

# **Unit 3 –Part 2: SQL –Basic Query**

Database System Concepts

Abraham Silberschatz, Henry F. Korth, S. Sudarshan

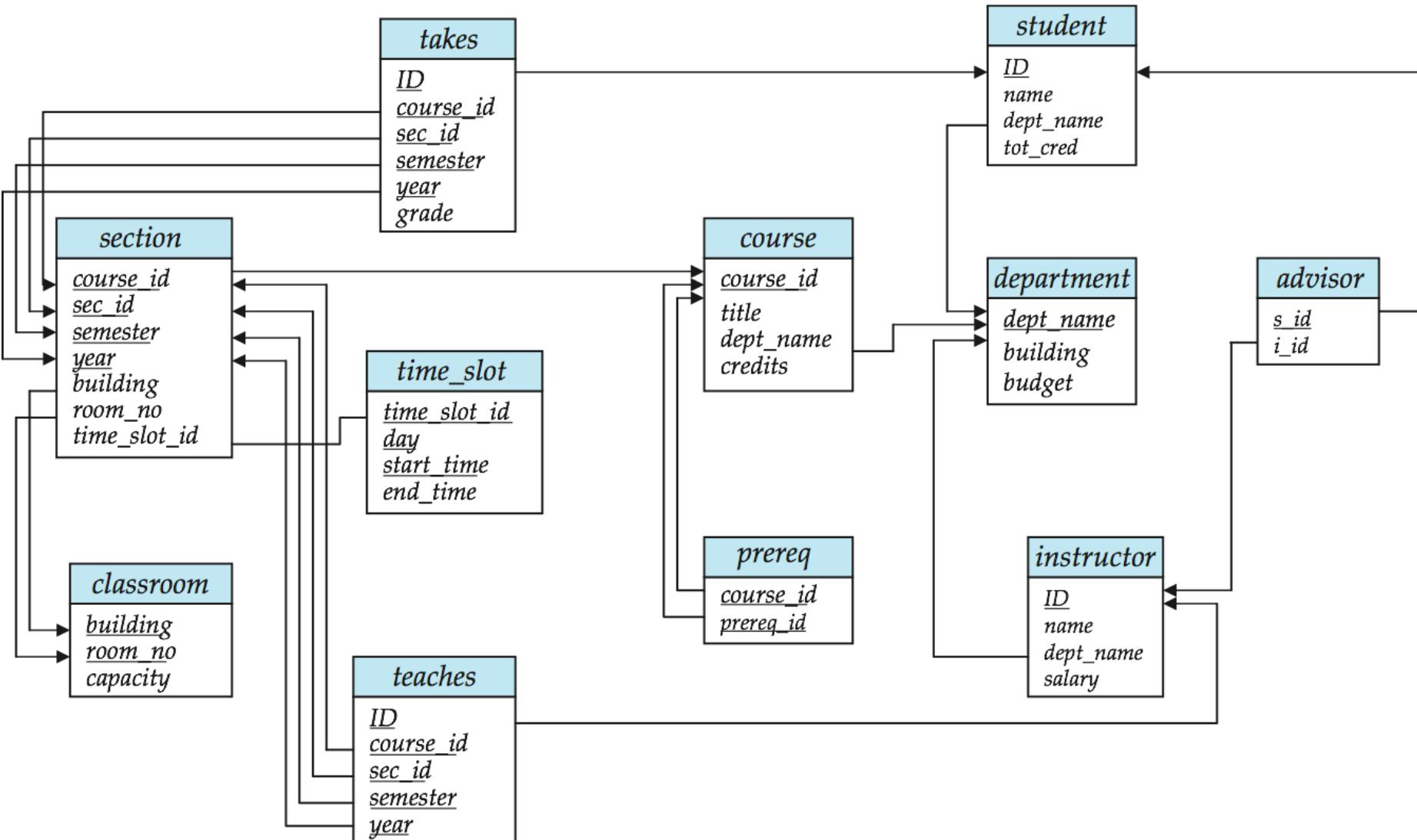
&

**Oracle database SQL Reference**

# Basic SQL Query

- Basic Query Structure
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database

# Schema



**Schema referred in query examples.**

# Basic Query Structure

- The SQL **data-manipulation language (DML)** provides the ability to query information, and insert, delete and update tuples
- A typical SQL query has the form:

```
select  $A_1, A_2, \dots, A_n$ 
from  $r_1, r_2, \dots, r_m$ 
where  $P$ 
```

- $A_i$  represents an **attribute**
- $R_i$  represents a **relation**
- $P$  is a **predicate -condition.**
- The **result** of an SQL query is a **relation.**

# The select Clause

- The **select** clause **list the attributes desired** in the result of a query
  - corresponds to the projection operation of the relational algebra
- **Example:** find the names of all instructors:

```
select name  
from instructor
```

- NOTE: SQL names are **case insensitive** (i.e., you may use upper- or lower-case letters.)

# The select Clause (Cont.)

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after select.
- Find the names of all departments with instructor, and remove duplicates

```
select distinct dept_name  
from instructor;
```

- The keyword **all** specifies that duplicates not be removed.

```
select all dept_name  
from instructor;
```

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

# The select Clause (Cont.)

- An **asterisk** in the select clause denotes “all attributes”

```
select *
from instructor;
```

- 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.

# Simple Queries

## Row selection (WHERE clause)

**WHERE clause consists of five basic search conditions:**

- **Comparison:** Compare the value of one expression to the value of another expression (= , < , > , <= , >= , <>).
- **Range:** Test whether the value of an expression falls within a specified range of values (BETWEEN/ NOT BETWEEN).
- **Set membership:** Test whether the value of an expression equals one of a set of values (IN/ NOT IN).
- **Pattern match:** Test whether a string matches a specified pattern (LIKE/ NOT LIKE).
- **NULL:** Test whether a column has null value (IS NULL/ IS NOT NULL).

