

```
CREATE TABLE CONSTRUCT

• An SQL relation is defined using the create table command: create table r (A_1 D_1, A_2 D_2, ..., A_n D_n, (integrity-constraint_1完整性约束), ..., (integrity-constraint_k))

• r is the name of the relation

• each A_i is an attribute name in the schema of relation r e<sub>CLXD</sub>

• D_i is the data type of values in the domain of attribute A_i
```

```
CREATE TABLE CONSTRUCT

• the create table command

• Example:

create table instructor (

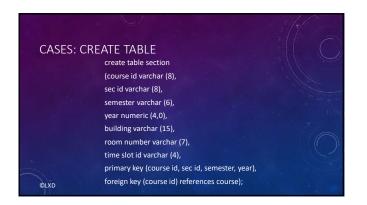
ID char(5),

name varchar(20),

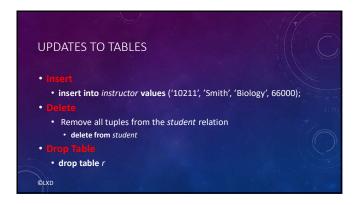
dept_name varchar(20),

salary numeric(8,2));
```



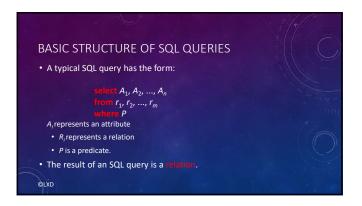


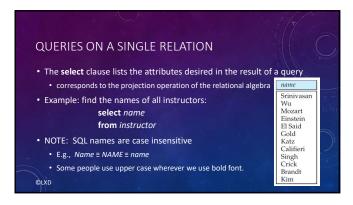


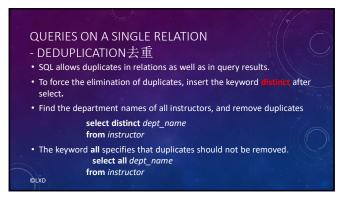


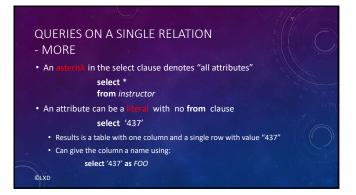


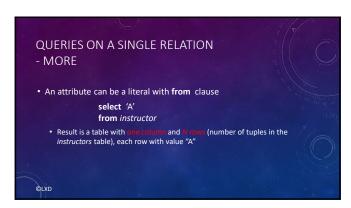












QUERIES ON A SINGLE RELATION

- MORE

• The select clause can contain arithmetic expressions involving the operation, +, -, \*, and /, and operating on constants or attributes of tuples.

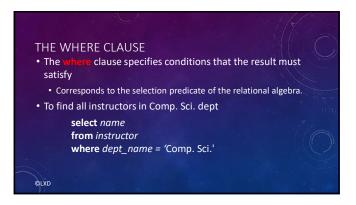
• The query:

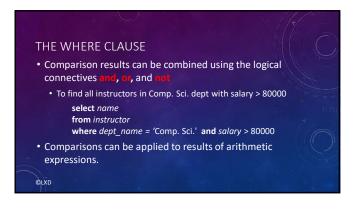
select ID, name, salary/12
from instructor

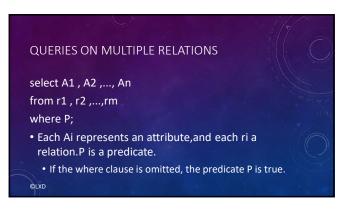
would return a relation that is the same as the instructor relation, except that the value of the attribute salary is divided by 12.

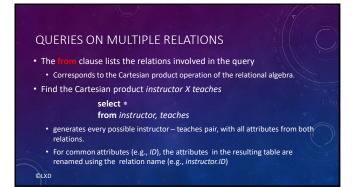
• Can rename "salary/12" using the as clause:

