC

Output

```
pub_name
------------
Pearson
ABP
```

S1

```
select pub_name
from publisher
where pid in  (select pid
                from titles
                where type = 'CSE')
```

S4

```
select pub_name
from publisher p
where exists    (select  *  from titles t
                  where t.pid = p.pid
                  and type = 'CSE')
```

```
select pub_name
from publisher p, titles t
where p.pid = t.pid and t.type='CSE';
```

S2

```
select pub_name
from publisher p
where p.pid = (select t.pid from titles t
                where t.pid = p.pid and t.type='CSE';
```

S3

Subquery: NOT EXISTS Operator

- The NOT EXISTS keyword is used to check whether a sub query produces any row(s) of result.
- If the query following the NOT EXISTS returns **no row**, then NOT EXISTS returns **TRUE**
- If the query following the NOT EXISTS returns any row(s), then NOT EXISTS returns FALSE

Subquery: NOT EXISTS Operator

- **not exists** is just like **exists** except that the **where** clause in which it is used is satisfied when no rows are returned by the subquery.
- Find the names of publishers who do *not* publish CSE books

publisher

pid	pub_name
1	Pearson
2	Tata
3	ABP
4	GitaPress

titles

pid	type
1	CSE
2	ISE
3	CSE
4	EC

OUTPUT should be

```
pub_name
------------
Tata
GitaPress
```

Subquery: NOT EXISTS Operator

- **not exists** is just like **exists** except that the **where** clause in which it is used is satisfied when no rows are returned by the subquery.
- Find the names of publishers who do *not* publish CSE books:

publisher

pid	pub_name
1	Pearson
2	Tata
3	ABP
4	GitaPress

titles

pid	type
1	CSE
2	ISE
3	CSE
4	EC

```
select pub_name
from publisher
where not exists    (select  *  from titles
                     where pid = publisher.pid
                     and type = 'CSE')
```

pub_name
Tata
GitaPress

Subqueries with EXISTS or NOT EXISTS

- The EXISTS operator tests for existence of rows in the results set of the subquery. If a subquery row value is found, EXISTS subquery is TRUE and in this case NOT EXISTS subquery is FALSE.
- **Syntax :**

SELECT column1 FROM table1 WHERE EXISTS (SELECT * FROM table2);

- Use the **exists** keyword with a subquery to test for the existence of some result from the subquery:
- {where | having} [not] exists (*subquery*)
- That is, the **where** clause of the outer query tests for the existence of the rows returned by the subquery. The subquery does not actually produce any data, but returns a value of TRUE or FALSE.

Subquery: UNIQUE Operator

- The UNIQUE keyword is used to check whether a sub query produces any row(s) of result.
- If the subquery following the UNIQUE returns **no two rows identical** then UNIQUE returns **TRUE**
- If the subquery following the UNIQUE returns atleast **any two rows identical**, then UNIQUE returns **FALSE**

Summary

- In **independent sub query** the inner query executes first and then the outer query executes utilizing the result obtained by inner query.
- In **correlated sub query** the inner query executed once for every row of the outer query.
- **EXISTS** returns TRUE if the inner query used returns any one record.
- **NOT EXISTS** returns TRUE if the inner query does not return anything.

SQL JOIN statement

- The SQL **Joins** clause is used to combine records from **two or more tables**.
- A JOIN is a means for combining fields from two tables by using values common to each.

Types of JOIN operator

- Cartesian product or Cross Join
- Inner Join
- Outer Join
  - Left-outer Join or Left Join
  - Right-outer Join or Right Join
- Full Join
- Self Join

CARTESIAN JOIN or CROSS JOIN

- The **CARTESIAN JOIN** or **CROSS JOIN** returns the Cartesian product of the sets of records from the two or more joined tables.
- Syntax of **CARTESIAN JOIN** or **CROSS JOIN** is as follows:

```
SELECT table1.column1, table2.column2...
FROM table1, table2 ;
```

CARTESIAN JOIN or CROSS JOIN

- The **CARTESIAN JOIN** or **CROSS JOIN** returns the Cartesian product of the sets of records from the two or more joined tables.
- Syntax of **CARTESIAN JOIN** or **CROSS JOIN** is as follows:

```
SELECT table1.column1, table2.column2...
FROM table1, table2 ;
```

table1

ID	M
1	a
2	b
4	c

table2

ID	N
2	p
3	q
5	r

Query

```
select * from table1,table2;
```

OUTPUT

```
ID M ID N
-- ---- --
 1 a  2 p
 1 a  3 q
 1 a  5 r
 2 b  2 p
 2 b  3 q
 2 b  5 r
 4 c  2 p
 4 c  3 q
 4 c  5 r
```

General Syntax of JOIN types: Inner, Left Outer, Right Outer and Full Join

- In order to perform a JOIN query, the required information we need are:

a) The name of the tables

b) Name of the columns of two or more tables, based on which a condition will perform.

- SYNTAX

```
SELECT ColumNames……..
FROM table1 join_type table2
ON join_condition;
```

General Syntax of JOIN operation

- In order to perform a JOIN query, the required information we need are:

a) The name of the tables

b) Name of the columns of two or more tables, based on which a condition will perform.

