



# Chapter 3: Introduction to SQL

Database System Concepts, 7<sup>th</sup> Ed.

©Silberschatz, Korth and Sudarshan

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



# Outline

- Overview of The SQL Query Language
- SQL Data Definition
- Basic Query Structure of SQL Queries
- 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 (language name became Y2K compliant!)
  - SQL:2003
  - SQL:2006,
  - SQL:2008,
  - SQL:2011,
  - and most recently SQL:2016.
- 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.



# SQL Parts

- DML -- provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.
- integrity – the DDL includes commands for specifying integrity constraints.
- View definition -- The DDL includes commands for defining views.
- Transaction control –includes commands for specifying the beginning and ending of transactions.
- Embedded SQL and dynamic SQL -- define how SQL statements can be embedded within general-purpose programming languages.
- Authorization – includes commands for specifying access rights to relations and views.

برای تعریف ساختار دیتا بیس: DML  
برای خود داده: DDI

-----  
برای کار روی نمونه های جدوله DML  
برای کار روی جدول و ایجاد جدول DDL



# Data Definition Language

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

- The schema for each relation.
- The type of values associated with each attribute.
- The Integrity constraints
- The set of indices to be maintained for each relation.
- 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  $d$  digits to the right of decimal point. (ex., **numeric(3,1)**, allows 44.5 to be stored exactly, but not 444.5 or 0.32)
- **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.



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

Diagram annotations:

- Arrows point from the labels "attributes (or columns)" to the column headers *ID*, *name*, *dept\_name*, and *salary*.
- Arrows point from the label "tuples (or rows)" to the first four data rows of the table body.

باید تعریف بکنیم تایپ داده ای ستون ها چیه

کاراکتر برای ما یک فضای ثابتی رو در نظر میگیره مثل (char 10) بنی برای هر سطر 10 تا اشغال میشه  
مثل برای name اگر 10 تا باشه بنی کلا ما 10 تا داریم حالا چه همشو استفاده بکنیم چه مثل 4 تاشو در  
هر صورت ما 10 تا جا برای فضا داریم ولی varchar ما ماکزیمم رو مشخص میکنیم



# 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-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),  
    name        varchar(20),  
    dept_name   varchar(20),  
    salary      numeric(8,2))
```



- تعریف table:

نام ستون های ما است  
Ai تایپ داده هامون است  
Di

می تونیم توی table یکسری محدودیت هایی هم داشته باشیم

نکته:

ترتیب اجرای عملیات ها

from

where

group by

having

select

order by



# Integrity Constraints in Create Table

- Types of integrity constraints
  - **primary key** ( $A_1, \dots, A_n$ )
  - **foreign key** ( $A_m, \dots, A_n$ ) **references**  $r$
  - **not null**
- SQL prevents any update to the database that violates an integrity constraint.
- Example: توى اين مثال ما محدوديت داريم

```
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);
```







# And a Few More Relation Definitions

بنی برای اون ستون حق نداریم **null** بذاریم بنی همه فیلد هاش باید پر بشه

- **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);**



# And more still

- **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

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

## ▪ Delete

داده های جدول رو حذف میکنه ینی کلا جدول خالی میشه ینی همه تاپل هارو حذف میکنه ینی نام ستون ها می مونه ولی جدول خالیه

- Remove all tuples from the *student* relation
  - **delete from *student***

## ▪ Drop Table

- **drop table *r***

## ▪ Alter

با این یه ستون اضافه یا کم میکنه



- **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 existing 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.

ما **table** مون ایجاد شده می خوایم داده اضافه بکنیم پس از **insert** استفاده میکنیم  
بنی کلا **table r** **drop table r** را حذفش بکن  
**alter**  
بیا به یک جدولی ستونی اضافه بکن  
A میشه نام ویژگی  
D میشه نوع تایپمون



# Basic Query Structure

- A typical SQL query has the form:

```
select A1, A2, ..., An  
from R1, R2, ..., Rm  
where P
```

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

-  
کن این ستون هارو از این جدول ها و توی where هم یه سری شرط می گه  
خروجی هم یک رابطه است اینجا



# The select Clause

- The **select** clause lists 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.,  $Name \equiv NAME \equiv name$

- نکته: SQL به حروف بزرگ و کوچک حساس نیست --> اسم ستون یا جدول  
رشته اینطوری نیست یعنی به حروف کوچک و بزرگ حساسن --> ولیو حساس است به حروف



# 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 should not be removed.

