United International **University** (UIU)

Course Code: CSE 3522

Course Title: Database Management Systems Laboratory

Faculty: Humaira Anzum Neha

Trimester: Fall 2024

TASK - 01

Create following tables (schema of University enterprise) with primary key and foreign keyconstraints.Bold indicates primary keys. Foreign keys are to be understood (See data).

classroom(**building, room_number**, capacity)
department(**dept_name**,building, budget)
course(**course_id**, title, dept_name,credits)
instructor(**ID**,name,dept_name,salary)
section(**course_id,sec_id,semester**,year,building, room_number,time_slot_id)
teaches(**ID, course_id,sec_id,semester,year**)
student(**ID**,name,dept_name,tot_cred)
takes(**ID,course_id, sec_id,semester, year**,grade)
advisor(**s_ID**,i_ID)
timeslot(**time_slot_id,day,start_time**,end_time)
prereq(**course_id,prereq_id**)

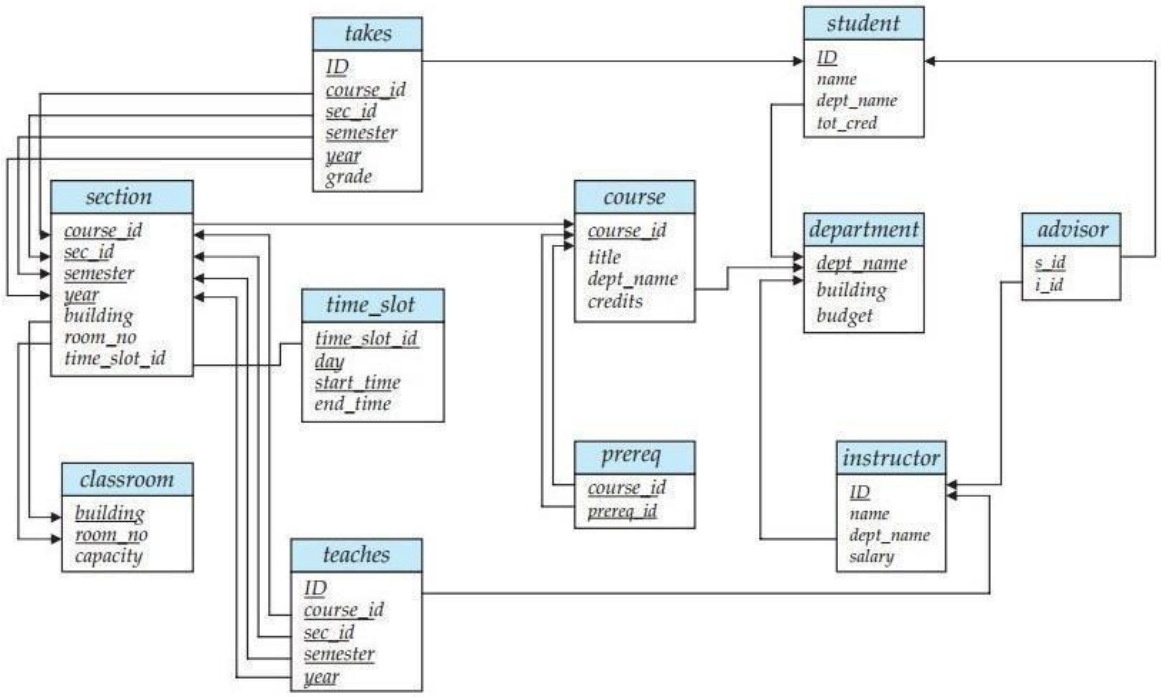

Figure 2.8 Schema diagram for the university database.

TASK - 02

Then Insert following data shown in the tables:

building	*room_number*	*capacity*
Packard	101	500
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

Figure A.3 The *classroom* relation.

dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

Figure 2.5 The *department* relation.

course_id	*title*	*dept_name*	*credits*
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

Figure 2.2 The *course* relation.

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Figure 2.1 The *instructor* relation.

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

Figure A.7 The *section* relation.

ID	*name*	*dept_name*	*tot_cred*
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

Figure 4.1 The *student* relation.

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

Figure 2.7 The *teaches* relation.