- SYNTAX

```
SELECT ColumNames.....
FROM table1 join_type table2
ON join_condition;
```

First table or **Left** table

Second table or **Right** table

Inner Join

- Inner Join or Equi Join returns rows when there is a match in both tables.
- The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.
- Syntax of **INNER JOIN**:

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

Inner Join

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

table1

ID	M
1	a
2	b
4	c

table2

ID	N
2	p
3	q
5	r

```
select *
from table1 inner join table2
on table1.id=table2.id;
```

ID	M	ID	N
2	b	2	p

Inner Join

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

table1

ID	M
1	a
2	b
4	c

table2

ID	N
2	p
3	q
5	r

```
select *
from table1 inner join table2
on table1.id=table2.id;
```

ID	M	ID	N
2	b	2	p

```
select *
from table1,table2
where table1.id=table2.id;
```

Inner Join: More than two tables

```
SELECT table1.column1, table2.column2,table3.colum3…….
FROM table1 INNER JOIN table2
ON table1.common_field = table2.common_field
INNER JOIN table3 ON table2.common_field = table3.common_field;
```

table1

ID	M
1	a
2	b
4	c

table2

ID	N
2	p
3	q
5	r

table3

ID	S
2	x
6	y
7	z

ID	M	ID	N	ID	S
2	b	2	p	2	x

```
SQL> select * from
     table1 inner join table2 on table1.id=table2.id
     inner join table3 on table2.id=table3.id;
```

Left Outer Join

- Left-Outer Join returns all rows from the left table, even if there are no matches in the right table.
- This means that if the ON clause matches 0 (zero) records in right table, the join will still return a row in the result, but with NULL in each column from right table.
- This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.
- Syntax of Left Outer Join or left join

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

Left Outer Join

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

table1

ID	M
1	a
2	b
4	c

table2

ID	N
2	p
3	q
5	r

```
SQL> select *
     from table1 left outer join table2
     on table1.id=table2.id;
```

ID	M	ID	N
2	b	2	P
1	a		
4	c		

Right Outer Join

- **RIGHT JOIN or RIGHT OUTER JOIN** returns all rows from the right table, even if there are no matches in the left table.
- This means that if the ON clause matches 0 (zero) records in left table, the join will still return a row in the result, but with NULL in each column from left table.
- This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.
- Syntax of Right Outer Join or Right join

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

Right Outer Join

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

table1

ID	M
1	a
2	b
4	c

table2

ID	N
2	p
3	q
5	r

```
SQL> select *
     from table1 right outer join table2
     on table1.id=table2.id;
```

ID	M	ID	N
2	b	2	P
		3	q
		5	r

Full Join

- **FULL JOIN** combines the results of both left and right outer joins.
- The joined table will contain all records from both tables, and fill in NULLs for missing matches on either side.
- Syntax of Full join