# Choosing Rows with the WHERE Clause

- The **where** clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra.
- Used to return rows that match a condition, such as having a column value that exactly matches a string, a number greater or less than a value, or a string that is a prefix of another.
- The simplest WHERE clause is one that exactly matches a value.
- Example: Display the location of Research department.

```
select location from dept where dname='Research';
```

```
+-----+
| location |
+-----+
| MNG    |
```

# Choosing Rows with the WHERE Clause

- To find number and name of all employees with salary > 80000
- select empno, ename,sal from emp where sal>80000;

```
+-----+-----+
| empno | ename | sal    |
+-----+-----+
| 109 | Shan | 99999.99 |
| 110 | Sona | 99999.99 |
+-----+-----+
```

- For numbers, the frequently used operators are equals (=), greater than (>), less than (<), less than or equal (<=), greater than or equal (>=), and not equal (<> or !=).

# The where Clause(Cont'd)

- The **where** clause specifies conditions that the result must satisfy
- To find all instructors in Comp. Sci. dept with salary > 80000

```
select name  
from instructor  
where dept_name = 'Comp. Sci.' and salary > 80000
```
- Comparison results can be combined using **the logical connectives and, or, and not.**
- **Comparisons** can be **applied to results of arithmetic expressions.**
- Find the instructors whose salary exceed 100000 after 10% increment

```
select name  
from instructor  
where Salary*1.1 >100000;
```

# Simple Queries –between/not between

- BETWEEN checks if a value is within a range.
- NOT BETWEEN checks if a value is outside a range.
- Example-Consider  
Emp(Empno,Ename,Job,Mgrid,Date\_birth,Sal,Comm,deptno,d  
ate\_join)
- List all employee with a salary between 30000 and 40000
  - select empno,ename,sal from emp where sal between 30000 and 40000;
- This would be expressed as-
  - select empno,ename,sal from emp where sal>=30000 and sal<=40000;

# Simple Queries-in/Not in

- IN tests whether a data value matches one of a list values.
- NOT IN checks for data values that do not lie in a specific list of values.
- ***List all Managers or Assistants.***

```
select empno,ename,job from emp where job  
in('MGR','A.MGR');
```

- It can also be expressed as:

```
select empno,ename,job from emp where job ='MGR'  
OR job='A.MGR';
```

# Simple Queries-Like/Not Like

- **SQL has special pattern matching symbol:**
- % represents any sequence of zero or more character(wildcard)
- \_represents any single character
- **Example:**
- Address LIKE 'H%'means that the first character must be *H*,but the rest can be anything.
- Address LIKE 'H\_\_\_\_' means that there must be exactly four characters in the string,the first of which must be *H*.
- Address LIKE '%e' means any sequence of characters,of length at least 1,with the last character an e.
- Address LIKE '%Glasgow%' means a sequence of characters of any length containing *Glasgow*.
- Address NOT LIKE 'H%' means the first character cannot be *H*.<sup>17</sup>

# Simple Queries-Like/Not Like

- Example – Display the employee with the name ending with ‘raj’  
select ename from emp where ename like '%raj';
  
- Example – Display the employee with the name four-letter word beginning with ‘S’  
select ename from emp where ename like 'S\_\_\_\_';

# Simple Queries-is NULL/is not NULL

- NULL represents missing or unknown value
- NULL does not represent a zero or a string of blank spaces
- A NULL value cannot be tested with = or <> to another string
- We have to test for NULL explicitly
- Example: To display the employee who does not have a manager

```
select empno,ename from emp where mgrid is NULL;
```

# Simple Queries-Order By clause

- Allows the retrieved records to be ordered in ascending(ASC) or descending order(DESC) on any column or combination of columns

```
SELECT { * | [column_expression] [, ...] }  
       FROM table_name  
       [ORDER BY column_list [ASC | DESC] ]
```

Display the details of employee in the descending order of their salary

```
select empno,ename,sal from emp order by sal desc;
```

# Simple Queries-Order By clause

- List all employees, ordered by department and within each department, ordered alphabetically by name

```
select ename from emp order by deptno, ename;
```

# The from Clause

## Cartesian Product

- The **from** clause lists the relations involved in the query

- Corresponds to the **Cartesian product** operation.

- Find the Cartesian product *instructor X teaches*

```
select *  
from instructor, teaches
```

- generates **every possible instructor – teaches pair**, with all attributes from both relations
- Cartesian product **not very useful directly**, but **useful combined with where-clause** condition.

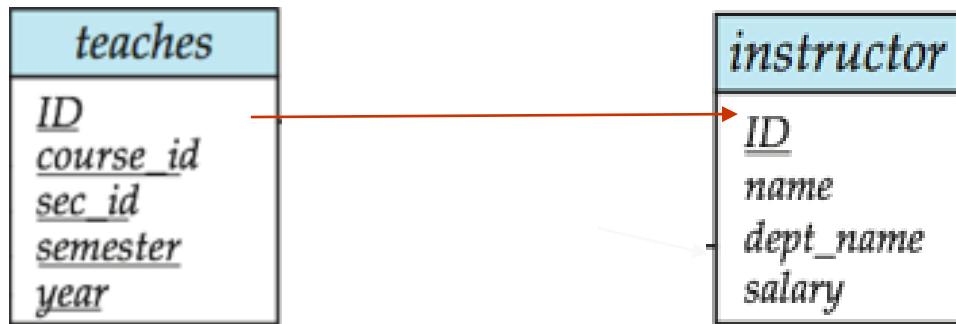
# Cartesian Product: *instructor X teaches*

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
62156	Gulli	Finance	87000

<i>ID</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>
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

# Join

For all instructors who have taught some course, find their names and the course ID of the courses they taught.



```
select name, course_id  
from instructor, teaches  
where instructor.ID = teaches.ID
```

# Join

Find the course ID, semester, year and title of each course offered by the Comp. Sci. department.



```
select section.course_id, semester, year, title  
from section, course  
where section.course_id = course.course_id and  
dept_name = 'Comp. Sci.'
```

# Natural Join

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column
- **select \***  
**from instructor natural join teaches;**

