



Chapter 3: Introduction to SQL

Database System Concepts, 6th Ed.

©Silberschatz, Korth and Sudarshan

See www.db-book.com for conditions on re-use



Chapter 3: Introduction to SQL

- Overview of the SQL Query Language
- Data Definition
- Basic Query Structure
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database



History

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
 - SQL-86, SQL-89, SQL-92
 - SQL:1999, SQL:2003, SQL:2008
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
 - Not all examples here may work on your particular system.



Data Definition Language

The SQL **data-definition language (DDL)** allows the specification of information about relations, including:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints
- And as we will see later, also other information such as
 - 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.



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.
- More are covered in Chapter 4.



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-constraint1),  
                ...,  
                (integrity-constraintk))
```

- 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),  
    name        varchar(20) not null,  
    dept_name   varchar(20),  
    salary     numeric(8,2))
```

- **insert into** *instructor* **values** ('10211', 'Smith', 'Biology', 66000);
- **insert into** *instructor* **values** ('10211', null, 'Biology', 66000);



Integrity Constraints in Create Table

- **not null**
- **primary key** (A_1, \dots, A_n)
- **foreign key** (A_m, \dots, A_n) **references** r

Example: Declare *dept_name* as the primary key for *department*

.

```
create table instructor (  
    ID          char(5),  
    name       varchar(20) not null,  
    dept_name varchar(20),  
    salary     numeric(8,2),  
    primary key (ID),  
    foreign key (dept_name) references department)
```

primary key declaration on an attribute automatically ensures **not null**



And a Few More Relation Definitions

- **create table** *student* (
 ID **varchar**(5),
 name **varchar**(20) not null,
 dept_name **varchar**(20),
 tot_cred **numeric**(3,0),
 primary key (*ID*),
 foreign key (*dept_name*) **references** *department*));
- **create table** *takes* (
 ID **varchar**(5),
 course_id **varchar**(8),
 sec_id **varchar**(8),
 semester **varchar**(6),
 year **numeric**(4,0),
 grade **varchar**(2),
 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 course in the same semester



And more still

- **create table** *course* (
 course_id **varchar(8) primary key**,
 title **varchar(50)**,
 dept_name **varchar(20)**,
 credits **numeric(2,0)**,
 foreign key (*dept_name*) **references** *department*));
- Primary key declaration can be combined with attribute declaration as shown above



Drop and Alter Table Constructs

- **drop table** *student*
 - Deletes the table and its contents
- **delete from** *student*
 - Deletes all contents of table, but retains table
- **alter table**
 - **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*.
 - ▶ All tuples in the relation are assigned *null* as the value for the new attribute.
 - **alter table** *r* **drop** *A*
 - ▶ where *A* is the name of an attribute of relation *r*
 - ▶ Dropping of attributes not supported by many databases



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.
- 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.)
 - E.g. *Name* \equiv *NAME* \equiv *name*
 - Some people use upper case wherever we use bold font.