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

Full Join

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

table1

ID	M
1	a
2	b
4	c

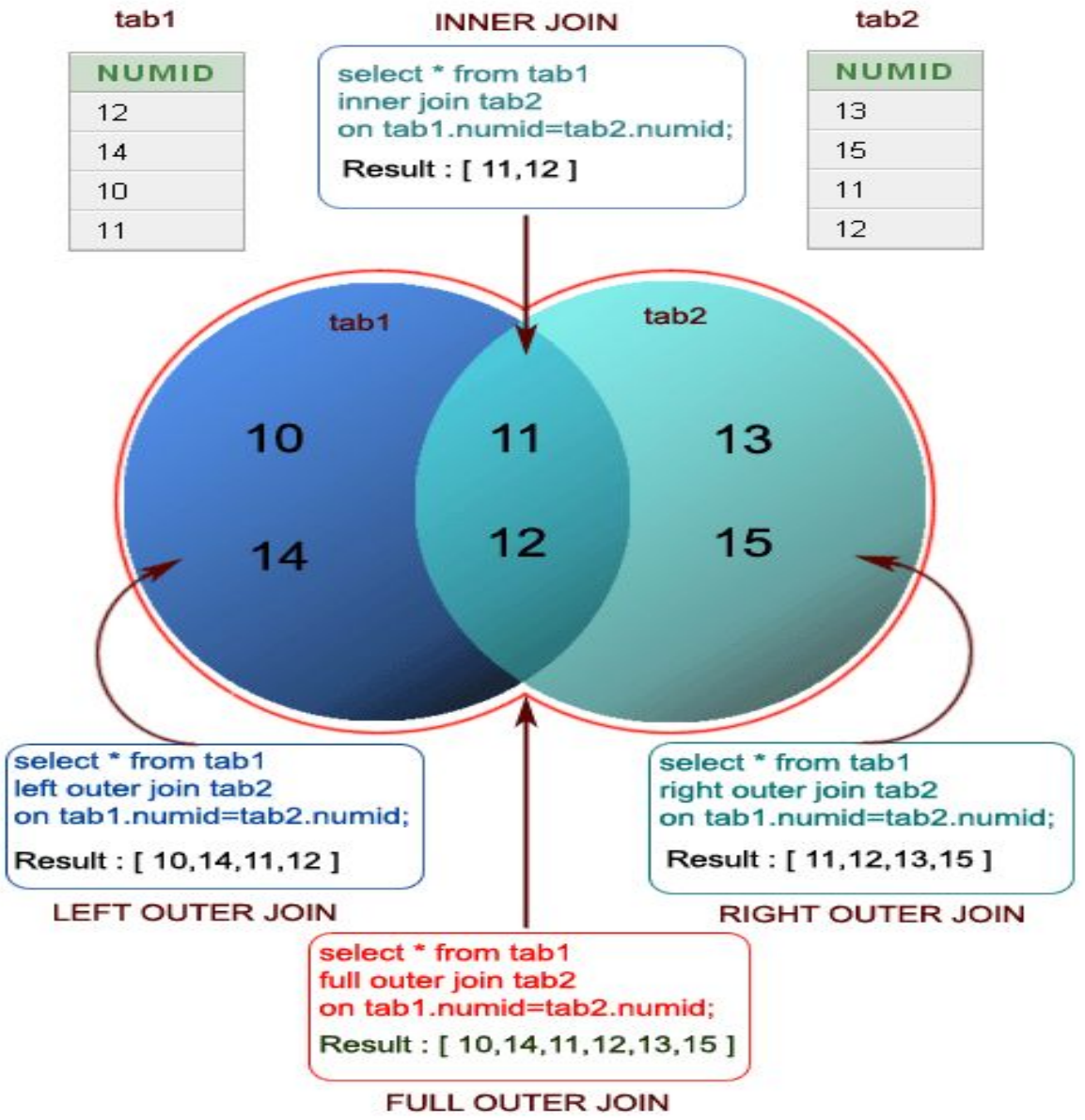
table2

ID	N
2	p
3	q
5	r

```
SQL> select *
     from table1 full join table2
     on table1.id = table2.id;
```

ID	M	ID	N
2	b	2	p
		3	q
		5	r
1	a		
4	c		

Types of Joins



Self Join

- **SELF JOIN** is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- Syntax:

```
SELECT a.column1, b.column2...
FROM table1 a , table1 b
WHERE table1.common_field = table1.common_field;
```

Self Join

Write SQL query
To list all employee names along with their manager's name

Employee table

EMP_ID	EMP_Name	Manager_ID
101	Avinash	103
102	Balaji	101
103	Chandan	
104	Dinesh	101

OUTPUT

EMP_Name	Manager_ID
Avinash	Chandan
Balaji	Avinash
Dinesh	Avinash

Logic of Self Join

Employee **E**

EMP_ID	EMP_Name	Manager_ID
101	Avinash	103
102	Balaji	101
103	Chandan	
104	Dinesh	101

Employee **M**

EMP_ID	EMP_Name	Manager_ID
101	Avinash	103
102	Balaji	101
103	Chandan	
104	Dinesh	101

Compare
E.Manager_ID= **M**.EMP_ID

If equal list
E.EMP_Name, **M**.EMP_Name

Logic of Self Join

Employee **E**

EMP_ID	EMP_Name	Manager_ID
101	Avinash	103
102	Balaji	101
103	Chandan	
104	Dinesh	101

Employee **M**

EMP_ID	EMP_Name	Manager_ID
101	Avinash	103
102	Balaji	101
103	Chandan	
104	Dinesh	101

Compare
E.Manager_ID= **M**.EMP_ID

If equal list
E.EMP_Name, **M**.EMP_Name

Logic of Self Join

Employee **E**

EMP_ID	EMP_Name	Manager_ID
101	Avinash	103
102	Balaji	101
103	Chandan	
104	Dinesh	101

Employee **M**

EMP_ID	EMP_Name	Manager_ID
101	Avinash	103
102	Balaji	101
103	Chandan	
104	Dinesh	101

Compare
E.Manager_ID= **M**.EMP_ID

If equal list
E.EMP_Name, **M**.EMP_Name

Output

Aninash, Chandan

Self Join

Write SQL query
To list all employee names along with their manager's name

Employee table

EMP_ID	EMP_Name	Manager_ID
101	Avinash	103
102	Balaji	101
103	Chandan	
104	Dinesh	101

```
SQL> select e.emp_name,m.emp_name
     from employee e, employee m
     where e.manager_id=m.emp_id;
```

EMP_Name	Manager_ID
Avinash	Chandan
Balaji	Avinash
Dinesh	Avinash

Practice SQL Query

- List details of customer who have purchased some items

Customer

Cust_ID	Cust_Name

Customer Purchase

Cust_ID	ItemID	QtyPurchased	TotalPrice

Practice SQL Query

- List details of customer who have purchased some items

```
SQL> select * from customer;

   CUST_ID CUST_NAME
---------- ----------------
       101 Avinash
       102 Balaji
       103 Chandan
```

```
SQL> select * from customer_purchase;

   CUST_ID IT QTYPURCHASE TOTALPRICE
---------- -- ---------- ----------
       101 I1          2        100
       103 I2          1         50
```

OUTPUT should be

```
CUST_ID CUST_NAME            CUST_ID IT QTYPURCHASE TOTALPRICE
------- -------------------- ------- -- ---------- ----------
    101 Avinash                  101 I1          2        100
    103 Chandan                  103 I2          1         50
```

Practice SQL Query

- List details of customer who have purchased some items

```
SQL> select * from customer;

   CUST_ID CUST_NAME
---------- ----------
       101 Avinash
       102 Balaji
       103 Chandan
```

```
SQL> select * from customer_purchase;

   CUST_ID IT QTYPURCHASE TOTALPRICE
---------- -- ---------- ----------
       101 I1          2        100
       103 I2          1         50
```

```
SQL> select *
     from customer inner join customer_purchase
     on customer.cust_id=customer_purchase.cust_id;
```

```
CUST_ID CUST_NAME            CUST_ID IT QTYPURCHASE TOTALPRICE
------- -------------------- ---------- -- ---------- ----------
    101 Avinash                   101 I1          2        100
    103 Chandan                   103 I2          1         50
```

