# Chapter 3 SQL



- Background
- Basic Structure query
- DDL
- DML(Insert ,Delete,Update)
- Embedded SQL
- Dynamic SQL
- ODBC,JDBC
- Transact\_SQL



- Structured query language (SQL)
  - 1970s, IBM Sequel(System R)
  - 1986, ANSI and ISO, **SQL\_86**;
- 1987, IBM, SAA\_SQL;
  - 1989, ANSI, SQL\_89, an extended standard for SQL;
- SQL 92;
  - 1999 SQL;
  - SQL:2003



## **Background**

- Data definition language (DDL);
- Data manipulation language (DML);
- Data control language (DCL);

Integrity;

Authorization;

SQL is based on set and relational operations with certain modifications and enhancements

Embedded SQL and dynamic SQL;



# Basic Structure\_\_query

A typical SQL query has the form:

select 
$$A_1$$
,  $A_2$ , ...,  $A_n$  from  $r_1$ ,  $r_2$ , ...,  $r_m$  where  $P$ 

A<sub>i</sub> represent attributes

 $r_i$  represent relations

P is a predicate.

 This query is equivalent to the relational algebra expression.

$$\prod_{A_1, A_2, \dots, A_n} (\sigma_P(r_1 \times r_2 \times \dots \times r_m))$$

The result of an SQL query is a relation



#### The select Clause

- The **select** clause corresponds to the projection operation of the relational algebra. It is used to list the attributes desired in the result of a query.
- Find the names of all instructors:

select name from instructor

 In the "pure" relational algebra syntax, the query would be:

 $\prod_{\mathsf{name}}(instructor)$ 

An asterisk in the select clause denotes "all attributes"

**select** \* **from** *instructor* 



## The select Clause

■ 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



#### The select Clause

- •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 department names of all instructors, and remove duplicates

select distinct dept\_name from instructor

• The keyword all specifies that duplicates should not be removed.

select all dept\_name from instructor



## The where Clause

The where clause corresponds to the selection predicate of the relational algebra.

• Find all instructors in Comp. Sci. dept

select name from instructor where dept\_name = 'Comp. Sci.'



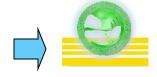
## The where Clause

- Comparison results can be combined using the logical connectives and, or, and not
  - To find all instructors in Comp. Sci. dept with salary > 80000

select name from instructor where dept\_name = 'Comp. Sci.' and salary > 80000

Comparisons can be applied to results of arithmetic expressions.

Between..and



- SQL includes a between comparison operator
  - Example: Find the names of all instructors with salary between \$90,000 and \$100,000
    - select namefrom instructorwhere salary between 90000 and 100000
  - Tuple comparison
    - select name, course\_id
      from instructor, teaches
      where (instructor.ID, dept\_name) =
       (teaches.ID, 'Biology');



## The from Clause

- The from clause corresponds to the Cartesian product operation of the relational algebra. It lists the relations to be scanned in the evaluation of the expression.
  - Find the Cartesian product instructor X teaches

select \*

from instructor, teaches

For common attributes (e.g., ID), the attributes in the resulting table are renamed using the relation name (e.g., instructor.ID)

• Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra).

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
00456	Lan	731	0=000

					2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 -
	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
. 1	22222	PHY-101	1	Fall	2009

Inst.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
(*) * *)	***	•••	•••	•••			•••	** *
			•:•:•	•••				***
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Pinance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Pinance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Pinance	90000	22222	PHY-101	1	Fall	2009
•••	N••(1•)	•••	• • •		• • •		•••	
• • •			• • •	•••	•••		*** *	1*(1*11*)

## **Examples**

- Find the names of all instructors who have taught some course and the course\_id
  - select name, course\_id from instructor, teaches where instructor.ID = teaches.ID
  - Find the names of all instructors in the Art department who have taught some course and the course\_id

```
select name, course_id
from instructor, teaches
where instructor.ID = teaches.ID and instructor.
dept_name = 'Art'
```