ID	*course_id*	*sec_id*	*semester*	*year*	*grade*
00128	CS-101	1	Fall	2009	A
00128	CS-347	1	Fall	2009	A-
12345	CS-101	1	Fall	2009	C
12345	CS-190	2	Spring	2009	A
12345	CS-315	1	Spring	2010	A
12345	CS-347	1	Fall	2009	A
19991	HIS-351	1	Spring	2010	B
23121	FIN-201	1	Spring	2010	C+
44553	PHY-101	1	Fall	2009	B-
45678	CS-101	1	Fall	2009	F
45678	CS-101	1	Spring	2010	B+
45678	CS-319	1	Spring	2010	B
54321	CS-101	1	Fall	2009	A-
54321	CS-190	2	Spring	2009	B+
55739	MU-199	1	Spring	2010	A-
76543	CS-101	1	Fall	2009	A
76543	CS-319	2	Spring	2010	A
76653	EE-181	1	Spring	2009	C
98765	CS-101	1	Fall	2009	C-
98765	CS-315	1	Spring	2010	B
98988	BIO-101	1	Summer	2009	A
98988	BIO-301	1	Summer	2010	*null*

Figure 4.2 The *takes* relation.

s_id	i_id
00128	45565
12345	10101
23121	76543
44553	22222
45678	22222
76543	45565
76653	98345
98765	98345
98988	76766

Figure A.11 The *advisor* relation.

course_id	prereq_id
BIO-301	BIO-101
BIO-399	BIO-101
CS-190	CS-101
CS-315	CS-101
CS-319	CS-101
CS-347	CS-101
EE-181	PHY-101

Figure 2.3 The *prereq* relation.

time_slot_id	day	start_hr	start_min	end_hr	end_min
A	M	8	0	8	50
A	W	8	0	8	50
A	F	8	0	8	50
B	M	9	0	9	50
B	W	9	0	9	50
B	F	9	0	9	50
C	M	11	0	11	50
C	W	11	0	11	50
C	F	11	0	11	50
D	M	13	0	13	50
D	W	13	0	13	50
D	F	13	0	13	50
E	T	10	30	11	45
E	R	10	30	11	45
F	T	14	30	15	45
F	R	14	30	15	45
G	M	16	0	16	50
G	W	16	0	16	50
G	F	16	0	16	50
H	W	10	0	12	30

Figure A.14 The *time_slot* relation

TASK - 03

Show and run the above scripts and then perform the following queries:

- Try to use subqueries wherever possible.
- Try to write and execute all the possible queries that would yield the same result.

a. Find the names of all departments with the instructor, and remove duplicates.
b. Find the names of all instructors in the History department.
c. Find the instructor_ID and department_name of all instructors associated with a department with a budget of greater than $95,000.
d. Find all instructors in the Computer Science department with salaries more than$80,000.
e. List the names of instructors along with the titles of courses that they teach.
f. Find the names of all instructors who have a higher salary than some instructors in 'Computer Science'.

g. Find the names of all instructors whose salary is greater than at least one instructor in theBiology department.
h. Find the names of all departments whose building name includes the substring Watson'.
i. List in alphabetic order the names of all instructors.
j. Find the names of instructors with salary amounts between $90,000 and $100,000.
k. Find the instructor names and the courses they taught for all instructors in theBiology department who have taught some course.
l. Find the set of all courses taught either in Fall 2009 or in Spring 2010 Semester, or both.
m. Find the set of all courses taught in the Fall 2009 as well as in Spring 2010 Semester.
n. Find all courses taught in the Fall 2009 semester but not in the Spring 2010 Semester.
o. Find all instructors whose salary is null.
p. Find courses offered in Fall 2009 and in Spring 2010 Semester.
q. Find courses offered in Fall 2009 but not in Spring 2010 Semester.
r. Find the total number of (distinct) students who have taken course sections taught by the instructor with ID 10101.
s. Find the names of instructors with salaries greater than that of some (at least one) instructor in the Biology department.
t. Find the names of all instructors whose salary is greater than the salary of all instructors in the Biology department.
u. Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester.
v. Find all students who have taken all courses offered in the Biology department.
w. Find all courses that were offered at most once in 2009

Compiled Reference

SELECT … FROM … WHERE …

The SELECT statement allows you to retrieve records from one or more tables in your database. The syntax for the SELECT statement is:

```
SELECT columns
FROM tables
WHERE predicates;
```

In general, the meaning of a SQLquery can be understood as follows:

1. Generate a Cartesian product of the relations (if there) listed in the from clause
2. Apply the predicates specified in the where clause on the result of Step 1.
3. For each tuple in the result of Step 2, output the attributes (or results of expressions) specified in the select clause.

The more-or-less full syntax of the SELECT statement is:

```
SELECT    column-list
FROM      table-join-expression
[WHERE    expression]
[GROUP BY column-list]
[HAVING  aggregate-Boolean-expression]
[ORDER BY column-list];
```

The meaning of each clause is as follows:

- The `SELECT` clause defines which columns will appear in the result set.

- The `FROM` clause specifies which tables will provide the data for the result set, and also how they should be joined together.
- The `WHERE` clause determines which rows will appear in the result set. The absence of the `WHERE` clause means that all rows will appear.
- The `GROUP BY` clause creates a group on the basis of related columns.
- The `HAVING` clause is only necessary when the `GROUP BY` clause is in place, and is used to filter out rows from the grouped result set.
- The `ORDER BY` clause sorts the result set.