Practice SQL Query

- List details of customer who have purchased some items

```
SQL> select * from customer;

   CUST_ID CUST_NAME
---------- ----------
       101 Avinash
       102 Balaji
       103 Chandan
```

```
SQL> select * from customer_purchase;

   CUST_ID IT QTYPURCHASE TOTALPRICE
---------- -- ---------- ----------
       101 I1          2        100
       103 I2          1         50
```

```
SQL> select *
     from customer,customer_purchase
     where customer.cust_id=customer_purchase.cust_id;
```

```
CUST_ID CUST_NAME            CUST_ID IT QTYPURCHASE TOTALPRICE
------- -------------------- ---------- -- ---------- ----------
    101 Avinash                   101 I1          2        100
    103 Chandan                   103 I2          1         50
```

Practice SQL Query

- List all customer details and loan details if they have availed loans or yet to avail loan

Customer_Account_Details

Cust_ID	Cust_Name

Customer_Loans

Loan_ID	Cust_ID	Amount_in_Rupees

Practice SQL Query

- List all customer details and loan details if they have availed loans or yet to avail loan

```
SQL> select * from customer;

   CUST_ID CUST_NAME
---------- --------------------
       101 Avinash
       102 Balaji
       103 Chandan
```

```
SQL> select * from loan;

LO    CUST_ID      AMOUNT
-- ---------- ----------
L1         101       1000
L2         103       2000
```

```
CUST_ID CUST_NAME            LO    CUST_ID      AMOUNT
------- -------------------- -- ---------- ----------
    101 Avinash              L1         101       1000
    103 Chandan              L2         103       2000
    102 Balaji
```

Practice SQL Query

- List all customer details and loan details if they have availed loans or yet to avail loan

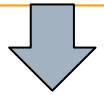
```
SQL> select * from customer;

   CUST_ID CUST_NAME
---------- ----------
       101 Avinash
       102 Balaji
       103 Chandan
```

```
SQL> select * from loan;

LO    CUST_ID     AMOUNT
-- ---------- ----------
L1         101       1000
L2         103       2000
```

```
SQL> select *
     from customer left outer join loan
     on customer.cust_id=loan.cust_id;
```



```
CUST_ID CUST_NAME            LO    CUST_ID     AMOUNT
------- -------------------- -- ---------- ----------
    101 Avinash              L1         101       1000
    103 Chandan              L2         103       2000
    102 Balaji
```

Practice SQL Query

- Display details of Supplier who has been ordered to supply more than one item

Supplier

Suplier_ID	Suplier_Name

Item Order

Item_ID	Item_name	Supplier_ID

Practice SQL Query

- Display details of Supplier who has been ordered to supply more than one item

```
SQL> select * from supplier;

SUPPLIER_ID SUPPLIER_NAME
----------- ---------------
        101 Avinash
        102 Avinash
```

```
SQL> select * from item_order;

IT ITEM_NAME            SUPPLIER_ID
-- -------------------- -----------
I1 Mobile                       101
I2 Laptop                       102
I3 Laptop                       101
```

OUTPUT

```
SUPPLIER_ID SUPPLIER_NAME
----------- ---------------
        101 Avinash
```

Practice SQL Query

- Display details of Supplier who has been ordered to supply more than one item

```
SQL> select * from supplier;

SUPPLIER_ID SUPPLIER_NAME
----------- ---------------
        101 Avinash
        102 Avinash
```

```
SQL> select * from item_order;

IT ITEM_NAME            SUPPLIER_ID
-- -------------------- -----------
I1 Mobile                       101
I2 Laptop                       102
I3 Laptop                       101
```

```
SQL> select s.supplier_id,s.supplier_name
     from supplier s,item_order i
     where s.supplier_id=i.supplier_id
     group by s.supplier_id,s.supplier_name
     having count(*) > 1;

SUPPLIER_ID SUPPLIER_NAME
----------- --------------------
        101 Avinash
```

Practice SQL Query

- Display details of Supplier who has been ordered to supply more than one item

Supplier

Suplier_ID	Suplier_Name

Item Order

Item_ID	Item_name	Supplier_ID

```
Select s.Supplier_ID,s.Supplier_Name
From Supplier s,ItemOrder i
Where s.Supplier_ID=i.SupplierID
Group By s.Supplier_ID,s.Supplier_Name
Having Count(*) > 1
```

NULL Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The predicate **is null** can be used to check for null values.
  - Example: Find all loan number which appear in the *loan* relation with null values for *amount.*

    select *loan_number*
    from *loan*
    where *amount* **is null**
- The result of any arithmetic expression involving *null* is *null*
  - Example: 5 + *null* returns null
- However, aggregate functions simply ignore nulls

Null value interpretation

- **Unknown value:** A particular person has date of birth but it is not known, so it is represented by NULL in the database.
- **Unavailable or withheld value:** A person has a home phone but does not want it to be listed, so it is withheld and represented as NULL in the database.
- **Not applicable attribute:** An attribute LastCollegeDegree would be NULL for a person who has no college degree because it does not apply to that person.

Null Values and Three Valued Logic

- **TRUE, FALSE and UNKNOWN**
- A comparison between known values gives you a result of TRUE or FALSE. This is Boolean logic.
- When you do a comparison with a NULL, you cannot get a Boolean (i.e. TRUE or FALSE) result. This is where the logical value UNKNOWN was invented.
- Logical Connectives in Three-valued logic

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

Topics Covered in Todays Class

Unit 1:
Views in SQL
Specifying Constraints as Assertions and Triggers

Views in SQL

- A view is a kind of "**Virtual Table**"
- A view is customized representation of data from **one or more tables**. The tables that the view is referencing are know as **base tables**.
- A view is considered as a **stored query** or a virtual table.
- Why we need Views ?

