



# Chapter 3: Introduction to SQL

**Database System Concepts, 6<sup>th</sup> Ed.**

©Silberschatz, Korth and Sudarshan

See [www.db-book.com](http://www.db-book.com) for conditions on re-use



# Database System Concepts

- Chapter 1: Introduction
- **Part 1: Relational databases**
  - Chapter 2: Introduction to the Relational Model
  - Chapter 3: Introduction to SQL
  - Chapter 4: Intermediate SQL
  - Chapter 5: Advanced SQL
  - Chapter 6: Formal Relational Query Languages
- **Part 2: Database Design**
  - Chapter 7: Database Design: The E-R Approach
  - Chapter 8: Relational Database Design
  - Chapter 9: Application Design
- **Part 3: Data storage and querying**
  - Chapter 10: Storage and File Structure
  - Chapter 11: Indexing and Hashing
  - Chapter 12: Query Processing
  - Chapter 13: Query Optimization
- **Part 4: Transaction management**
  - Chapter 14: Transactions
  - Chapter 15: Concurrency control
  - Chapter 16: Recovery System
- **Part 5: System Architecture**
  - Chapter 17: Database System Architectures
  - Chapter 18: Parallel Databases
  - Chapter 19: Distributed Databases
- **Part 6: Data Warehousing, Mining, and IR**
  - Chapter 20: Data Mining
  - Chapter 21: Information Retrieval
- **Part 7: Specialty Databases**
  - Chapter 22: Object-Based Databases
  - Chapter 23: XML
- **Part 8: Advanced Topics**
  - Chapter 24: Advanced Application Development
  - Chapter 25: Advanced Data Types
  - Chapter 26: Advanced Transaction Processing
- **Part 9: Case studies**
  - Chapter 27: PostgreSQL
  - Chapter 28: Oracle
  - Chapter 29: IBM DB2 Universal Database
  - Chapter 30: Microsoft SQL Server
- **Online Appendices**
  - Appendix A: Detailed University Schema
  - Appendix B: Advanced Relational Database Model
  - Appendix C: Other Relational Query Languages
  - Appendix D: Network Model
  - Appendix E: Hierarchical Model



# Chapter 3: Introduction to SQL

- 3.1 Overview of the SQL Query Language
- 3.2 SQL Data Definition
- 3.3 Basic Structure of SQL Queries
- 3.4 Additional Basic Operations
- 3.5 Set Operations
- 3.6 Null Values
- 3.7 Aggregate Functions
- 3.8 Nested Subqueries
- 3.9 Modification of the Database



# History

- **IBM Sequel language** developed as part of **System R project** at the IBM San Jose Research Laboratory (1974)
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
  - SQL-86
  - SQL-89
  - SQL-92
  - SQL:1999 (language name became Y2K compliant!)
  - SQL:2003
- **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.



# Chapter 3: Introduction to SQL

- 3.1 Overview of the SQL Query Language
- 3.2 SQL Data Definition
- 3.3 Basic Structure of SQL Queries
- 3.4 Additional Basic Operations
- 3.5 Set Operations
- 3.6 Null Values
- 3.7 Aggregate Functions
- 3.8 Nested Subqueries
- 3.9 Modification of the Database