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010

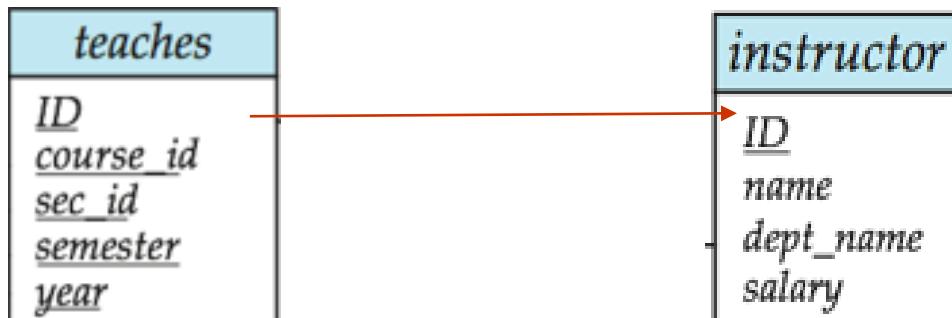
# Natural Join Example

- List the names of instructors along with the course ID of the courses that they taught.

- **select** *name, course\_id*  
**from** *instructor, teaches*  
**where** *instructor.ID = teaches.ID*;

OR

- **select** *name, course\_id*  
**from** *instructor natural join teaches*;



# Tutorial-1 07-02-2023

Write relational algebraic expressions for the following queries using schema given below.

*Product (Product\_no, Prod\_Name, Company, Cost\_price, Stock )*

*Sales(SaleNo, Product\_no, quantity) where Product\_no references Product*

1. List all the product\_name, company of the products with product number - P1, P2, P5, P7.
2. List all product\_names which are from company -TATA and cost\_price is more than 500.
3. List all the product numbers which are sold more than 20 in quantity.
4. Display the product number along with the total quantity sold.
5. Find product name of products for which stock is below 15.

# Tutorial 2

21-02-2023

Write the SQL-DDL commands to do the following:

1) Create the following tables

SALESMAN (Salesman\_id, Name, City, Commission)

CUSTOMER (Customer\_id, Cust\_Name, City, Grade, Salesman\_id)

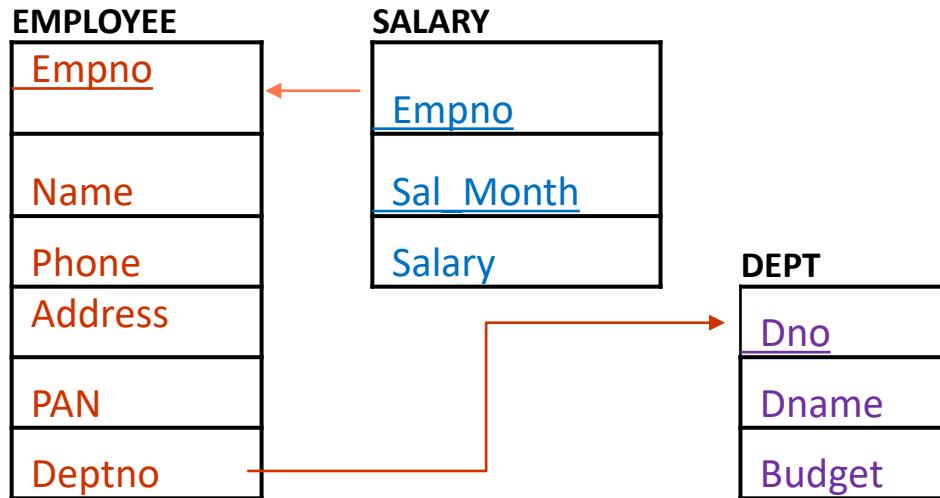
ORDERS (Ord\_No, Purchase\_Amt, Ord\_Date, Customer\_id, Salesman\_id)

- Underlined attributes are primary keys and other column with same names are foreign keys.
- Assume appropriate datatype and size. Give your own proper constraint names to the constraints
- Also impose following constraints-
  - Commission – minimum 1000 and maximum 20000
  - Grade – Silver or Gold or Diamond or Platinum
  - Minimum Purchase\_amount 500/-

2) Create a new column- **Points** number type into table Customer and put constraint minimum points 10;

# Try Writing Some Queries in SQL

- Suggest queries to be written.....



- Display the employee Names whose PAN number is **APQTY4679Z**
- Display name of employee along with dept name who is working in department number **10** and having PAN number **APHFR6945G**
- Display the name of employees who got salary more than **70000** during month **August.**
- Display the name of employees and their respective department names.
- Display the name of employees who are working in the '**Research**' department
- Display the name of employee, Salary and name of department in which he is working.

**DNO DNAME****BUDGET****DEPT**

10 Research	569080
11 Production	6798980
12 Finance	989808

**EMPNO ENAME****PHONE****ADDRESS****PAN****DEPTNO****EMPLOYEE**

101	Ashu	8980909091	Manipal	APQTY4679Z	10
102	Vemu	9800909091	Mumbai	APHFR6945G	10
103	Saru	8030909091	Mumbai	BPHFR9945G	11
104	Paru	7803909091	Pune	DPHFR9745G	11
105	Vishu	7865390909	Pune	ANHFR9748G	12

**EMPNO SALARY\_MON****SALARY**

101	August	65000
101	September	75000
102	August	174000
102	September	175000
103	August	94000
104	August	64000
105	August	65000
105	September	65500

**SALARY**

1) Display the employee Names whose PAN number is **APQTY4679Z**

```
SELECT ename  
FROM employee  
WHERE pan='APQTY4679Z';
```

**ENAME**

-----  
**Ashu**

2. Display name of employee along with dept name who is working in department number **10** and having PAN number **APHFR6945G**

```
SELECT ename,dname  
FROM employee join dept on employee.deptno=dept.dno  
WHERE deptno=10 and pan='APHFR6945G';
```

**ENAME**

**DNAME**

-----  
**Vemu**

**Research**

**3.**Display the name of employees who got salary more than **70000** during month **August**.

```
SELECT ename  
FROM employee natural join salary  
WHERE salary>70000 and salary_month='August';
```