  Views, which are kind of virtual tables, allow users to do the following:
  - **Structure data** in a way that users or classes of users find natural or intuitive.
  - **Restrict access** to the data such that a user can see and (sometimes) modify exactly what they need and no more.
  - **Summarize data** from various tables which can be used to generate reports.

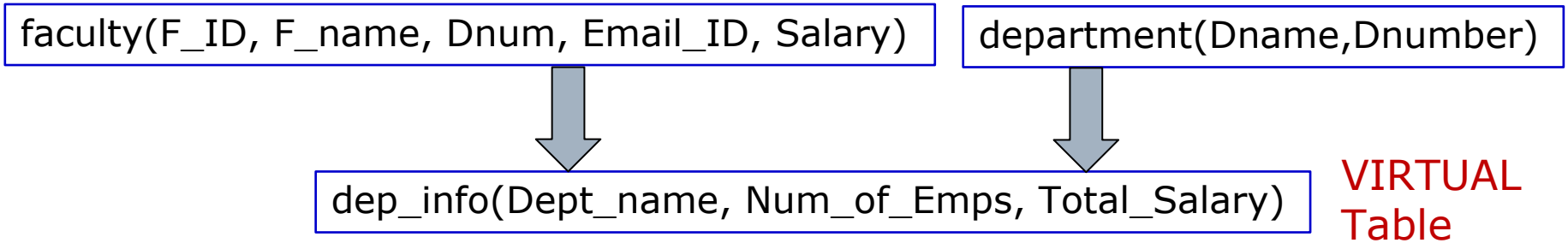
Creating Views

- Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables, or another view.
- To create a view, a user must have the appropriate system privilege according to the specific implementation.
- The basic CREATE VIEW syntax is as follows:

```
CREATE VIEW view_name
AS SELECT column1, column2.....
    FROM table_name
    WHERE [condition];
```