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
```



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.



The where Clause

- The **where** clause specifies conditions that the result must satisfy
 - Corresponds to the selection predicate of the relational algebra.
- 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.



The from Clause

- The **from** clause lists the relations involved in the query
 - Corresponds to the Cartesian product operation of the relational algebra.

- 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 (selection operation in relational algebra)



Cartesian Product: *instructor X teaches*

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 |

teaches

| 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 |

| 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 | Finance | 90000 | 10101 | CS-347 | 1 | Fall | 2009 |
| 12121 | Wu | Finance | 90000 | 12121 | FIN-201 | 1 | Spring | 2010 |
| 12121 | Wu | Finance | 90000 | 15151 | MU-199 | 1 | Spring | 2010 |
| 12121 | Wu | Finance | 90000 | 22222 | PHY-101 | 1 | Fall | 2009 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |



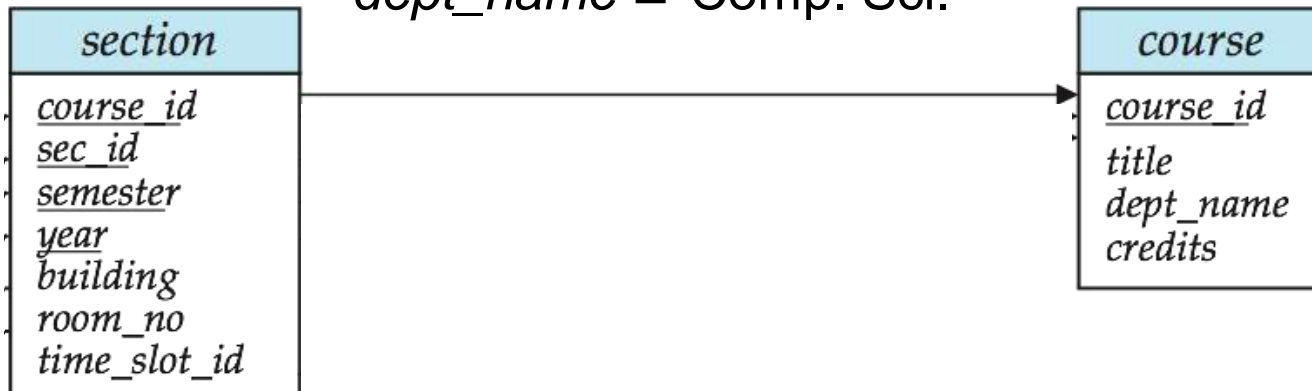
Joins

- 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
```

- 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.'
```





Try Writing Some Queries in SQL

- Suggest queries to be written.....



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*;

| ID | name | dept_name | salary | course_id | sec_id | semester | year |
|-------|------------|------------|--------|-----------|--------|----------|------|
| 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;*
 - **select** *name, course_id*
from *instructor* **natural join** *teaches;*



Natural Join (Cont.)

- Danger in natural join: beware of unrelated attributes with same name which get equated incorrectly
- List the names of instructors along with the the titles of courses that they teach
 - Incorrect version (makes `course.dept_name = instructor.dept_name`)
 - ▶ **select** *name, title*
from *instructor* **natural join** *teaches* **natural join** *course*;
 - Correct version
 - ▶ **select** *name, title*
from *instructor* **natural join** *teaches, course*
where *teaches.course_id = course.course_id*;
 - Another correct version
 - ▶ **select** *name, title*
from (*instructor* **natural join** *teaches*)
join *course* **using**(*course_id*);



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*

- 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*

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



String Operations

- SQL includes a string-matching operator for comparisons on character strings. The operator “like” uses patterns that are described using two special characters:
 - 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%'
```

- Match the string “100 %”

```
like '100 \%' escape '\'
```




String Operations (Cont.)

- 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.
- 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*



Where Clause Predicates

- SQL includes a **between** comparison operator
- Example: Find the names of all instructors with salary between \$90,000 and \$100,000 (that is, $\geq \$90,000$ and $\leq \$100,000$)
 - **select** *name*
from *instructor*
where *salary* **between** 90000 **and** 100000
- Tuple comparison
 - **select** *name, course_id*
from *instructor, teaches*
where (*instructor.ID, dept_name*) = (*teaches.ID, 'Biology'*);



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

Suppose a tuple occurs m times in r and n times in s , then, it occurs:

- $m + n$ times in r **union all** s
- $\min(m, n)$ times in r **intersect all** s
- $\max(0, m - n)$ times in r **except all** s



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: $5 + \text{null}$ 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 and Three Valued Logic

- Any comparison with *null* returns *unknown*
 - Example: $5 < null$ or $null <> null$ or $null = null$
- Three-valued logic using the truth value *unknown*:
 - OR: $(unknown \text{ or } true) = true$,
 $(unknown \text{ or } false) = unknown$
 $(unknown \text{ or } unknown) = unknown$
 - AND: $(true \text{ and } unknown) = unknown$,
 $(false \text{ and } unknown) = false$,
 $(unknown \text{ and } unknown) = unknown$
 - NOT: $(\text{not } unknown) = unknown$
 - “*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*



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 (Cont.)