select ID, name, salary/12 as monthly\_salary



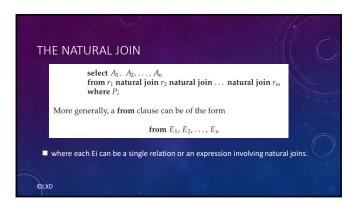










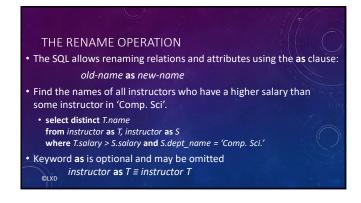


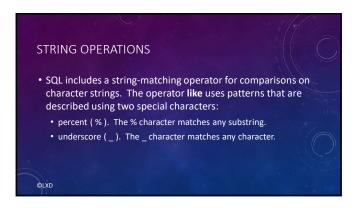


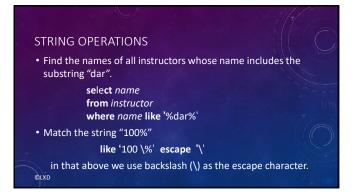






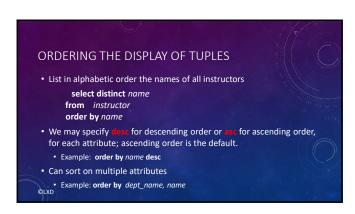


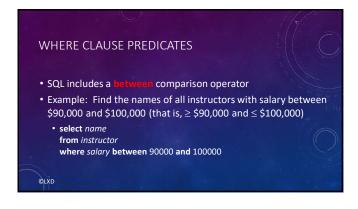


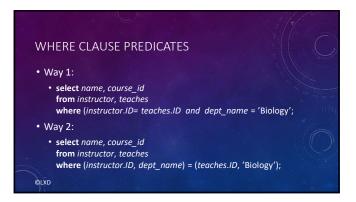


## STRING OPERATIONS Patterns are case sensitive. Pattern matching examples: 'Intro%' matches any string beginning with "Intro". '%Comp%' matches any string containing "Comp" as a substring. '\_\_\_' matches any string of exactly three characters. '\_\_\_ %' matches any string of at least three characters.

## STRING OPERATIONS • SQL supports a variety of string operations such as • concatenation (using "||") • converting from upper to lower case (and vice versa) • finding string length, extracting substrings, etc.