# 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.

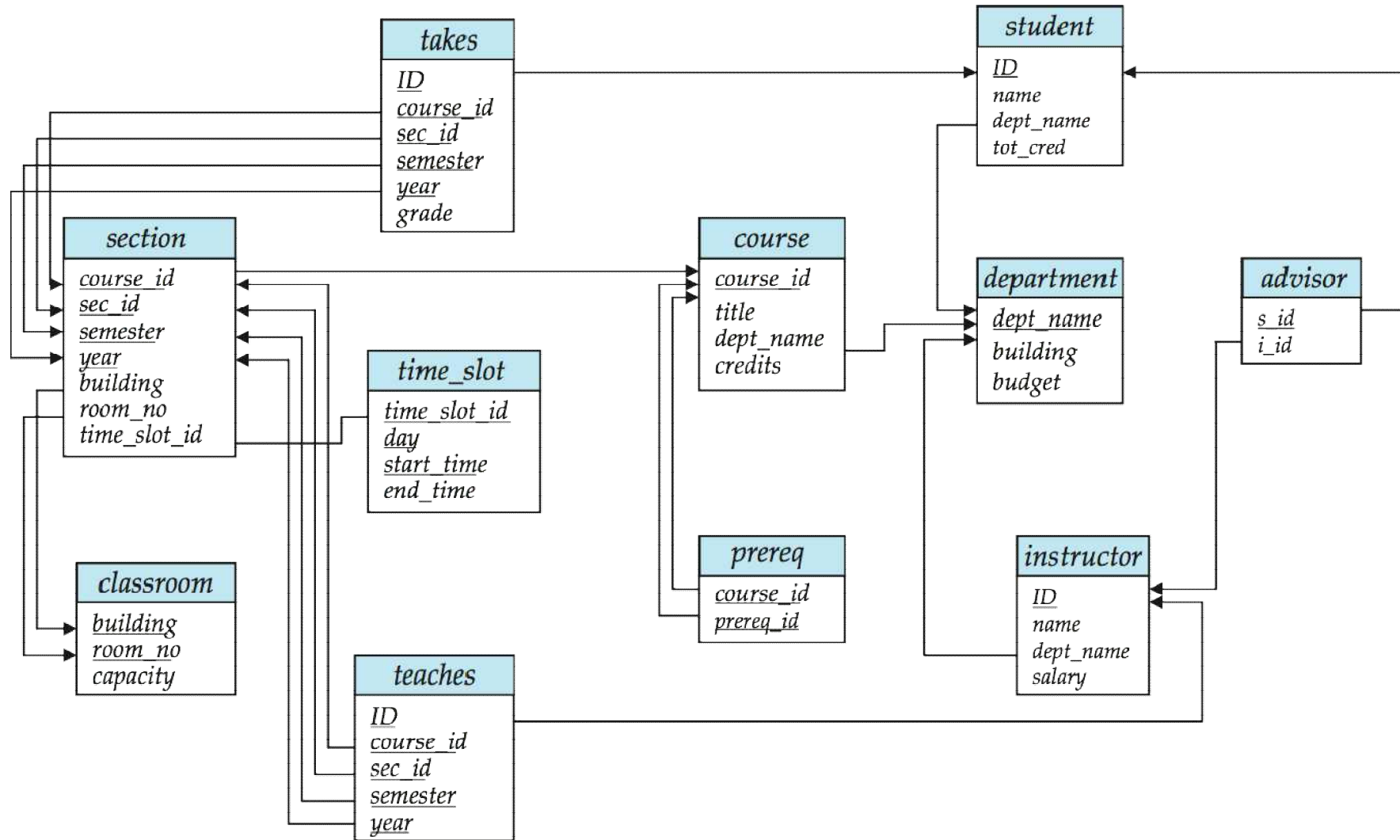


# Domain Types in SQL

- **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.



# Schema Diagram for University Database







# Create Table Construct

- An SQL relation is defined using the **create table** command:

```
create table  $r$  ( $A_1 D_1, A_2 D_2, \dots, A_n D_n,$   
                (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),  
    dept_name varchar(20),  
    salary     numeric(8,2))
```



# 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  $ID$  as the primary key for *instructor*

```
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**

```
■ 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 );
```



## And a Few More Relation Definitions

- **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
- **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* );



# Updates to tables

## ■ Insert a tuple

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

## ■ Delete tuples

- delete from *student* where dept\_name = 'Biology'
- delete from *r*

## ■ Drop Table

- drop table *r*

## ■ 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.



# Chapter 3: Introduction to SQL

- 3.1 Overview of the SQL Query Language
- 3.2 SQL Data Definition
- 3.3 Basic Structure of SQL Queries
- 3.4 Additional Basic Operations
- 3.5 Set Operations
- 3.6 Null Values
- 3.7 Aggregate Functions
- 3.8 Nested Subqueries
- 3.9 Modification of the Database



# Basic Query Structure

- 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 department names of all instructors, 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*

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

*Instructor X teaches*

<u>Inst.ID</u>	name	dept_name	salary	<u>teaches.ID</u>	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 [1]

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

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

*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



## Joins[2]

- 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.'
```

**Figure 2.02: The Course relation**

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

**Figure 2.06: The Section relation**

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



# Try Writing Some Queries in SQL

- Try writing some queries..



# Chapter 3: Introduction to SQL

- 3.1 Overview of the SQL Query Language
- 3.2 SQL Data Definition
- 3.3 Basic Structure of SQL Queries
- 3.4 Additional Basic Operations
- 3.5 Set Operations
- 3.6 Null Values
- 3.7 Aggregate Functions
- 3.8 Nested Subqueries
- 3.9 Modification of the Database





# 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;**

*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

*Instructor* ⋈ *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
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



# 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;*

*instructor*

<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

*teaches*

<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



# 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* → *instructor T*



# 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 '\'
```

in that above we use backslash (\) as the escape character.



## 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 the names of all instructors in alphabetic order

```
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'*) ;

*instructor*

<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

*teaches*

<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



# 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.  $\Pi_A(r_1)$ : For each copy of tuple  $t_1$  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  $t_1$ .
  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  $t_1 \cdot t_2$  in  $r_1 \times r_2$ .





## Duplicates (Cont.)

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

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

- 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**  $A_1, A_2, \dots, A_n$   
**from**  $r_1, r_2, \dots, r_m$   
**where**  $P$

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

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



**Figure 2.02: The Course relation**

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

**Figure 2.05: The Department Relation**

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

**Figure 2.03: The Pre-requisite relation**

<i>course_id</i>	<i>prereq_id</i>
BIO-301	BIO-101
BIO-399	BIO-101
CS-190	CS-101
CS-315	CS-101
CS-319	CS-101
CS-347	CS-101
EE-181	PHY-101

**Figure 2.04: The Instructor relation**

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



**Figure 2.06: The Section relation**

<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

**Figure 2.07: The Teaches relation**

<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
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009



**Figure 3.02: select  
name from instructor**

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

**Figure 3.03: select dept\_name  
from instructor**

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

**Figure 3.04: Find the names of all  
instructors in the computer  
science department who have  
salary greater than \$70,000**

<i>name</i>
Katz
Brandt

**Figure 3.05: Retrieve the names of all  
instructors, along with their department  
names and department building name**

<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 3.06: Cartesian Product of the instructor and the teaches relations**

<i>Inst.ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>	<i>teaches.ID</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>
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
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2009
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2010
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2009
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2010
15151	Mozart	Music	40000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2009
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2010
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2009
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2010
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...