- 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*)
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.
- A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality.



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);
```



Example Query

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

- **Note:** Above query can be written in a much simpler manner. The formulation above is simply to illustrate SQL features.



Set Comparison

- 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

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

Where <comp> can be: <, ≤, >, =, ≠

$(5 < \text{some } \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{true}$ (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, $(\neq \text{some}) \not\equiv \text{not in}$



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 = 'Biology');
```



Definition of all Clause

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

$$(5 < \text{all } \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{false}$$

$$(5 < \text{all } \begin{array}{|c|} \hline 6 \\ \hline 10 \\ \hline \end{array}) = \text{true}$$

$$(5 = \text{all } \begin{array}{|c|} \hline 4 \\ \hline 5 \\ \hline \end{array}) = \text{false}$$

$$(5 \neq \text{all } \begin{array}{|c|} \hline 4 \\ \hline 6 \\ \hline \end{array}) = \text{true (since } 5 \neq 4 \text{ and } 5 \neq 6)$$

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

However, $(= \text{all}) \not\equiv \text{in}$



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** $r \Leftrightarrow r = \emptyset$



Correlation Variables

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

- **Correlated subquery**
- **Correlation name** or **correlation variable**



Not Exists

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

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

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



Test for Absence of Duplicate Tuples

- The **unique** construct tests whether a subquery has any duplicate tuples in its result.
 - (Evaluates to “true” on an empty set)
- Find all courses that were offered at most once in 2009

```
select T.course_id
from course as T
where unique (select R.course_id
                  from section as R
                  where T.course_id= R.course_id
                      and R.year = 2009);
```




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

```
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;
```



Subqueries in the From Clause (Cont.)

- And yet another way to write it: **lateral** clause

```
select name, salary, avg_salary
from instructor I1,
        lateral (select avg(salary) as avg_salary
                from instructor I2
                where I2.dept_name= I1.dept_name);
```

- Lateral clause permits later part of the **from** clause (after the lateral keyword) to access correlation variables from the earlier part.
- Note: lateral is part of the SQL standard, but is not supported on many database systems; some databases such as SQL Server offer alternative syntax



With Clause

- The **with** clause provides a way of defining a temporary view 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 budget  
from department, max_budget  
where department.budget = max_budget.value;
```



Complex Queries using With Clause

- With clause is very useful for writing complex queries
- Supported by most database systems, with minor syntax variations
- 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;
```



Scalar Subquery

- Scalar subquery is one which is used where a single value is expected
- E.g.

```
select dept_name,  
      (select count(*)  
       from instructor  
       where department.dept_name = instructor.dept_name)  
      as num_instructors  
from department;
```
- E.g.

```
select name  
from instructor  
where salary * 10 >  
      (select budget from department  
       where department.dept_name = instructor.dept_name)
```
- Runtime error if subquery returns more than one result tuple



Modification of the Database

- Deletion of tuples from a given relation
- Insertion of new tuples into a given relation
- Updating values in some tuples in a given relation



Modification of the Database – Deletion

- Delete all instructors

delete from *instructor*

- 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');



Deletion (Cont.)

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

delete from *instructor*
where *salary* < (**select avg** (*salary*) **from** *instructor*);

- 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

- Add a new tuple to *course*

insert into *course*

values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);

- or equivalently

insert into *course* (*course_id*, *title*, *dept_name*, *credits*)

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', *null*);



Insertion (Cont.)

- Add all instructors to the *student* relation with *tot_creds* set to 0

insert into *student*

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 problems, if *table1* did not have any primary key defined.