```
select all dept_name  
from instructor
```

<i>??</i>	<i>dept_na</i>
vasan	Comp.
	Finance
art	Music
tein	Physics
uid	History
l	Physics
	Comp.
ieri	History
h	Finance
k	Biology
dt	Comp.
	Elec. E

: distinct

ینی متمایز کردن

براساس این که چه چیزی بعد از distinct نوشتیم میاد کنترل میکنه داده هایی رو توی سلکت به ما بر میگردونه که تکراری هاش حذف شده باشه ینی تاپل های تکراری رو حذف میکنه نکته: خود پایگاه داده رکوردهای تکراری رو بهمون نشون میده پس در حالت کلی ینی به صورت all است



# The select Clause (Cont.)

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

```
select *
  from instructor
```

- An attribute can be a literal with no **from** clause

```
select '437'
```

- Results is a table with one column and a single row with value “437”
- Can give the column a name using:

```
select '437' as FOO
```

- An attribute can be a literal with **from** clause

```
select 'A'
  from instructor
```

- Result is a table with one column and  $N$  rows (number of tuples in the *instructors* table), each row with value “A”

استار همه ستون ها رو نشون میده

'select '437' يني 437 به ما توي خروجي ميده يني يه دونه خونه داريم بدون هيچ عنوان ستوني select '437' as FOO يني يه دونه خونه داريم با عنوان ستون FOO

'select 'A' اينجا خروجي به تعداد رديف جدول هاي instructor است يني به اندازه تعداد تاپل هاي جدولمون A رو چاپ ميکنه يني اگر سه تا تاپل داشته باشيم سه بار A رو واسمون چاپ ميکنه

	(No column name)
1	437
2	437
3	437
4	437
5	437
6	437
7	437
8	437
9	437
10	437
11	437
12	437
13	437
14	437



# The select Clause (Cont.)

- 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
```

می تونیم عملیات های ریاضی هم روی ستون ها انجام بدیم ینی توی خروجی مقدار اولیه دیگه نیست مقدار تقسیم بر 12 میشه دیگه با as می تونیم یه اسم جدید برایش در نظر بگیریم مثلا برای همین سلری تقسیم بر 12 به اسم جدید به صورت گذاشته clause



# 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

```
select name  
from instructor  
where dept_name = 'Comp. Sci.'
```

- SQL allows the use of the logical connectives **and, or, and not**
- The operands of the logical connectives can be expressions involving the comparison operators **<, <=, >, >=, =, and <>**.
- Comparisons can be applied to results of arithmetic expressions
- To find all instructors in Comp. Sci. dept with salary > 70000

```
select name  
from instructor  
where dept_name = 'Comp. Sci.' and salary > 70000
```

<i>name</i>
Katz
Brandt

می خوایم یه سری شرط رو بررسی کنیم توی where روی سطرها اعمال میشه ینی روی ستونها داره بررسی میشه ولی سطرهارو فیلتر میکنه می تونیم از پرانتز استفاده بکنیم برای اولویت توی شرط ها ولی مهمه که پرانتزا رو درست بذاریم