**ENAME**

-----  
**Vemu**  
**Saru**

**4.**Display the name of employees and their respective department names

```
SELECT ename,dname  
FROM employee join dept on employee.deptno=dept.dno;
```

<b>ENAME</b>	<b>DNAME</b>
Ashu	Research
Vemu	Research
Saru	Production
Paru	Production
Vishu	Finance

5. Display the name, phone no, location of employees who are working in the 'Account' department

```
SELECT ename,phone,address  
FROM employee join dept on employee.deptno=dept.dno  
WHERE dname='Account';
```

ENAME	PHONE	ADDRESS
Ashu	8980909091	Manipal
Vemu	9800909091	Mumbai

6. Display the name of employee, month, salary and name of department in which he is working

```
SELECT ename,month_salary,salary,dname  
FROM employee e JOIN dept d ON e.deptno=d.dno  
JOIN salary s ON e.empno=s.empno;
```

ENAME	SALARY_MON	SALARY	DNAME
Ashu	August	65000	Research
Ashu	September	75000	Research
Vemu	August	174000	Research
Vemu	September	175000	Research
Saru	August	94000	Production
Paru	August	64000	Production
Vishu	August	65000	Finance
Vishu	September	65500	Finance

# Outer Join

The **join** operation combines data from two tables by forming pairs of related rows where the matching columns in each table have the same value. If one row of a table is unmatched, the row is omitted from the result table.

**Outer join** include the unmatched rows in the result table.

Three types of outer join:

- **Left**
- **Right**
- **Full**

# Join Example

BRANCH

BranchNo	BCity
B003	Glasgow
B004	Bristol
B002	London

PROPERTY

PropertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

BranchNo	BCity	PropertyNo	pCity
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

```
SELECT b.* , p.*  
FROM branch b, property p  
WHERE b.bcity = p.pcity;
```

In the example  
Branch b , property p,  
**b** and **p** are said to be  
Alias<sup>36</sup>names to the  
tables branch &  
property respectively.

# Left Outer Join

## Example:

List the branch offices and properties that are in the same city along with any unmatched branches.

```
SELECT b.* , p.*  
FROM branch b  
LEFT JOIN property p ON  
    b.bcity = p.pcity;
```

## Result of Left Outer Join

BRANCH

BranchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PROPERTY

PropertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

BranchNo	bCity	PropertyNo	pCity
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

```
SELECT b.* , p.*  
FROM branch b  
LEFT JOIN property p ON  
b.bcity = p.pcity;
```

# Right Outer Join

## Example:

List the branch offices and properties in the same city and any unmatched property.

```
SELECT b.* , p.*  
FROM branch b  
RIGHT JOIN property p ON  
    b.bcity = p.pcity;
```

## Result of Right Outer Join

BRANCH

BranchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PROPERTY

PropertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

BranchNo	bCity	PropertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PL94	London
B002	London	PG4	Glasgow

```
SELECT b.* , p.*  
FROM branch b  
RIGHT JOIN property p ON  
b.bcity = p.pcity;
```

# Full Outer Join

## Example:

List the branch offices and properties that are in the same city and any unmatched branches or properties.

```
SELECT b.* , p.*  
FROM branch b  
FULL JOIN property p ON  
    b.bcity = p.pcity;
```

## Result of FULL Outer Join

BRANCH

BranchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PROPERTY

PropertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

BranchNo	bCity	PropertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

```
SELECT b.* , p.*  
FROM branch b  
FULL JOIN property p ON  
b.bcity = p.pcity;
```

# The Rename Operation

- The SQL allows renaming relations and attributes using the **as** clause:

*old-name as new-name*

- E.g.

- **select** *ID, name, salary/12 as monthly\_salary*  
**from** *instructor*

- Keyword **as** is optional and may be omitted

*instructor as T*  $\equiv$  *instructor T*

- Keyword **as** must be omitted in Oracle

# Compare Tuples in the Same Relation

- Another reason to rename a relation is a case where we wish to compare tuples in the same relation. We then need to take the Cartesian product of a relation with itself.
- Since we need to join a tuple of relation with another tuple in the same relation, It becomes impossible to distinguish one tuple from another without renaming.
- **Example:** Find the names of all instructors who have a higher salary than some instructor in ‘Comp. Sci’.

□ **select distinct  $T.name$**

**from  $instructor T, instructor S$**

**where  $T.salary > S.salary$  and  $S.dept\_name = 'Comp. Sci.'$**

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
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

An identifier, such as  $T$  and  $S$ , that is used to rename a relation is referred to as a **correlation name** and is also commonly referred to as a **table alias**, or a **correlation variable**, or a **tuple variable**.

# Set Operations-UNION

- Display Employee names who are working in deptno 10 , 20.
- Assume the schema EMP(empno,ename,sal,deptno)

```
SELECT Ename  
FROM Emp WHERE deptno=10
```

**UNION**

```
SELECT Ename  
FROM Emp WHERE deptno=20 ;
```

The query is equivalent to-

```
SELECT Ename  
FROM Emp  
WHERE deptno=10 OR deptno=20;
```

# Set Operations-INTERSECT

- Find the name of employees working in department –‘RESEARCH’ and drawing salary more than 25000.
- Assume Emp(empno,ename,sal,deptno) & Dept(Deptno,Dname,City)
  - Emp.deptno is F.key referencing Dept.Deptno(p.key)

**Select** ename from emp,dept where **emp.deptno=dept.deptno**  
**and dname='RESEARCH'**

## INTERSECT

**Select** ename from emp,dept where **emp.deptno=dept.deptno**  
**and sal>25000;**

# **Set Operations-MINUS**

- Find ename and deptno from emp who are not working in deptno 20.