Modification of the Database – Updates

- Increase salaries of instructors whose salary is over \$100,000 by 3%, and all others receive a 5% raise
 - Write two **update** statements:

```
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 (next slide)



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* **natural join** *course*
 where *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



End of Chapter 3

Database System Concepts, 6th Ed.

©Silberschatz, Korth and Sudarshan

See www.db-book.com for conditions on re-use



Advanced SQL Features**

- Create a table with the same schema as an existing table:
create table *temp_account* **like** *account*



Figure 3.02

| <i>name</i> |
|-------------|
| Srinivasan |
| Wu |
| Mozart |
| Einstein |
| El Said |
| Gold |
| Katz |
| Califieri |
| Singh |
| Crick |
| Brandt |
| Kim |



Figure 3.03

| <i>dept_name</i> |
|------------------|
| Comp. Sci. |
| Finance |
| Music |
| Physics |
| History |
| Physics |
| Comp. Sci. |
| History |
| Finance |
| Biology |
| Comp. Sci. |
| Elec. Eng. |



Figure 3.04

| <i>name</i> |
|-------------|
| Katz |
| Brandt |



Figure 3.05

| <i>name</i> | <i>dept_name</i> | <i>building</i> |
|-------------|------------------|-----------------|
| Srinivasan | Comp. Sci. | Taylor |
| Wu | Finance | Painter |
| Mozart | Music | Packard |
| Einstein | Physics | Watson |
| El Said | History | Painter |
| Gold | Physics | Watson |
| Katz | Comp. Sci. | Taylor |
| Califieri | History | Painter |
| Singh | Finance | Painter |
| Crick | Biology | Watson |
| Brandt | Comp. Sci. | Taylor |
| Kim | Elec. Eng. | Taylor |



Figure 3.07

| <i>name</i> | <i>Course_id</i> |
|-------------|------------------|
| Srinivasan | CS-101 |
| Srinivasan | CS-315 |
| Srinivasan | CS-347 |
| Wu | FIN-201 |
| Mozart | MU-199 |
| Einstein | PHY-101 |
| El Said | HIS-351 |
| Katz | CS-101 |
| Katz | CS-319 |
| Crick | BIO-101 |
| Crick | BIO-301 |
| Brandt | CS-190 |
| Brandt | CS-190 |
| Brandt | CS-319 |
| Kim | EE-181 |



Figure 3.08

| <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 |
| 83821 | Brandt | Comp. Sci. | 92000 | CS-190 | 1 | Spring | 2009 |
| 83821 | Brandt | Comp. Sci. | 92000 | CS-190 | 2 | Spring | 2009 |
| 83821 | Brandt | Comp. Sci. | 92000 | CS-319 | 2 | Spring | 2010 |
| 98345 | Kim | Elec. Eng. | 80000 | EE-181 | 1 | Spring | 2009 |



Figure 3.09

| <i>course_id</i> |
|------------------|
| CS-101 |
| CS-347 |
| PHY-101 |



Figure 3.10

| <i>course_id</i> |
|------------------|
| CS-101 |
| CS-315 |
| CS-319 |
| CS-319 |
| FIN-201 |
| HIS-351 |
| MU-199 |



Figure 3.11

| <i>course_id</i> |
|------------------|
| CS-101 |
| CS-315 |
| CS-319 |
| CS-347 |
| FIN-201 |
| HIS-351 |
| MU-199 |
| PHY-101 |



Figure 3.12

| |
|------------------|
| <i>course_id</i> |
| CS-101 |



Figure 3.13

| <i>course_id</i> |
|------------------|
| CS-347 |
| PHY-101 |



Figure 3.16

| <i>dept_name</i> | <i>count</i> |
|------------------|--------------|
| Comp. Sci. | 3 |
| Finance | 1 |
| History | 1 |
| Music | 1 |



Figure 3.17

| <i>dept_name</i> | <i>avg(salary)</i> |
|------------------|--------------------|
| Physics | 91000 |
| Elec. Eng. | 80000 |
| Finance | 85000 |
| Comp. Sci. | 77333 |
| Biology | 72000 |
| History | 61000 |