# 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
```

این معادل ضرب کارتزین است

- generates every possible instructor – teaches pair, with all attributes from both relations.
- 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).

-  
استار به معنی همه ستون ها است



# Examples

اول ضرب کارتزین رخ میده و بعد شرط برسی میکنه

- 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'**

<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





# 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** اسم جدول بعد اسم ويُذكر

**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**

-  
این داره میگه استادایی که حداقل حقوقشون از یکی از استادای کامپیوتر ساینس بیشتر باشه



# 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%”

اینجا باید دقیقا عبارت 100% رو بده بهمون like '100 \%'

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

با like می تونیم توی رشته ها بگردیم و بررسی کنیم  
این توی where به کار میره

ینی رشته ای که داریم بررسی میکنیم حتی اگر شبیه هم بود باز بهمون داده بشه ولی اون رشته ای هم که میخوایم بهمون حتما داده بشه ینی داخلش باشه اونا علامت % --> این که سمت چپ باشه یا راست باشه مهمه ینی هر سمتی که % باشه هر چی بود اون سمت مهم نیست مثلا عبارتی میخوایم که تهش ali باشه پس میگیم %ali% ینی اولش مهم نیست چی باشه ولی اخرش حتما باید ali باشه ali% ینی اولش علی باشه

ینی علی هر جا بود برای ما برگردان -->%ali%  
اینجا به حروف کوچک و بزرگ حساس است چون value است علامت \_ ینی فقط یه کاراکتر

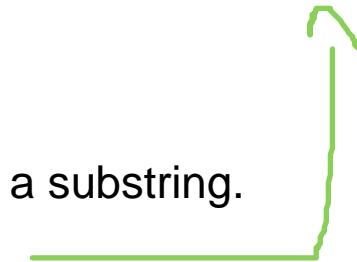
میخوایم عبارتمون 100% اولش به کار رفته باشه ینی عبارتمون 100% است اینجا یک کاراکتر به عنوان escape در نظر میگیریم که اون عبارت خاص رو نادیده بگیر و به عنوان کاراکتر بهش نگا کن اینجا escape ما \ است که میگه علامت % رو از جنس like نبین و کاراکتر در نظر بگیر نکته: اگر پشت like یک not بذاریم ینی بگیم not like در این حالت میگه اون رشته یا عبارت رو نداشته باشیم توی رشته ای که میده بهمون



# 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 “||”) ali || reza = ali + reza
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

سه تا کاراکتر باشه هر کاراکتری بود مهم نیست



حداقل سه تا کاراکتر باشه

\* می تونیم دو مقدار را بهم بچسبو نیم با || ینی می تونه رشته ها رو بهم بچسبو نه با + هم می تونیم این کارو بکنیم  
\* تبدیل حروف بزرگ به کوچک و بالعکس ینی می تونیم با upper or lower بیایم خاصیت بزرگ و کوچک رو از بین ببریم:

upper همه حروف توی اون رشته رو به حروف بزرگ تبدیل میکنه



# SQL Server

Function	Example	Result	Description
<b>CONCAT</b>	CONCAT('WWW','.com')	WWW.com	Adds two or more strings together
<b>LEN</b>	LEN('com')	3	Returns the length of a string
<b>LOWER</b>	LOWER('SQL!')	sql!	Converts a string to lower-case
<b>LTRIM</b>	LTRIM(' SQL !')	SQL !	Removes leading spaces from a string
<b>PATINDEX</b>	PATINDEX('%sc%','W3Sc.com')	3	Returns the position of a pattern in a string
<b>REPLACE</b>	REPLACE('TSQL','T','M')	MSQL	Replaces all occurrences of a substring within a string, with a new substring
<b>RTRIM</b>	RTRIM('SQL ')	SQL	Removes trailing spaces from a string
<b>SUBSTRING</b>	SUBSTRING('SQL',1,2)	SQ	Extracts some characters from a string
<b>TRIM</b>	TRIM(' SQL ')	SQL	Removes leading and trailing spaces (or other specified characters) from a string
<b>UPPER</b>	UPPER('sQl')	SQL	Converts a string to upper-case
<b>REPLICATE</b>	REPLICATE('SQ', 2)	SQSQ	Repeats a string a specified number of times
<b>RIGHT</b>	RIGHT('SQL',2)	QL	Extracts a number of characters from a string (starting from right)
<b>LEFT</b>	LEFT('SQL', 2)	SQ	Extracts a number of characters from a string (starting from left)

**TRIM** : اسپیس های اضافه رو حذف میکنه کلا برای تمیز کردن رشته به کار میره این این هم سمت چپ و راست رشته رو تمیز میکنه --> بهتره توی کار با رشته ها همیشه اینو داشته باشیم که چیزی خراب نشه

**LEN** : رشته رو به عنوان ورودی می گیره و طول رشته رو می ده به ما

**LTRM** : اگر اسپیس اضافی سمت چپ رشته باشه رو حذف میکنه

**PATINDEX** : جای پوزیشن یک پترن خاص توی رشتمون رو متوجه بشیم --> اینجا اندیس از یک شروع میشه مثلاینجا دنبال کاراکتر Sc هستیم که میشه کاراکتر سوم

**REPLACE** : میخوایم جای دوتا چیزو عوض کنیم

**RTRIM** : اگر اسپیس سمت راست بود رو حذف میکنه

**SUBSTRING** : یک رشته رو به عنوان ورودی می گیره و میگه فقط این تیکه هارو میخوام مثلا 1.3 (substring(1.3

ینی از اندیس یک شروع کن و سه تا از اون شروع کننده اندیس به ما بده

**REPLICATE** : یه عبارتی میدیم بهش و میگیم چند بار تکرارش بکن

**RIGHT** : یه بخشی از سمت راست یه عبارتو می خوایم برداریم

**LEFT** : یه بخشی از سمت چپ یه عبارتو می خوایم برداریم



# Oracle

Function	Example	Result	Purpose
CONCAT	CONCAT('A','BC')	'ABC'	Concatenate two strings and return the combined string
INSTR	INSTR( 'This is a playlist', 'is')	3	Search for a substring and return the location of the substring in a string
LENGTH	LENGTH('ABC')	3	Return the number of characters (or length) of a specified string
LOWER	LOWER('Abc')	'abc'	Return a string with all characters converted to lowercase
LPAD	LPAD('ABC',5,'*')	***ABC'	Return a string that is left-padded with the specified characters to a certain length.
LTRIM	LTRIM(' ABC ')	'ABC '	Remove spaces or other specified characters in a set from the left end of a string
REPLACE	REPLACE('JACK AND JOND','J','BL');	'BLACK AND BLOND'	Replace all occurrences of a substring by another substring in a string
RPAD	RPAD('ABC',5,'*')	'ABC**'	Return a string that is right-padded with the specified characters to a certain length.
RTRIM	RTRIM(' ABC ')	' ABC'	Remove all spaces or specified character in a set from the right end of a string
SUBSTR	SUBSTR('Oracle Substring', 1, 6 )	'Oracle'	Extract a substring from a string
TRIM	TRIM(' ABC ')	'ABC'	Remove the space character or other specified characters either from the start or end of a string
UPPER	UPPER('Abc')	'ABC'	Convert all characters in a specified string to uppercase

LPAD : با این طول رشته خروجی رو کنترل می کنه این میگه اگه طول رشته کمتر از 5 بود مثلا بیا سمت چپ رشته ستاره اضافه بکن



# 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**



نکته : order by می تونه براساس چندتا فیلد باشه ینی محدودیتی روی order by نداریم

نکته : اگر dept\_name برابر باشه بعد میره سراغ name و براساس name مرتب میکنه ولی اول براساس name مرتب میکنه در صورت برابری می ره سراغ dept\_name

نکته: توی دیتا بیس یک جدول یک مجموعه است و توی مجموعه ترتیب اهمیت نداره پس اگر بخوایم جدول به یه ترتیب خاص نمایش داده بشه باید از **order by** استفاده بکنیم

پس باید به **order by** بگیم بر چه اساسی مرتب کنه

نکته: اگ فقط بگیم **order by** براساس صعودی مرتب میکنه داده ها رو ولی اگه بخوایم نزولی باشه باید حتما اخرش بنویسیم **desc** که بشه نزولی



# 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');** 

این میاد اولی رو با اولی اون رو مقایسه میکنه ینی میاد `teaches.ID` رو با `instructor.ID` مقایسه میکنه و دومی هم به همین صورت به جای این کار میتوانستیم از `and instructor.ID=teaches.ID and dept_name='Biology'` استفاده بکنیم ینی بگیم

عبارت `between` که توی `where` به کار میره

ما میخوایم یک عبارتمون بین دوتا چیز مختلف باشه مثل حقوق اساتیدی رو بگو که حقوقشون بین 90000 و 100000 باشه

می تونستیم به جای `between` هم بگیم  $\text{salary} >= 90000 \text{ and } \text{salary} <= 100000$



# Set Operations

- Find courses that ran in Fall 2017 or in Spring 2018