**Select** empno,ename,deptno from emp

**MINUS**

**Select** empno,ename,deptno from emp where deptno=20;

## Section

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

# Set Operations

- Find courses that ran in Fall 2009 or in Spring 2010

(**select course\_id from section where sem = 'Fall' and year = 2009**)  
**union**

(**select course\_id from section where sem = 'Spring' and year = 2010**)

- Find courses that ran in Fall 2009 and in Spring 2010

(**select course\_id from section where sem = 'Fall' and year = 2009**)  
**intersect**

(**select course\_id from section where sem = 'Spring' and year = 2010**)

- Find courses that ran in Fall 2009 but not in Spring 2010

(**select course\_id from section where sem = 'Fall' and year = 2009**)  
**except**

(**select course\_id from section where sem = 'Spring' and year = 2010**)

# Set Operations

- Set operations **union**, **intersect**, and **Minus**
  - Each of the above operations automatically **eliminates duplicates**
- To **retain all duplicates** use the corresponding multiset versions **union all**.

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

- Suppose a tuple occurs  $m$  times in  $r$  and  $n$  times in  $s$ , then, it occurs:
  - $m + n$  times in  $r \text{ union all } s$

# IS NULL/ IS NOT NULL

## Example:

List the details of all staff who have only first name but not last name.

```
SELECT sno, fname, lname, salary  
FROM staff  
WHERE lname IS NULL.
```

# Aggregate Functions

- These functions operate on the **multiset of values of a column** of a relation, and return a value

**avg:** average value

**min:** minimum value

**max:** maximum value

**sum:** sum of values

**count:** number of values

Consider the following relation to write SQL queries

Student(RollNo,Name,Mark1)

Roll No	Name	Mark1
101	Anu	45
102	Shan	44
103	Diya	30
104	Arun	25
105	Farah	30

□ **AVG()**

`SELECT AVG(column_name) FROM table_name WHERE condition;`

`SELECT AVG(Mark1) FROM Student;`                            AVG(Mark1)

34.8

□ **SUM()**

`SELECT SUM(column_name) FROM table_name WHERE condition;`

`SELECT SUM(Mark1) FROM Student;`                            SUM(Mark1)

- **COUNT()**

`SELECT COUNT(column_name) FROM table_name WHERE condition;`

`SELECT COUNT(Mark1) FROM Student;`

- **MAX()**

`SELECT MAX(column_name) FROM table_name WHERE condition;`

`SELECT MAX(Mark1) FROM Student;`

- **MIN()**

`SELECT MIN(column_name) FROM table_name WHERE condition;`

`SELECT MIN(Mark1) FROM Student;`

# Aggregate Functions (Cont.)

- Find the average salary of instructors in the Computer Science department

- **select avg (salary)**  
**from instructor**  
**where dept\_name= 'Comp. Sci.';**

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

- Find the total number of instructors who teach a course in the Spring 2010 semester

- **select count (distinct ID)**  
**from teaches**  
**where semester = 'Spring' and year = 2010**

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

- Find the number of tuples in the course relation

- **select count (\*) from course;**

# Aggregate Functions – Group By

- Find the average salary of instructors in each department
  - `select dept_name, avg (salary)  
from instructor  
group by dept_name;`
  - Note: departments with no instructor will not appear in result

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

<i>dept_name</i>	<i>avg_salary</i>
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

# Aggregation (Cont.)

- Attributes in **select** clause outside of aggregate functions must appear in **group by** list
  - /\* erroneous query \*/

```
select dept_name, ID, avg (salary)  
from instructor  
group by dept_name;
```

# Aggregate Functions – Having Clause

- Find the names and average salaries of all departments whose average salary is greater than 42000

```
select dept_name, avg (salary)  
from instructor  
group by dept_name  
having avg (salary) > 42000;
```

Note: predicates in the **having** clause **are applied after the formation of groups** whereas predicates in the **where** clause are applied **before forming groups**

# Null Values and Aggregates

- Total all salaries

```
select sum (salary)  
from instructor
```

- Above statement **ignores** null amounts
- Result is **null** if there is **no** non-null amount
- All aggregate operations **except count(\*)** **ignore tuples with null** values on the aggregated attributes
- What if collection has only null values?
  - **count** returns **0**
  - **all other aggregates return null**

# Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries
- A **subquery** is a **select-from-where** expression that is nested within another query
- The subquery can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery
- A subquery is also called an inner query or inner select, while the statement containing a subquery is also called an outer query or outer select
- The inner query executes first before its parent query so that the results of an inner query can be passed to the outer query

# Nested Subqueries

- A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality
- Syntax:

```
SELECT      select_list
FROM        table
WHERE       expr operator
            (SELECT      select_list
             FROM        table);
```

# Nested Subqueries

**Student**

StudentId	Name
V001	Ajay
V002	Asha
V003	Anu
V004	Sona

**Marks**

StudentId	Total_marks
V001	95
V002	80
V003	74
V004	81

Write a query to identify all students who get better marks than that of the student who's StudentID is 'V002'

# Query

```
SELECT a.studentid, a.name, b.total_marks  
FROM student a, marks b  
WHERE a.studentid = b.studentid AND b.total_marks >80;
```

```
SELECT a.studentid, a.name, b.total_marks  
FROM student a, marks b  
WHERE a.studentid = b.studentid  
AND b.total_marks >  
(SELECT total_marks FROM marks WHERE studentid =  
'V002');
```

# Subqueries

There are some guidelines to consider when using subqueries :

- A subquery must be enclosed in parentheses.
- A subquery must be placed on the right side of the comparison operator.
- Subqueries cannot manipulate their results internally, therefore ORDER BY clause cannot be added into a subquery. You can use an ORDER BY clause in the main SELECT statement (outer query) which will be the last clause.
- Use single-row operators with single-row subqueries.
- If a subquery (inner query) returns a null value to the outer query, the outer query will not return any rows when using certain comparison operators in a WHERE clause.

# Types of Subquery

- Single row subquery : Returns zero or one row.
- Multiple row subquery : Returns one or more rows.
- Multiple column subqueries : Returns one or more columns.
- Correlated subqueries : Reference one or more columns in the outer SQL statement. The subquery is known as a correlated subquery because the subquery is related to the outer SQL statement.
- Nested subqueries : Subqueries are placed within another subquery.

# Set Membership

- SQL allows testing tuples for membership in a relation.
- The **in** connective tests for set membership, where the set is a collection of values produced by a **select** clause.
- The **not in** connective tests for the absence of set membership.