# The Rename Operation

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

```
old_name as new_name
```

 Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept_name = 'Comp.
Sci.'
```

Keyword as is optional and may be omitted instructor as  $T \equiv instructor T$ 





# String Operations

- percent (%). The % character matches any substring;
  - underscore (\_). The \_ character matches any character;

 Find the names of all instructors whose name includes the substring "dar".

> select name from instructor where name like '%dar%'



# **String Operations**

■Match the name "Main%"

like 'Main\%' escape '\'

- ■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.



# Ordering the Display of Tuples

List in alphabetic order the names of all instructors

select distinct name from instructor order by name

 We may specify desc for descending order or asc for ascending order, for each attribute; ascending order is the default.

Example: order by name desc

Can sort on multiple attributes

Example: order by dept\_name, name



# **Duplicates**

- In relations with duplicates, SQL can define how many copies of tuples appear in the result.
- Multiset versions of some of the relational algebra operators given multiset relations  $r_1$  and  $r_2$ :
  - 1.  $\sigma_{\theta}(r_1)$ : If there are  $c_1$  copies of tuple  $t_1$  in  $r_1$ , and  $t_1$  satisfies selections  $\sigma_{\theta}$ , then there are  $c_1$  copies of  $t_1$  in  $\sigma_{\theta}(r_1)$ .
  - 2.  $\Pi_A(r)$ : For each copy of tuple  $t_I$  in  $r_1$ , there is a copy of tuple  $\Pi_A(t_1)$  in  $\Pi_A(r_1)$  where  $\Pi_A(t_1)$  denotes the projection of the single tuple  $t_I$ .
  - 3.  $r_1 \times r_2$ : If there are  $c_1$  copies of tuple  $t_1$  in  $r_1$  and  $c_2$  copies of tuple  $t_2$  in  $r_2$ , there are  $c_1 \times c_2$  copies of the tuple  $t_1$ .  $t_2$  in  $r_1 \times r_2$

• Example: Suppose multiset relations  $r_1$  (A, B) and  $r_2$  (C) are as follows:

$$r_1 = \{(1, a) (2,a)\}$$
  $r_2 = \{(2), (3), (3)\}$ 

• Then  $\Pi_B(r_1)$  would be  $\{(a), (a)\}$ , while  $\Pi_B(r_1) \times r_2$  would be

$$\{(a,2), (a,2), (a,3), (a,3), (a,3), (a,3)\}$$

• SQL duplicate semantics:

select 
$$A_1, A_2, ..., A_n$$
  
from  $r_1, r_2, ..., r_m$   
where  $P$ 

is equivalent to the *multiset* version of the expression:



## **Set Operations**

- The set operations union, intersect, and except operate on relations and correspond to the relational algebra operations  $\cup$ ,  $\cap$ , -.
- Each of the above operations automatically eliminates duplicates; to retain all duplicates use the corresponding multiset versions union all, intersect all and except all.



# Set Operations\_Example

year = 2010)

- 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

# **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



## Aggregate Functions Example DataBase System Concepts

- Find the average salary of instructors in the Computer Science department
  - select avg (salary)
     from instructor
     where dept\_name= 'Comp. Sci.';
- 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;
- 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) as avg\_salary from instructor group by dept\_name;

Note: 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 - Group By DataBase System Concepts

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



## Aggregate Functions Having clause DataBase System Concepts

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**

- 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 result of any arithmetic expression involving *null* is *null* 
      - Example: 5 + null returns null
    - The predicate is null can be used to check for null values.
      - Example: Find all instructors whose salary is null.

select name from instructor where salary is null



## **Null Values**

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



## **Null Values**

- Any comparison with *null* returns *unknown* 
  - Example: 5 < null or null <> null or null = null
- Three-valued logic using the truth value *unknown*:
  - OR: (unknown **or** true) = true, (unknown **or** false) = unknown (unknown **or** unknown) = unknown
  - AND: (true and unknown) = unknown,
     (false and unknown) = false,
     (unknown and unknown) = unknown
  - NOT:  $(\mathbf{not} \ unknown) = unknown$
  - "P is unknown" evaluates to true if predicate P
     evaluates to unknown
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*

# **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 nesting can be done in the following SQL query