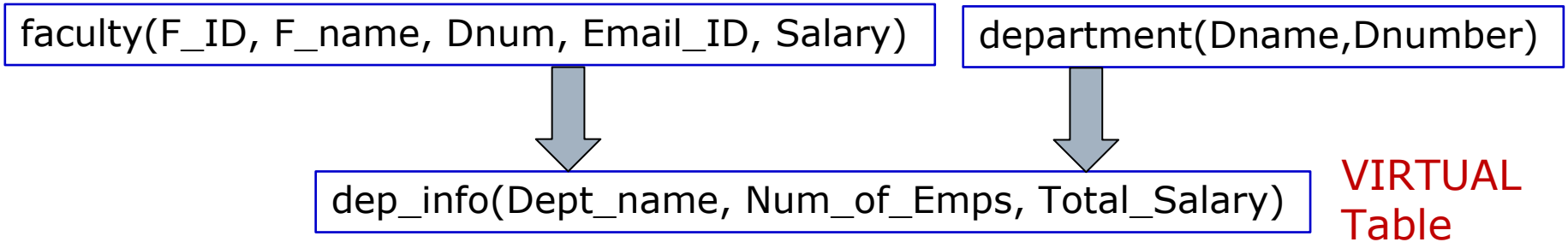
Creating Views

- Example



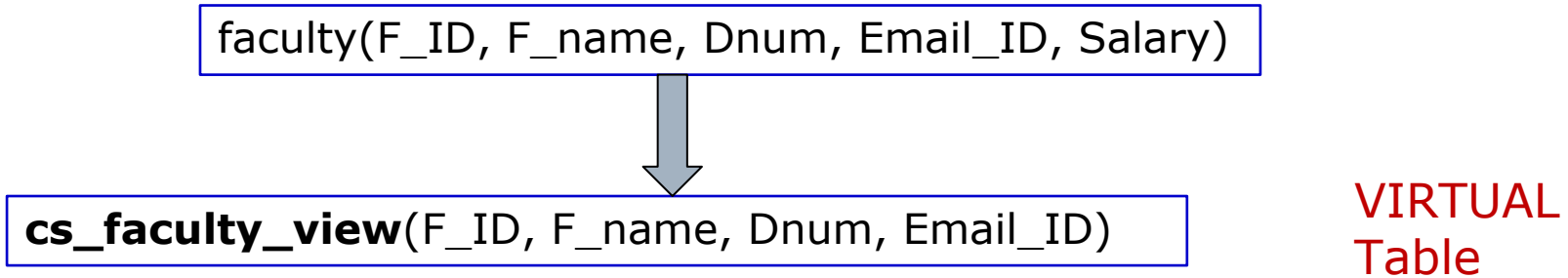
Creating Views

- Example



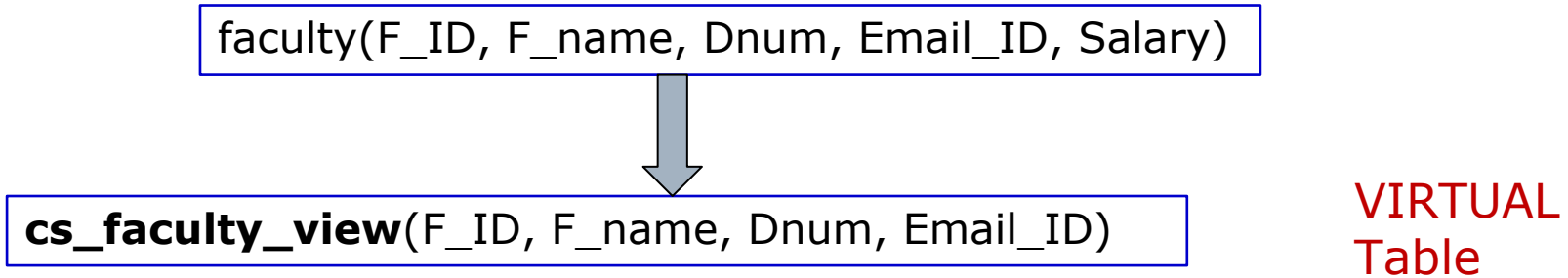

```
CREATE VIEW dep_info(Dept_name,Num_of_Emps,Total_Salary)
AS Select d.Dname,count(*),sum(salary)
    From department d, faculty f
    Where d.Dnumber=f.Dnum
    Group By d.Dname;
```

Updating View



VIRTUAL Table

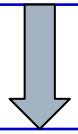
Updating View




```
CREATE VIEW cs_faculty_view(F_ID, F_name, Dnum, Email_ID)
AS Select *
    From faculty
    Where Dnum=10;
```

Updating View

faculty(F_ID, F_name, Dnum, Email_ID, Salary)



cs_faculty_view(F_ID, F_name, Dnum, Email_ID)

VIRTUAL Table

```
CREATE VIEW cs_faculty_view(F_ID, F_name, Dnum, Email_ID)
AS Select *
    From faculty
    Where Dnum=10;
```

```
Update cs_faculty_view
Set Email_ID='balaji@gmail.com'
Where Emp_name='Balaji';
```

Updating a View

A view can be updated under certain conditions:

- The SELECT clause may not contain the keyword DISTINCT.
- The SELECT clause may not contain aggregate functions.
- The SELECT clause may not contain set functions.
- The SELECT clause may not contain set operators.
- The SELECT clause may not contain an ORDER BY clause.
- The FROM clause may not contain multiple tables.
- The WHERE clause may not contain subqueries.
- The query may not contain GROUP BY or HAVING.
- Calculated columns may not be updated.
- All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

So if a view satisfies all the above-mentioned rules then you can update a view.

Note: Similarly

- Rows of data can be inserted into a view.
- Rows of data can be deleted from a view.

Dropping Views:

- Syntax
- DROP VIEW view_name;

- Example: drop view cs_faculty_view;

Advantages and Disadvantages of views

Advantages of views

- **Security:** Each user can be given permission to access the database only through a small set of views that contain the specific data the user is authorized to see, thus restricting the user's access to stored data
- **Query Simplicity:** A view can draw data from several different tables and present it as a single table, turning multi-table queries into single-table queries against the view.
- **Structural simplicity:** Views can give a user a "personalized" view of the database structure, presenting the database as a set of virtual tables that make sense for that user.
- **Consistency:** A view can present a consistent, unchanged image of the structure of the database, even if the underlying source tables are split, restructured, or renamed.
- **Data Integrity:** If data is accessed and entered through a view, the DBMS can automatically check the data to ensure that it meets the specified integrity constraints.
- **Logical data independence:** View can make the application and database tables to a certain extent independent. If there is no view, the application must be based on a table. With the view, the program can be established in view of above, to view the program with a database table to be separated.

Disadvantages of views

- **Performance:** Views create the appearance of a table, but the DBMS must still translate queries against the view into queries against the underlying source tables. If the view is defined by a complex, multi-table query then simple queries on the views may take considerable time.
- **Update restrictions:** When a user tries to update rows of a view, the DBMS must translate the request into an update on rows of the underlying base tables. This is possible for simple views, but more complex views are often restricted to read-only.

Topics Covered in Todays Class

Unit 1:
Specifying Constraints as Assertions and Triggers

Specifying Constraints in SQL

- Specifying Attributes constraints and Attribute values
- Specifying Key and Referential Integrity constraints
- Specifying constraints on Tuples using CHECK

Specifying Attributes constraints and Attribute values

```
create table student.student_info(
USN char(10),
Name char(30) NOT NULL,
DepName char(3) NOT NULL DEFAULT 'CSE',
Marks int NOT NULL CHECK (Marks > 0 AND Marks < 101)
);
```

Constraints on Attributes

- NOT NULL
- DEAFULT
- CHECK

Specifying Key and Referential Integrity constraints

- Referential Integrity constraint or Foreign Key Constraint

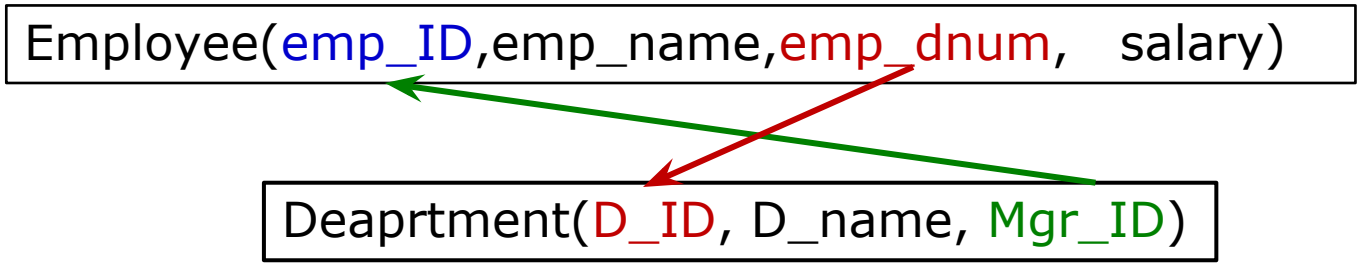
```
create  table Department(
D_ID int,
D_Name char(3),
PRIMARY KEY (D_ID)
);
```

```
create table Student(
USN char(10),
S_Name char(20),
Dep_Num int,
PRIMARY KEY (USN),
FOREIGN KEY(Dep_Num) REFERENCES Department(D_ID)
);
```