<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>	<i>building</i>	<i>room_number</i>	<i>time_slot_id</i>
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

# Example Query

- Find courses offered in Fall 2009 and in Spring 2010

```
select distinct course_id  
from section  
where semester = 'Fall' and year= 2009 and  
course_id in (select course_id  
               from section  
               where semester = 'Spring' and year= 2010);
```

- Find courses offered in Fall 2009 but not in Spring 2010

```
select distinct course_id  
from section  
where semester = 'Fall' and year= 2009 and  
course_id not in (select course_id  
                   from section  
                   where semester = 'Spring' and year= 2010);
```

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

Takes

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

Teaches

# Example Query

- Find the total number of (distinct) students who have taken course sections taught by the instructor with *ID* 10101

**SELECT COUNT(distinct S.ID)**

**FROM takes S join teaches T on S.course\_id=T.course\_id and S.sec\_id=T.sec\_id and S.semester=T.semester and S.year=T.year  
WHERE T.ID=10101.**

OR

```
select count (distinct ID)
from takes
where (course_id, sec_id, semester, year) in
    (select course_id, sec_id, semester, year
     from teaches
     where teaches.ID= 10101);
```

# Set Comparison

- Find names of instructors with salary greater than that of some (at least one) instructor in the Comp.Sci.department.

```
select distinct T.name
```

```
from instructor as T, instructor as S
```

```
where T.salary > S.salary and S.dept_name  
= 'Comp.Sci.';
```

- Same query using **> some** clause

```
select name
```

```
from instructor
```

```
where salary > some (select salary
```

```
from instructor
```

```
where dept_name = 'Comp.Sci.');
```

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

Instructor

# Definition of Some Clause

- $F \text{ <comp> } \text{some } r \Leftrightarrow \exists t \in r \text{ such that } (F \text{ <comp> } t)$

Where  $\text{<comp>}$  can be:  $<$ ,  $\leq$ ,  $>$ ,  $=$ ,  $\neq$

$(5 < \text{some} \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{true}$  (read: 5 < some tuple in the relation)

$(5 < \text{some} \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{false}$

$(5 = \text{some} \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{true}$

$(5 \neq \text{some} \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline \end{array}) = \text{true}$  (since  $0 \neq 5$ )

$(= \text{some}) \equiv \text{in}$

However,  $(\neq \text{some}) \not\equiv \text{not in}$

**SQL> select empno, ename,sal,deptno from emp order by deptno;**

EMPNO	ENAME	SAL	DEPTNO
7934	MILLER	10000	10
7782	CLARK	2450	10
7839	KING	5000	10
1000	DDD	3333	10
7566	JONES	2975	20
7369	SMITH	800	20
1001	EEEE	2005	20
7902	FORD	3000	20
7876	ADAMS	1100	20
7788	SCOTT	3000	20
7900	JAMES	950	30
7499	ALLEN	1600	30
7844	TURNER	1500	30
50	BBB	777	30
7521	WARD	1250	30
7698	BLAKE	2850	30
7654	MARTIN	1250	30

Find the Employee No, Sal and Deptno of employees who earn salary more than any employee in Dept 20

SQL> select sal, deptno from emp where deptno=20;

SAL	DEPTNO
2005	20
800	20
2975	20
3000	20
1100	20
3000	20

SQL> select empno,sal,deptno from emp where  
sal > some(select sal from emp where deptno=20);

EMPNO	SAL	DEPTNO
-------	-----	--------

7934	10000	10
7839	5000	10
1000	3333	10
7788	3000	20
7902	3000	20
7566	2975	20
7698	2850	30
7782	2450	10
1001	2005	20
7499	1600	30
7844	1500	30
7654	1250	30
7521	1250	30
7876	1100	20
7900	950	30

50 777 30  
will not appear as  
result here, because  
salary 777 is lesser  
than salary of all the  
employees in the  
department 20

## Example: Find employee name and their manager names.

EMP(Empno,Ename,Mgrno,Deptno)

**SQL>** select s.empno ,t.empno mgr\_no,s.ename,t.ename mgr\_name from emp  
s, emp t where **s.mgr\_no=t.empno;**

EMPNO	MGR_NO	ENAME	MGR_NAME
7902	7566	FORD	JONES
7788	7566	SCOTT	JONES
7900	7698	JAMES	BLAKE
7844	7698	TURNER	BLAKE
7654	7698	MARTIN	BLAKE
7521	7698	WARD	BLAKE
7499	7698	ALLEN	BLAKE
7934	7782	MILLER	CLARK
7876	7788	ADAMS	SCOTT
7782	7839	CLARK	KING
7698	7839	BLAKE	KING
7566	7839	JONES	KING
7369	7902	SMITH	FORD

# Example Query

- Find the names of all instructors whose salary is greater than the salary of all instructors in the Biology department.

```
select name  
from instructor  
where salary > all (select salary  
                      from instructor  
                     where dept_name = 'Comp.Sci.');
```

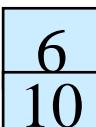
Instructor

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

# Definition of all Clause

- $F \text{ <comp> } \mathbf{all} \ r \Leftrightarrow \forall t \in r (F \text{ <comp> } t)$

$(5 < \mathbf{all}$   ) = false

$(5 < \mathbf{all}$   ) = true

$(5 = \mathbf{all}$   ) = false

$(5 \neq \mathbf{all}$   ) = true (since  $5 \neq 4$  and  $5 \neq 6$ )

$(\neq \mathbf{all}) \equiv \mathbf{not} \ \mathbf{in}$

However,  $(= \mathbf{all}) \neq \mathbf{in}$

**SQL> select empno, ename,sal,deptno from emp order by deptno;**

EMPNO	ENAME	SAL	DEPTNO
7934	MILLER	10000	10
7782	CLARK	2450	10
7839	KING	5000	10
1000	DDD	3333	10
7566	JONES	2975	20
7369	SMITH	800	20
1001	EEEE	2005	20
7902	FORD	3000	20
7876	ADAMS	1100	20
7788	SCOTT	3000	20
7900	JAMES	950	30
7499	ALLEN	1600	30
7844	TURNER	1500	30
50	BBB	777	30
7521	WARD	1250	30
7698	BLAKE	2850	30
7654	MARTIN	1250	30