```
select A_1, A_2, ..., A_n
from r_1, r_2, ..., r_m
where P
```

#### as follows:

- A<sub>i</sub> can be replaced be a subquery that generates a single value.
- $r_i$  can be replaced by any valid subquery
- P can be replaced with an expression of the form:

Where B is an attribute and operation> to be defined ater.

Subqueries in the Where Clause





- A common use of subqueries is to perform tests:
  - For set membership
  - For set comparisons
  - For set cardinality.



# Set Membership

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
```





- Set Membership (Cont.)
- Find the total number of (distinct) students who have taken course sections taught by the instructor with ID 10101
- select count (distinct ID) from takes
- - Note: Above query can be written in a much simpler manner.
    - The formulation above is simply to illustrate SQL features.



# Set Comparison - "some" Clause Data Base System Concepts

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

> select distinct T.name from instructor as T, instructor as S where T.salary > S.salary and S.dept name = 'Biology';

Same query using > some clause

select name

from instructor
where salary > some (select salary
from instructor
where dept name = 'Biology

#### **Definition of Some Clause**

• F <comp> some  $r \Leftrightarrow \exists t \in r \text{ such that } (F <$ comp> t) Where <comp> can be: <,  $\leq$ , >, =,  $\neq$ 

$$(5 < \mathbf{some} \begin{vmatrix} 0 \\ 5 \end{vmatrix}) = \text{true}$$

$$(5 < \mathbf{some} \ \boxed{\frac{0}{5}}) = \text{false}$$

$$(5 = \mathbf{some} \boxed{\frac{0}{5}}) = \text{true}$$

$$(5 \neq \mathbf{some} \ \boxed{\frac{0}{5}}) = \text{true (since } 0 \neq 5)$$



#### Set Comparison – "all" Clause

- 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 = 'Biology');



#### Definition of All Clause

•  $F < comp > all r \Leftrightarrow \forall t \in r (F < comp > t)$ 

$$(5 < \mathbf{all} \quad \boxed{\frac{6}{10}}) = \text{true}$$

$$(5 = \mathbf{all} \mid \frac{4}{5} \mid ) = \text{false}$$

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



## **Test for Empty Relations**

- •The **exists** construct returns the value **true** if the argument subquery is nonempty.
- •exists  $r \Leftrightarrow r \neq \emptyset$
- •not exists  $r \Leftrightarrow r = \emptyset$



• Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester"

**2010** 

and S.course\_id = T.course\_id);

- Correlation name variable S in the outer query
- Correlated subquery the inner query



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

- Note that  $X Y = \emptyset \iff X \subseteq Y$
- Note: Cannot write this query using = all and its variants

- The unique construct tests whether a subquery has any duplicate tuples in its result.
  - The unique construct evaluates to "true" if a given subquery contains no duplicates.
  - Find all courses that were offered at most once in 2009



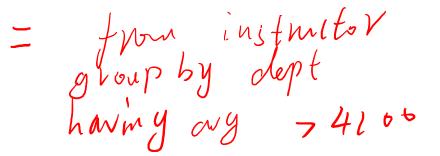
Subqueries in the From Clause



#### 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
from (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
- Another way to write above query



#### 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.
- 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;
```



 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) as
    (select avg(value)
    from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept total.value > dept total avg.value;
```



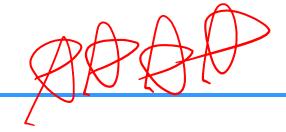
Subqueries in the Select Clause



#### Scalar Subquery

- Scalar subquery is one which is used where a single value is expected
  - List all departments along with the number of instructors in each department