General Assertions

- Constraints on entire relation or entire database
- Syntax: CREATE ASSERTION <name> CHECK(<condition>);

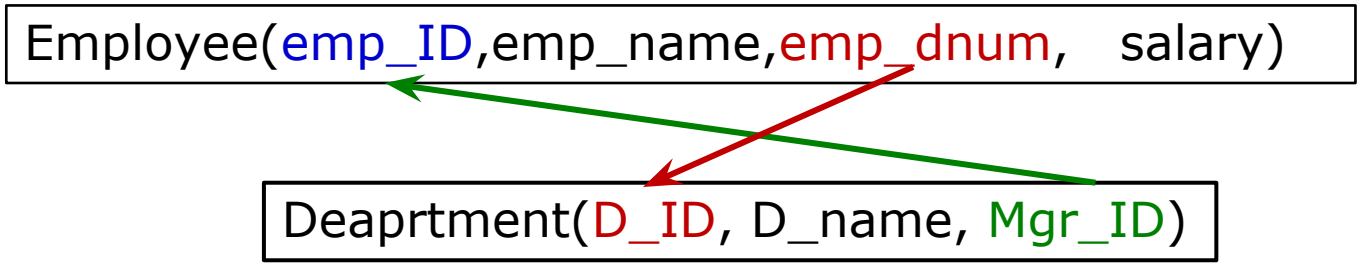
Example: Salary of the employee must not be greater than the salary of the manager of the department that the employee works for



General Assertions

- Constraints on entire relation or entire database
- Syntax: CREATE ASSERTION <name> CHECK(<condition>);

Example: Salary of the employee must not be greater than the salary of the manager of the department that the employee works for

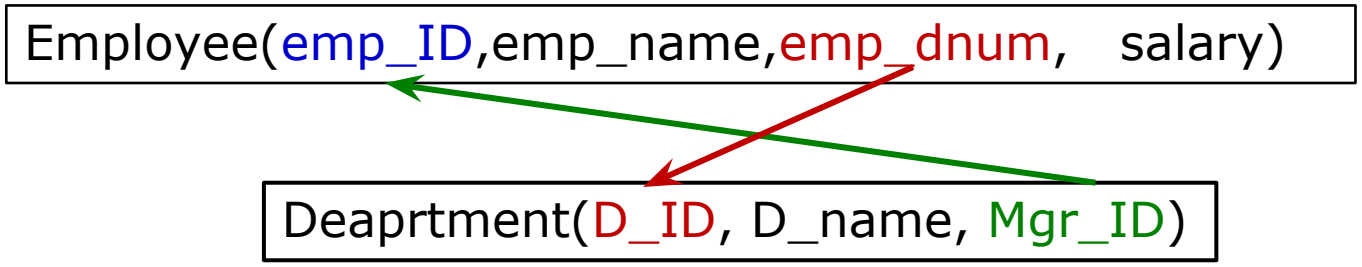



```
CREATE ASSERTION Salary_constraint
CHECK (NOT EXISTS (Select e.emp_name
From Employee e, Employee m, Department d
Where e.salary > m.salary
        and e.emp_dnum=d.D_ID
        and d.Mgr_ID=m.emp_ID));
```

General Assertions

- Constraints on entire relation or entire database
- Syntax: CREATE ASSERTION <name> CHECK(<condition>);

Example: Salary of the employee must not be greater than the salary of the manager of the department that the employee works for



ASSERTION
Salary Constraint
Not Violated

```
CREATE ASSERTION Salary_constraint
CHECK (NOT EXISTS (Select e.emp_name
From Employee e, Employee m, Department d
Where e.salary > m.salary
        and e.emp_dnum=d.D_ID
        and d.Mgr_ID=m.emp_ID));
```

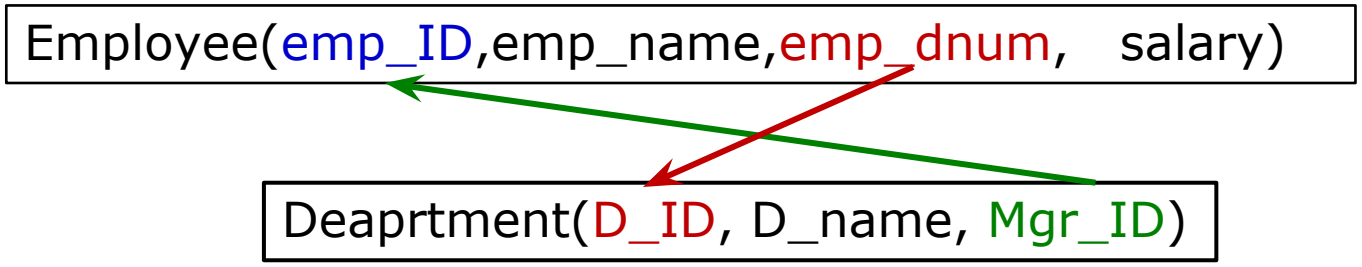
```
Employee(ID,name,dnum,sal)
------------------------------
        E1,Avinash,10,1000
        E2,Balaji,10,2000
```

```
Department(D_ID,D_name,Mgr_ID)
------------------------------------
            D1,      10,   E2
```