**Figure 3.07: For all instructors in the university who have taught some course, find their names and the course ID of all courses they taught**

<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: Natural Join of the instructor and the teaches relations**

<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



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

**Figure A.9** The *student* relation.

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

**Figure A.10** The *takes* relation.



# Chapter 3: Introduction to SQL

- 3.1 Overview of the SQL Query Language
- 3.2 SQL Data Definition
- 3.3 Basic Structure of SQL Queries
- 3.4 Additional Basic Operations
- 3.5 Set Operations
- 3.6 Null Values
- 3.7 Aggregate Functions
- 3.8 Nested Subqueries
- 3.9 Modification of the Database





# 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)

## section

<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



# 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**  $s$



**Figure 3.09: The C1 relation, listing courses taught in Fall 2009**

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

**Figure 3.10: The c2 relation, listing courses taught in Spring 2010**

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

**Figure 3.11: c1 UNION c2**

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

**Figure 3.12: c1 INTERSECTION c2**

<i>course_id</i>
CS-101

**Figure 3.13: c1 EXCEPT c2**

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

Consider!!!  
C1 union all C2  
C1 intersect all C2  
C1 except all C2



# Chapter 3: Introduction to SQL

- 3.1 Overview of the SQL Query Language
- 3.2 SQL Data Definition
- 3.3 Basic Structure of SQL Queries
- 3.4 Additional Basic Operations
- 3.5 Set Operations
- 3.6 Null Values
- 3.7 Aggregate Functions
- 3.8 Nested Subqueries
- 3.9 Modification of the Database



# 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*

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



# Null Values and Three Valued Logic

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



# Chapter 3: Introduction to SQL

- 3.1 Overview of the SQL Query Language
- 3.2 SQL Data Definition
- 3.3 Basic Structure of SQL Queries
- 3.4 Additional Basic Operations
- 3.5 Set Operations
- 3.6 Null Values
- 3.7 Aggregate Functions
- 3.8 Nested Subqueries
- 3.9 Modification of the Database



# 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

- 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;**

**Figure 3.16**

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





# 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*;

**Figure 3.14: Instructors grouped by the dept\_name attribute**

<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

**Figure 3.15: Find the average salary in each department**

<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;
```

- 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;
```

**Figure 3.17**

dept_name	avg(salary)
Physics	91000
Elec. Eng.	80000
Finance	85000
Comp. Sci.	77333
Biology	72000
History	61000

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



# Chapter 3: Introduction to SQL

- 3.1 Overview of the SQL Query Language
- 3.2 SQL Data Definition
- 3.3 Basic Structure of SQL Queries
- 3.4 Additional Basic Operations
- 3.5 Set Operations
- 3.6 Null Values
- 3.7 Aggregate Functions
- 3.8 Nested Subqueries
- 3.9 Modification of the Database



# 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.

- 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.

**takes**

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

**teaches**

<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



# 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

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

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

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

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

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

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

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





## Example Query with > all

- 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

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

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

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

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

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

$(\neq \text{all}) \equiv \text{not in}$  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**  $r \Leftrightarrow r = \emptyset$

- 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**



## Example Query with 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));
```

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

■ 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.
- The **unique** construct 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 Form 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 Form 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);
```

- 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



## Example Query with “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 budget  
from department, max_budget  
where department.budget = max_budget.value;
```





# Complex Queries using With Clause

- Find all departments where the total salary is greater than the average of the total salary at all departments

```
with dept_total (dept_name, value) as  
    (select dept_name, sum(salary)  
     from instructor  
     group by dept_name),  
dept_total_avg(value) 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**

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

- Runtime error if subquery returns more than one result tuple



# Chapter 3: Introduction to SQL

- 3.1 Overview of the SQL Query Language
- 3.2 SQL Data Definition
- 3.3 Basic Structure of SQL Queries
- 3.4 Additional Basic Operations
- 3.5 Set Operations
- 3.6 Null Values
- 3.7 Aggregate Functions
- 3.8 Nested Subqueries
- 3.9 Modification of the Database



# Modification of the Database

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



# 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');

## instructor

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000



## 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)



# 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*);

course			
<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4

student			
<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>tot_cred</i>
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110



## 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 problem





# 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

- 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**

**takes**

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

**courses**

course_id	title	dept_name	credits
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3



# End of Chapter 3

**Database System Concepts, 6<sup>th</sup> Ed.**

©Silberschatz, Korth and Sudarshan

See [www.db-book.com](http://www.db-book.com) for conditions on re-use