Runtime error if subquery returns more than one result tuple



Select customer\_name

From borrower

Where exists (select \*

from depositor

where depositor.customer\_name=borrower.customer\_name)

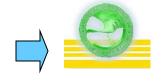
Set operation:



in, hot in, >some/all, <=some/all, =some/all, <>some/all,

exists, not exists,

unique,not unique



```
SELECT [DISTINCT] <Attributes list>
FROM < relations list>

[WHERE < predicate >]
       [GROUP BY < Attributes list >
               [ HAVING < Attributes list >] ]
        [ORDER BY < Attribute > [ASC|DESC], ..., <
 Attribute > [ASC|DESC] ];
```



#### Data Definition Language (DDL) DataBase System Concepts

- Allows the specification of not only a set of relations but also information about each relation, including:
  - The schema for each relation.
  - The domain of values associated with each attribute.
  - Integrity constraints
  - •The set of indices to be maintained for each relations.
  - Security and authorization information for each relation.
  - The physical storage structure of each relation on disk



#### Create Table Construct

- create table r ( $A_1$   $D_1$ ,  $A_2$   $D_2$ , ...,  $A_n$   $D_n$ , (integrity\_constraint<sub>1</sub>), ..., (integrity\_constraint<sub>k</sub>))
- r is the name of the relation each  $A_i$  is an attribute name in the schema of relation r  $D_i$  is the data type of values in the domain of attribute  $A_i$

Example:

```
create table instructor (
ID char(5), 2k
name varchar(20), k
dept_name varchar(20),
salary numeric(8,2))
```



### **Integrity Constraints**

- not null
- primary key  $(A_1, ..., A_n)$
- •Foreign Key (A1,...,An) reference table r on delete restrict/cascade/set NULL
- **check** (P), where P is a predicate

```
create table instructor (
                       char(5),
                      varchar(20) not null,
            name
           dept name varchar(20),
           salary numeric(8,2),
           primary key (ID),
           foreign key (dept_name) references
department);
             百八月参照关约的主建石
```

#### And a Few More Relation Definitions DataBase System Concepts

```
create table student (
    ID
                  varchar(5),
                  varchar(20) not null,
    name
    dept_name varchar(20),
                  numeric(3,0),
    tot cred
    primary key (ID),
    foreign key (dept_name) references department);
create table takes (
                 varchar(5),
    ID
    course_id varchar(8),
    sec_id varchar(8),
    semester varchar(6),
    year
                 numeric(4,0),
                 varchar(2),
    grade
    primary key (ID, course_id, sec_id, semester, year) ,
    foreign key (ID) references student,
    foreign key (course_id, sec_id, semester, year) references section);
```

 Note: sec\_id can be dropped from primary key above, to ensure a student cannot be registered for two sections of the same cours same semester

```
create table course (
    course_id varchar(8),
    title varchar(50),
    dept_name varchar(20),
    credits numeric(2,0),
    primary key (course_id),
    foreign key (dept_name) references
    department);
```



#### Domain Types in SQL

- •char(n). Fixed length character string, with user\_specified length n.
  - •varchar(n). Variable length character strings, with user\_specified maximum length n.
  - •int. Integer (a finite subset of the integers that is machine\_dependent).
  - •smallint. Small integer (a machine\_dependent subset of the integer domain type).
- •numeric(p,d). Fixed point number, with user\_specified precision of p digits, with n digits to the right of decimal point.
- •real, double precision. Floating point and double\_precision floating point numbers, with machine\_dependent precision.
- •float(n). Floating point number, with user\_specified precision of at least n digits.

### Drop and Alter Table Constructs DataBase System Concepts

- The drop table command deletes all information about the dropped relation from the database.
- •The **alter table** command is used to add attributes to an existing relation.

#### alter table r add A D

where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A*.