General Assertions

- Constraints on entire relation or entire database
- Syntax: CREATE ASSERTION <name> CHECK(<condition>);

Example: Salary of the employee must not be greater than the salary of the manager of the department that the employee works for



ASSERTION Salary Constraint **Violated**

```
CREATE ASSERTION Salary_constraint
CHECK (NOT EXISTS (Select e.emp_name
From Employee e, Employee m, Department d
Where e.salary > m.salary
        and e.emp_dnum=d.D_ID
        and d.Mgr_ID=m.emp_ID));
```

Employee(ID,name,dnum,sal)

E1,Avinash,10,**2000**
E2,Balaji,10,**1000**

Department(D_ID,D_name,Mgr_ID)

D1, 10, E2

Assertion: To Do

Write Assertion to check,

The minimum price charged for products made by Coca-Cola Company should be Rs.20/-

Tables are:

- cool_drink(name, manf)
- sells(cooldrink_name, price)

General Syntax of assertions

```
CREATE ASSERTION <assertion_name>
CHECK (NOT EXISTS <SQL Query>);
```

Assertion: To Do

Write Assertion to check,

The minimum price charged for products made by Coca-Cola Company should be Rs.20/-

Tables are:

- cool_drink(name, manf)
- sells(cooldrink_name, price)

```
CREATE ASSERTION NoCheapCooolDrink
CHECK( NOT EXISTS( SELECT * FROM cool_drink, sells
    WHERE sells.cooldrink_name = cool_drink.name
            AND cool_drink.manf = 'Coca-Cola'
            AND sells.price < 20
            ));
```

Triggers

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events:

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Benefits of Triggers, Triggers can be written for the following purposes:

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating Triggers: Syntax

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

Creating Triggers: Syntax

Where,

- CREATE [OR REPLACE] TRIGGER trigger_name: Creates or replaces an existing trigger with the *trigger_name*.
- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

Triggers: Example

- To start with, we will be using the CUSTOMERS table we had created and used in the previous chapters:

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

The following program creates a **row level** trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values:

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN ( ID > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff := :NEW.salary  - :OLD.salary;
   dbms_output.put_line('Old salary: ' || :OLD.salary);
   dbms_output.put_line('New salary: ' || :NEW.salary);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
```

Thank You for Your Time and Attention !

Cascading referential integrity

Options when setting up Cascading referential integrity constraint:

1. No Action: This is the default behaviour. No Action specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, an error is raised and the DELETE or UPDATE is rolled back.

2. Cascade: Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are also deleted or updated.

3. Set NULL: Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are set to NULL.

4. Set Default: Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are set to default values.

Subquery: EXISTS Operator

- **List EMPNO of the employees whose salary is greater than the salary of any employees in the department 10**

```
SELECT e1.empno, e1.sal
FROM   emp e1
WHERE  e1.sal > ANY (SELECT e2.sal
                     FROM   emp e2
                     WHERE  e2.deptno = 10);
```

EMPNO	SAL
7839	5000
7902	3000
7788	3000
7566	2975
7698	2850
7782	2450
7499	1600
7844	1500

```
-- Transformed to equivalent statement without ANY.

SELECT e1.empno, e1.sal
FROM   emp e1
WHERE  EXISTS (SELECT e2.sal
               FROM emp e2
               WHERE e2.deptno = 10
               AND   e1.sal > e2.sal);
```

EMPNO	SAL
7839	5000
7902	3000
7788	3000
7566	2975
7698	2850
7782	2450
7499	1600
7844	1500

Subquery: NOT EXISTS Operator

- **List EMPNO of the employees whose salary is greater than the salary of all employees in the department 20**

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-1980 00:00:00	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-1981 00:00:00	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-1981 00:00:00	1250	500	30
7566	JONES	MANAGER	7839	02-APR-1981 00:00:00	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-1981 00:00:00	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-1981 00:00:00	2850		30
7782	CLARK	MANAGER	7839	09-JUN-1981 00:00:00	2450		10
7788	SCOTT	ANALYST	7566	19-APR-1987 00:00:00	3000		20
7839	KING	PRESIDENT		17-NOV-1981 00:00:00	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-1981 00:00:00	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-1987 00:00:00	1100		20
7900	JAMES	CLERK	7698	03-DEC-1981 00:00:00	950		30
7902	FORD	ANALYST	7566	03-DEC-1981 00:00:00	3000		20
7934	MILLER	CLERK	7782	23-JAN-1982 00:00:00	1300		10

```
-- Transformed to equivalent statement using ANY.

SELECT e1.empno, e1.sal
FROM   emp e1
WHERE  NOT (e1.sal <= ANY (SELECT e2.sal
                           FROM emp e2
                           WHERE e2.deptno = 20));
```

EMPNO	SAL
7839	5000

```
-- Transformed to equivalent statement without ANY.

SELECT e1.empno, e1.sal
FROM   emp e1
WHERE  NOT EXISTS (SELECT e2.sal
                   FROM emp e2
                   WHERE e2.deptno = 20
                   AND   e1.sal <= e2.sal);
```

EMPNO	SAL
7839	5000