YOU MUST KNOW:

- SQL statements are not case sensitive. E.g. Name ≡ NAME ≡ name
- SQL statements can be on one or more lines.
- Keywords cannot be abbreviated or split across lines.
- Clauses are usually placed on separate lines.
- Indents are used to enhance readability.

MORE TERMS:

- KEYWORD:
- CLAUSE:
- STATEMENT:

Combining the "AND" and "OR" Conditions in WHERE clause

The AND and OR conditions can be combined in a single SQL statement. When combining these conditions, it is important to use brackets so that the database knows what order to evaluate each condition. See the examples:

```
SELECT *FROM suppliers
WHERE (city = 'New York' and name = 'IBM')or (city = 'Newark');
```

This would return all suppliers that reside in New York whose name is IBM and all suppliers that reside in Newark. The brackets determine what order the AND and OR conditions are evaluated in.Test the following SQL:

```
SELECT *FROM suppliers
WHERE city = 'New York' and name = 'IBM' or city = 'Newark';
```

Eliminate Duplicates with DISTINCT

If a result set produced by a SELECT statement contains duplicate rows, these can be eliminated by using the DISTINCT keyword in the query immediately following the SELECT keyword. For example,

```
SELECT DISTINCT model
FROM    aircraft
```

Note that DISTINCT applies to entire rows in the result set and not individual columns. Indiscriminate use of DISTINCT can occasionally produce misleading results, particularly in those cases when it is perfectly allowable for duplicate rows to exist in a result set.

Specify range using BETWEEN...AND

The following is an SQL statement that uses the BETWEEN function:

```
SELECT *FROM suppliers
 WHERE supplier_id between 5000 AND 5010;
```

This would return all rows where the supplier_id is between 5000 and 5010, inclusive. It is equivalent to the following SQL statement:

```
SELECT *FROM suppliers
WHERE supplier_id >= 5000 AND supplier_id <= 5010;
```

You can use BETWEEN...AND operators to do the exact same thing in a more descriptive fashion, and using less typing to boot:

```
SELECT *FROM airport
WHERE   city BETWEEN 'London' AND 'Rome';
```

Both of the above queries produce exactly the same result set. The BETWEEN...AND operators specify an inclusive range (i.e. the endpoints of the range are included). To only select value outside of the range, the NOT keyword is used:

```
SELECT *FROM airport
WHERE   city NOT BETWEEN 'London' AND 'Rome';
```

The BETWEEN...AND operators work with all column datatypes, including number, date, and string types.

Specify Lists in a WHERE Clause using IN

When filtering the results of a query so that just a list of values is returned in the result set, you can either specify each item in the list in turn using OR:

```
SELECT  *FROM  airport
WHERE  city = 'Berlin'
OR      city = 'Madrid'
OR      city = 'Rome';
```

Or, you can use the IN operator to accomplish the same thing but with much less typing:

```
SELECT *FROM    airport
WHERE   city IN ('Berlin', 'Madrid', 'Rome');
```

Note the parentheses around the list of values. We can also use the NOT operator to filter by values that are not in a list:

```
SELECT *FROM    airport
WHERE   city NOT IN ('Berlin', 'Madrid', 'Rome');
```

Sort Result Sets with ORDER BY

The ORDER BY clause allows you to sort the records in your result set. The ORDER BY clause can only be used in SELECT statements. By default, each column is sorted in ascending order. To sort in descending order, type the keyword DESC after the column name.For instance,

```
SELECT supplier_city, supplier_state
FROM suppliers
WHERE supplier_name = 'IBM'
ORDER BY supplier_city DESC, supplier_state ASC;
```

This would return all records sorted by the supplier_city field in descending order, with a secondary sort by supplier_state in ascending order.

Specify Wildcard String Matches in a WHERE Clause using LIKE

Strings can be matched using wildcards in SQL just as one would use wildcards to match files in MS-DOS or UNIX. The wildcards are different, however. In SQL, the percent sign (%) matches "zero, one, or more characters" and the underscore (_) matches "exactly one character".

The LIKE condition allows you to use wildcards in the where clause of an SQL statement. This allows you to perform pattern matching. The LIKE condition can be used in any valid SQL statement. The patterns that you can choose from are:

percent (%)allows you to match any string of any length (including zero length)
underscore (_)allows you to match on a single character

```
SELECT * FROM suppliers
WHERE supplier_name LIKE 'Hew%';
```

Remember: Patterns are case sensitive.
Pattern matching examples:

'Intro%' matches any string beginning with "Intro".
'%Comp%' matches any string containing "Comp" as a substring.
'_ _ _' matches any string of exactly three characters.
'_ _ _%' matches any string of at least three characters.