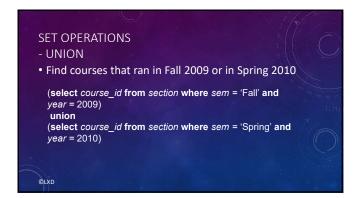






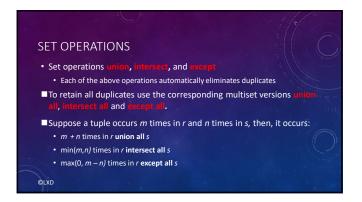




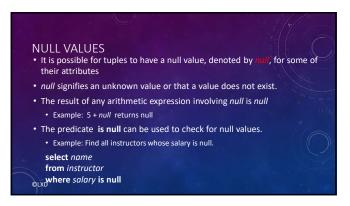


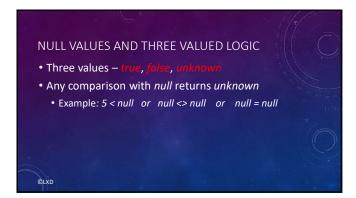


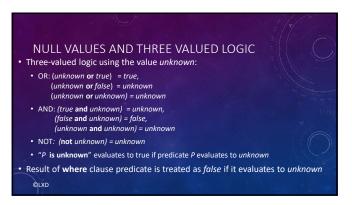


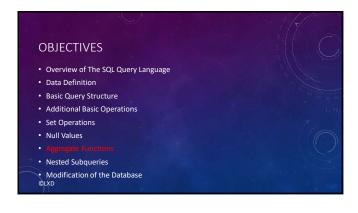




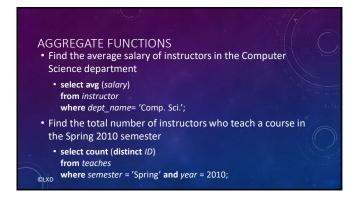


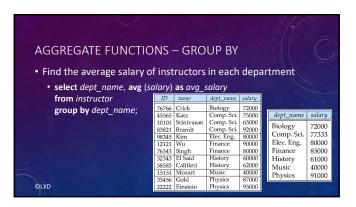


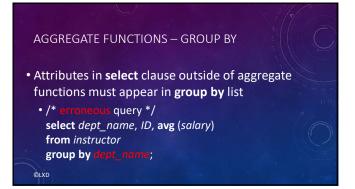


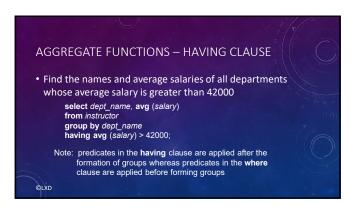




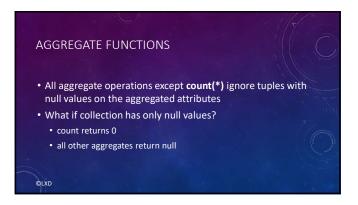








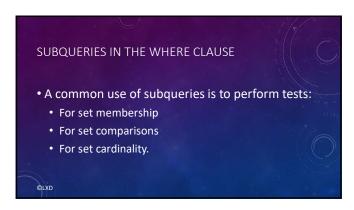




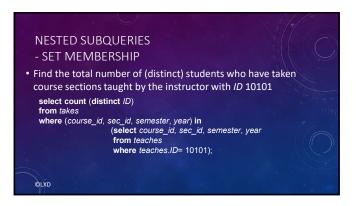


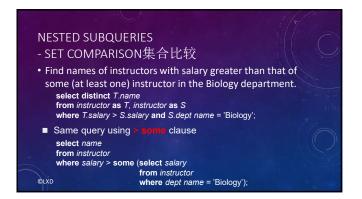


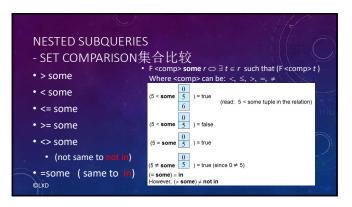












NESTED SUBQUERIES		
- SET COMPARISON集作	合比较	
	$F < comp > all r \Leftrightarrow \forall t \in r (F < comp > comp > r \in r (F < comp > r (F < comp > r \in r (F < comp > r (F < com$	t)
• > all	0	
• < all	(5 < all 5 ) = false	
• <=all	(5 < all 6 10 ) = true	
• >=all	(5 = all 5 ) = false	
• =all (not same to in)	4	
• <> all (same to not in)	(5 ≠ all 6 ) = true (since 5 ≠ 4 and 5 ≠ 6)	
an (same to not m)	(≠ all) = not in	
©LXD	However, (= all) ≠ in	



```
TEST FOR EMPTY RELATIONS

• exists, not exists

• Yet another way of specifying the query "Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester"

select course_id
from section as S
where semester = 'Fall' and year = 2009 and
exist (select *
from section as T
where semester = 'Spring' and year= 2010 and S.course_id = T.course_id);

• Correlation name = variable S in the outer query

• Virelated subquery = the inner query

• Virelated subquery = the inner query
```

```
TEST FOR EMPTY RELATIONS

• Find all students who have taken all courses offered in the Biology department.

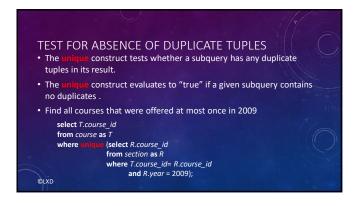
select distinct S.ID, S.name from student as S
where ind exists ((select course_id from course
where dept_name = Biology')
except
(select T.course_id from takes as T
where S.ID = T.ID));

• First nested query lists all courses offered in Biology
• Second nested query lists all courses a particular student took

■ Note that X - Y = Ø ⇔ X ⊆ Y

■ Note: Cannot write this query using = all and its variants

©LXD
```



```
SUBQUERIES IN THE FROM CLAUSE

• SQL allows a subquery expression to be used in the from clause

• Find the average instructors' salaries of those departments where the average salary is greater than $42,000."

select dept_name, avg_salary
name (select dept_name, avg (salary) as avg_salary
from instructor
group by dept_name)
where avg_salary > 42000;

• Note that we do not need to use the having clause
```

## SUBQUERIES IN THE FROM CLAUSE • Find the average instructors' salaries of those departments where the average salary is greater than \$42,000." • Another way to write above query select dept\_name, avg\_salary from (select dept\_name, avg (salary) from instructor group by dept\_name) as dept\_avg (dept\_name, avg\_salary) where avg\_salary > 42000;

```
WITH CLAUSE

• The with clause provides a way of defining a temporary relation whose definition is available only to the query in which the with clause occurs

• Introduced in SQL:1999

• Find all departments with the maximum budget

with max budget (value) as

(select max(budget)

from department)

select department.name

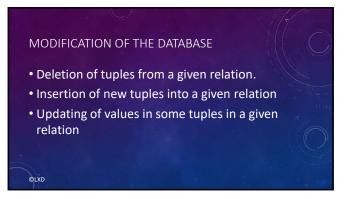
from department, max_budget

where department.budget = max_budget.value;
```

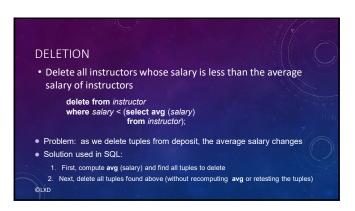
## COMPLEX QUERIES USING WITH CLAUSE • Find all departments where the total salary is greater than the average of the total salary at all departments with dept\_total (dept\_name, value) as (select dept\_name, sum(salary) from instructor group by dept\_name), dept\_total\_avg(value) from dept\_total) select dept\_name from dept\_total, dept\_total\_avg where dept\_total, value > dept\_total\_avg.value;



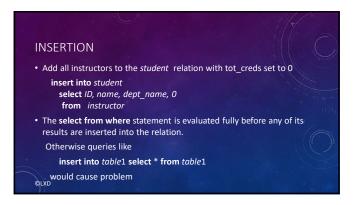








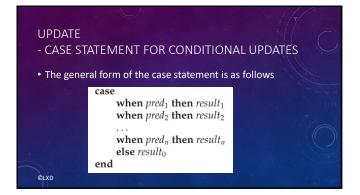






```
UPDATE
- CASE STATEMENT FOR CONDITIONAL UPDATES

• Same query as before but with case statement
update instructor
set salary = case
when salary <= 100000 then salary * 1.05
else salary * 1.03
end
```



UPDATES WITH SCALAR SUBQUERIES
 Recompute and update tot\_creds value for all students
 update student S
 set tot\_cred = (select sum(credits)
 from takes, course
 where takes.course\_id = course.course\_id and
 S.ID= takes.ID and takes.grade <> 'F' and
 takes.grade is not null);
 Sets tot\_creds to null for students who have not taken any course