**Find the Employee No, Sal and Deptno of employees who earn salary more than every employee in Dept 20**

**SQL> select sal,deptno from emp where deptno=20;**

**SAL DEPTNO**

SAL	DEPTNO
2005	20
800	20
2975	20
3000	20
1100	20
3000	20

**SQL> select empno,sal,deptno from emp where sal > all ( select sal from emp where deptno=20);**

**EMPNO SAL DEPTNO**

EMPNO	SAL	DEPTNO
1000	3333	10
7839	5000	10
7934	10000	10

- Write the previous query by comparing salary of employees with maximum salary of employees in the department 20;

**Select empno,sal,deptno from emp where sal >**

**( Select max(sal) From Emp where deptno=20);**

# Correlated Subquery

- Correlated subqueries are used for row-by-row processing
- Each subquery is executed once for every row of the outer query.
- A correlated subquery is evaluated once for each row processed by the parent statement
- The parent statement can be a **SELECT**, **UPDATE**, or **DELETE** statement.

**SELECT** column1, column2, ....

**FROM** table1 outer

**WHERE** column1 operator

(**SELECT** column1, column2  
**FROM** table2  
**WHERE** expr1 = outer.expr2 );

# Correlated Subquery

- Find all the instructors who earn more than their department average salary.

```
SELECT name, salary, dept_name  
FROM instructor outer  
WHERE salary >  
(SELECT AVG(salary)  
FROM instructor  
WHERE dept_name= outer.dept_name);
```

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

dept_name	avg_salary
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

# Correlated Subquery

## □ Use of Exists and Not Exists

- The EXISTS operator tests for existence of rows in the results set of the subquery
- If a subquery row value is found the condition is flagged **TRUE** and the search does not continue in the inner query, and if it is not found then the condition is flagged **FALSE** and the search continues in the inner query

# Correlated Subquery

- “Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester”

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

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

- **select course\_id  
from section S  
where semester = 'Fall' and year= 2009**
  - Result is –
  - course\_id={**CS-101,CS-347,PHY-101**}
  
  - **select \*  
from section T  
where semester = 'Spring' and year= 2010**
  - Result is-  
**{<CS-101,...>,< CS-315,...>,<CS-319,...>,<FIN-201,...>,<HIS-351,...>,<MU-199,...>}**
- 
- and S.course\_id= T.course\_id);**
- CS-101=CS-101 YES, inner query return row(row exists satisfying)
- CS-347=CS-101/CS-325/....../MU-199 NO, No row exists.
- PHY-101= CS-101/CS-325/....../MU-199 NO, No row exists

## Find the customers who have placed order on the date 2016/04/18

**Customer**

customer_id	last_name	first_name
4000	Jackson	Joe
5000	Smith	Jane
6000	Ferguson	Samantha
7000	Reynolds	Allen
8000	Anderson	Paige
9000	Johnson	Derek

**Order**

order_id	customer_id	order_date
1	7000	2016/04/18
2	5000	2016/04/18
3	8000	2016/04/19
4	4000	2016/04/20
5	NULL	2016/05/01

Select First\_Name FROM Customer natural join Order where  
order\_date=to\_date('2016/04/18','yyyy/mm/dd'));

### Equivalently using IN

Select First\_Name FROM Customer Where Customer\_id IN ( select Customer\_id  
From Order where order\_date=to\_date('2016/04/18','yyyy/mm/dd'));

### Equivalently using EXISTS

SELECT First\_Name FROM customers **T** WHERE **EXISTS** (SELECT \* FROM  
order\_details WHERE **T**.customer\_id = order\_details.customer\_id and  
order\_date=to\_date('2016/04/18','yyyy/mm/dd'));

# Subqueries in the From Clause

- SQL allows a subquery expression to be used in the from clause.
- Find the average salaries of those departments where the average salary is greater than \$42,000.

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

- Note that we do not need to use the **having** clause
- Another way to write above query**

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

dept_name	avg_salary
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

# 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.

**Syntax:** WITH query\_name As ( SQL query ) SELECT \* FROM query\_name;

- Consider the tables Emp(Empno,Ename,Sal,deptno)
- Dept(Dno,Dname,location)

```
with max_sal(value) as (select max(sal) From emp)
select Dname,sal from emp,Dept, max_sal where
Deptno=dno and emp.sal=max_sal.value;
```

# With Clause

- Consider department(dept\_name,building,budget)
- Find all departments with the maximum budget

**with** *max\_budget* (*value*) **as**

(**select** **max**(*budget*)

**from** *department*)

**select** *dept\_name*

**from** *department*, *max\_budget*

**where** *department.budget* = *max\_budget.value*;

# With Clause

Consider the table Emp(Empno,Ename,Sal,deptno)

Display the details of employee with salary greater than average salary of all employees

WITH temporaryTable(averageValue) as

(SELECT avg(Sal)

from Emp)

SELECT Empno,Ename, Sal

FROM Emp, temporaryTable

WHERE Emp.Sal > temporaryTable.averageValue;

# **CREATE .. AS.. SELECT...**

**Create a table with the same schema as an existing table**

## **Syntax:**

```
CREATE TABLE <new_table_Name> AS SELECT  
<column1>,<column2>,... FROM <existing_table_name>  
WHERE <condition>;
```

## **Example:**

- Create a table Emp\_Copy1 with the same schema as an existing Emp table:

```
CREATE TABLE Emp_Copy1 AS SELECT * FROM EMP;
```

## **Example:**

- Create a table Emp\_Copy2 with columns Empno,Ename,Sal from an existing Emp table:

```
CREATE TABLE Emp_Copy2 AS SELECT Empno,Ename,Sal FROM EMP;
```

# **CREATE .. AS.. SELECT...**

**Create a table with the same schema as an existing table**

**Example:**

- Create a table Emp\_Copy3 with columns Eno,Name,Salary by taking the column structure information from an existing Emp table:

```
CREATE TABLE Emp_Copy3(Eno,Name,Salary) AS SELECT  
          Empno,Ename,Sal FROM EMP;
```

**Create.. AS.. Select .. Command** not only copies the structure of select copies but **also copies data from** those columns.