Renaming Result Set Columns using Column Aliases

Just as tables can have aliases in queries, so too can columns. While table aliases are a technique to reduce typing, column aliases are strictly decorative. A column alias simply determines the name of the column in the result set--nothing else is affected in the query. To specify a column alias, type the alias immediately following the column name (or calculated expression) with the keyword AS between the name and alias (PostgreSQL insists on the AS keyword, unlike other RDBMSs):

```
SELECT  r.rental_id, apd.city AS departure_city,
FROM    rental r, airport apd
ORDER BY departure_city, arrival_city;
```

Column aliases are most useful when two or more column names would otherwise be displayed in the result set with the same name. Column aliases can also be used in an ORDER BY clause, as in the previous example, instead of the normal column name (particularly handy when you want to sort by a calculated expression) or in the ORDER BY clause used by the UNION of two or more queries. In the latter case, the column alias must be defined in the first SELECT statement in the query.

Combine Multiple Result Sets with UNION

The UNION query allows you to combine the result sets of 2 or more "select" queries. It removes duplicate rows between the various "select" statements.
Each SQL statement within the UNION query must have the same number of fields in the result sets with similar data types.

```
select field1, field2, . field_n
from tables
UNION / UNION ALL
select field1, field2, . field_n
from tables;
```

The UNION ALL query allows you to combine the result sets of 2 or more "select" queries. It returns all rows (even if the row exists in more than one of the "select" statements).
Each SQL statement within the UNION ALL query must have the same number of fields in the result sets with similar data types.

SUBQUER Y

A subquery is a query within a query. In Oracle, you can create subqueries within your SQL statements. These subqueries can reside in the WHERE clause, the FROM clause, or the SELECT clause.

WHERE clause

Most often, the subquery will be found in the WHERE clause. These subqueries are also called nested subqueries.For example:

```
select * from all_tables tabs
where tabs.table_name in (select cols.table_name
                from all_tab_columns cols
                where cols.column_name = 'SUPPLIER_ID');
```

FROM clause

A subquery can also be found in the FROM clause. These are called inline views.For example:

```
select suppliers.name, subquery1.total_amt
from suppliers,
          (select supplier_id, Sum(orders.amount) as total_amt
          from orders
          group by supplier_id) subquery1,
where subquery1.supplier_id = suppliers.supplier_id;
```

In this example, we've created a subquery in the FROM clause as follows:

```
(select supplier_id, Sum(orders.amount) as total_amt
from orders
group by supplier_id) subquery1
```

This subquery has been aliased with the name subquery1. This will be the name used to reference this subquery or any of its fields.

SELECT clause

A subquery can also be found in the SELECT clause.For example:

```
select tbls.owner, tbls.table_name,
        (select count(column_name) as total_columns
        from all_tab_columns cols
        where cols.owner = tbls.owner
        and cols.table_name = tbls.table_name) subquery2
from all_tables tbls;
```

In this example, we've created a subquery in the SELECT clause as follows:

```
(select count(column_name) as total_columns
from all_tab_columns cols
where cols.owner = tbls.owner
and cols.table_name = tbls.table_name) subquery2
```

The subquery has been aliased with the name subquery2. This will be the name used to reference this subquery or any of its fields.

The trick to placing a subquery in the select clause is that the subquery must return a single value. This is why an aggregate function such as SUM, COUNT, MIN, or MAX is commonly used in the subquery.

EXISTS / NOT EXISTS

The EXISTS condition is considered "to be met" if the subquery returns at least one row.

```
SELECT columns
FROM tables
WHERE EXISTS ( subquery );
```

Let's take a look at a simple example. The following is an SQL statement that uses the EXISTS condition:

```
SELECT *
FROM suppliers
WHERE EXISTS
  (select *
    from orders
    where suppliers.supplier_id = orders.supplier_id);
```

This select statement will return all records from the suppliers table where there is at least one record in the orders table with the same supplier_id.

The EXISTS condition can also be combined with the NOT operator. For example,

```
SELECT *
FROM suppliers
WHERE not exists (select * from orders Where suppliers.supplier_id =
orders.supplier_id);
```

This will return all records from the suppliers table where there are no records in the orders table for the given supplier_id.

DDL, DML

Data Definition Language (DDL)

DDL uses just five statements: CREATE, ALTER, DROP, GRANT, and REVOKE.
Data Definition Language (DDL) is all about creating, modifying and deleting relations.

```
Create table, drop table, ALTER TABLE,
```

Data Manipulation Language

DML uses just four statements: INSERT, UPDATE, DELETE, and SELECT.

Data Manipulation Language (DML)is all about adding, modifying, and removing data/tuples.

The four DML statements implement what is known as **CRUD** functionality: Create, Read, Update, and Delete. The actual DML statements use a couple of different names:
INSERT for Create and SELECT for Read, but the other two--UPDATE and DELETE--are the same.