#### **Views**

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

create view v as <query expression>

where:

- P<query expression> is any legal expression
- PThe 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.

■When a view is created, the query expression is stored in the database; the expression is substituted into queries using the view.



#### Views \_Example

A view consisting of branches and their customers

```
create view all_customer as
  (select branch name, customer name
  from depositor, account
  where depositor.account_number =
 account.account number)
   union
 (select branch_name, customer_name
  from borrower, loan
  where borrower.loan number = loan.loan number)
Find all customers of the Perryridge branch
       select customer name
       from all customer
       where branch name = 'Perryridge'
```

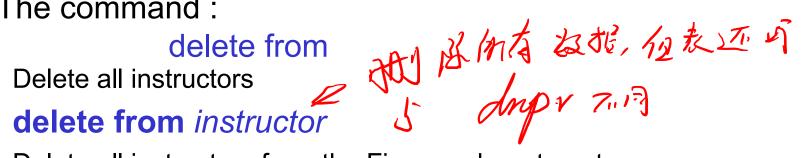


# View Expansion

- One view may be used in the expression defining another view. A way to define the meaning of views defined in terms of other views.
- Let view  $v_1$  be defined by an expression  $e_1$  that may itself contain uses of view relations.
- View expansion of an expression repeats the following replacement step:
  - repeat
    - Find any view relation  $v_i$  in  $e_1$
    - Replace the view relation  $v_i$  by the expression defining  $v_i$
- until no more view relations are present in  $e_1$ 
  - ■As long as the view definitions are not recursive, this loop
- will terminate

# Modification of the Database -Deletion

- The command :



- Delete all instructors from the Finance department delete from instructor where dept name= 'Finance';
- Delete all tuples in the *instructor* relation for those instructors associated with a department located in the Watson building.
  - delete from instructor

where dept name in (select dept name from department **where** *building* = 'Watson');



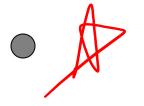
Delete all instructors whose salary is less than the average salary of instructors

- Problem: as we delete tuples from deposit, the average salary changes
- Solution used in SQL:
  - 1. First, compute avg (salary) and find all tuples to delete
    - 2. Next, delete all tuples found above (without recomputing avg or retesting the tuples)

# Modification of the Database -Insertion

- The command :
  - Insert into Values()
- Add a new tuple to course
- insert into course 不能成為中人。公和民性是和表中属外生和 values ('CS-437', 'Database Systems', 'Comp. Sci.',
  - 4);
  - insert into course (course\_id, title, dept\_name, credits) recourse values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
    - Add a new tuple to *student* with *tot creds* set to null
- insert into student values ('3003', 'Green', 'Finance', nul

■ Add all instructors to the *student* relation with tot\_creds set to 0



insert into student (ID, Name, dept. Nove, &£\_creds)
select ID, name, dept\_name, 0
from instructor

The select from where statement is evaluated fully before any of its results are inserted into the relation.

Otherwise queries like

insert into table1 select \* from table1

would cause problem



# Modification of the Database -Update

The command:

Update Set

Increase salaries of instructors whose salary is over \$100,000 by 3%, and all others by a 5%

```
update instructor
set salary = salary * 1.03
where salary > 100000;
update instructor
set salary = salary * 1.05
where salary <= 100000;
```

- The order is important
- Can be done better using the case statement



# Case Statement for Conditional Updates System Concepts

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</pre>
```



# Updates with Scalar Subqueries DataBase System Concepts

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
- Instead of **sum**(*credits*), use:

```
case
  when sum(credits) is not null then sum(credits)
  else 0
end
```

CREATE INDEX H\_INDEX ON STUDENT(HEIGHT)

CREATE UNIQUE INDEX SC\_INDEX ON

SC(SNO ASC,CNO DESC)

**DROP INDEX H INDEX** 



#### **SUMMARY**

Query

SELECT..

Modification

Delete, Insert, update

Definition

Table, View