**Example:**

```
CREATE TABLE Emp_Copy3(Eno,Name,Salary) AS SELECT  
          Empno,Ename,Sal FROM EMP WHERE Deptno=10;
```

## Views

- In some cases, it is **not desirable for all users to see the entire logical model** (that is, all the actual relations stored in the database.)
- Consider a person who needs to know an instructors name and department, but not the salary. This person should see a relation described, in SQL, by

```
select ID, name, dept_name from instructor;
```

- A **view** provides a mechanism to hide certain data from the view of certain users.
- Any relation that is not of the conceptual model but is made visible to a user as a “**virtual relation**” is called a **view**.

## View Definition

- A view is defined using the **create view** statement which has the form  
**create view v as < query expression >**

where <query expression> is any legal SQL expression. The view name is represented by v.
- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- View definition is not the same as creating a new relation by evaluating the query expression
  - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.

## Example:

- create view V1 as select \* from emp;

Name	Null?	Type
EMPNO	NOT NULL	NUMBER(4)
ENAME		VARCHAR2(10)
JOB		VARCHAR2(9)
MGR		NUMBER(4)
HIREDATE		DATE
SAL		NUMBER(7,2)
COMM		NUMBER(7,2)
DEPTNO		NUMBER(2)

## Example: Renaming column names in the View

```
create view V2(employee_number, Name, Date_of_Join) as select  
empno,ename,hiredate from emp;
```

Name	Null?	Type
EMPLOYEE_NUMBER	NOT NULL	NUMBER(4)
NAME		VARCHAR2(10)
DATE_OF_JOIN		DATE

## Example:

```
create view v3 as select job, avg(sal) Avg_sal from emp group by job;
```

Name	Null?	Type
JOB		VARCHAR2(9)
AVG_SAL		NUMBER

## Views Defined Using Other Views

- One view may be used in the expression defining another view
- A view relation  $v_1$  is said to *depend directly* on a view relation  $v_2$  if  $v_2$  is used in the expression defining  $v_1$

## Example: Creating a View from another view

- create view v5 as select employee\_number,name from v2 where date\_of\_join>'17-DEC-1980';

```
SQL> desc v5;
   Name          Null?    Type
-----  -----
EMPLOYEE_NUMBER      NOT NULL NUMBER(4)
NAME                  VARCHAR2(10)
```

## Example Views

- A view of instructors without their salary
  - create view faculty as**  
**select ID, name, dept\_name**  
**from instructor**
- Find all instructors in the Biology department
  - select name**  
**from faculty**  
**where dept\_name = 'Biology'**

- Create a view of department salary totals
  - create view departments\_total\_salary(dept\_name, total\_salary) as**  
**select dept\_name, sum (salary)**  
**from instructor**  
**group by dept\_name;**

## Views Defined Using Other Views

- **create view** *physics\_fall\_2009* **as**  
**select** course.course\_id, sec\_id, building, room\_number  
**from** course, section  
**where** course.course\_id = section.course\_id  
    **and** course.dept\_name = 'Physics'  
    **and** section.semester = 'Fall'  
    **and** section.year = '2009';
- **create view** *physics\_fall\_2009\_watson* **as**  
**select** course\_id, room\_number  
**from** *physics\_fall\_2009*  
**where** building= 'Watson';

## Querying a View

```
SQL> select * from v5;
```

EMPLOYEE_NUMBER	NAME
7499	ALLEN
7521	WARD
7566	JONES
7654	MARTIN
7698	BLAKE
7782	CLARK
7788	SCOTT
7839	KING
7844	TURNER
7876	ADAMS
7900	JAMES

```
-----  
7499 ALLEN  
7521 WARD  
7566 JONES  
7654 MARTIN  
7698 BLAKE  
7782 CLARK  
7788 SCOTT  
7839 KING  
7844 TURNER  
7876 ADAMS  
7900 JAMES
```

## View Expansion

- A way to define the meaning of views defined in terms of other views.
- Replace the view relation  $v_i$  by the expression defining  $v_i$

Assume V2 is created and v5 created using V2.

- **create view V2(employee\_number, Name, Date\_of\_Join) as select empno,ename,hiredate from emp;**
- **create view v5 as select employee\_number,name from v2 where date\_of\_join>'17-DEC-1980';**

## Update of a View

- Add a new tuple to *faculty* view which we defined earlier

```
insert into faculty values ('30765', 'Green', 'Music');
```

This insertion must be represented by the insertion of the tuple  
('30765', 'Green', 'Music', null) into the *instructor* relation

```
insert into v2 values(7868,'Ravi','10-Oct-2012');
```

```
SQL> desc v2;
Name          Null?    Type
-----        -----
EMPLOYEE_NUMBER      NOT NULL NUMBER(4)
NAME            VARCHAR2(10)
DATE_OF_JOIN      DATE
```

```
select empno,ename,hiredate from emp;
```

```
SQL> select empno,ename,hiredate from emp;
EMPNO  ENAME       HIREDATE
-----  -----
      50  BBB
     1212  XXX
     1213
     7868  Ravi      10-OCT-12
     1000  DDD
     1001  EEEE
     7369  SMITH     17-DEC-80
```

## Some Updates cannot be Translated Uniquely

- Assume the following view is created-
- **create view** *instructor\_info* **as**  
**select** *ID, name, building* **from** *instructor, department*  
**where** *instructor.dept\_name= department.dept\_name*;
- **insert into** *instructor\_info* **values** ('69987', 'White', 'Taylor');
  - ▶ which department, if multiple departments in Taylor?
  - ▶ what if no department is in Taylor?
- Most SQL implementations **allow updates only on simple views**
  - The **from** clause has only one database relation.
  - The **select** clause contains only attribute names of the relation, and does not have any expressions, aggregates, or **distinct** specification.
  - Any attribute not listed in the **select** clause can be set to null
  - The query does not have a **group by** or **having** clause.