```
(select course_id from section where sem = 'Fall' and year = 2017)
union
(select course_id from section where sem = 'Spring' and year = 2018)
```

- Find courses that ran in Fall 2017 and in Spring 2018

```
(select course_id from section where sem = 'Fall' and year = 2017)
intersect
(select course_id from section where sem = 'Spring' and year = 2018)
```

- Find courses that ran in Fall 2017 but not in Spring 2018

```
(select course_id from section where sem = 'Fall' and year = 2017)
except
(select course_id from section where sem = 'Spring' and year = 2018)
```

-

union : میاد اجتماع میگیره --> اینا باید از یک جنس و از یک تعداد باشند  
و union میاد تکراری ها رو حذف میکنه و همین طور دستور سنگینی است چون باید بره بگرده چی باهاش تکرار داره ینی یک رکورد رو باید با همه رکورد ها مقایسه بکنه تا اینو بفهمه

intersect : ینی اشتراک

except : ینی منها

نکته : union رو میاد با هیچی می زنه ینی چی ینی با شرطی که وجود نداره در این حالت تکراری نداریم؟؟؟



# Set Operations (Cont.)

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

اگر بعد از این سه تا به صورت عادی لغت **all** رو نداریم و اگر تکرار توى خروجى بیاد حذف میشه و رکوردهای غیر تکراری و اسمون نگه می داره ولی اگر **all** رو بذاریم چک نمیکنه که ایا اون رکورد تکرار شده یا نه پس کوئری ما با **all** سبک تر میشه چون خروجی اول رو بدست میاره و مستقل از اینکه داده هاش چی هستن میاد کنار هم می ذاره و به ما میده --> این رو می تونیم زمانی استفاده بکنیم که هیچ اشتراکی وجود نداشته باشه داخل کوئری ها یعنی هنر ما است استفاده از این دستور مثلا **all** واسه جایی کاربرد داره که ما لزوما هیچ اشتراکی نداریم و زمانی که از **all** استفاده می کنیم منطقا رکوردهای تکراری هم نداریم چون از قبل اشتراکی هم نداشتن



# 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
```

نام اساتیدی رو بده که حقوقشون **null** است

- The predicate **is not null** succeeds if the value on which it is applied is not null.

`null` ینی یه مقدار غیر مشخص ینی نه صفره و نه یه رشته تهی اگر برای رکوردی مقداری وارد نکنیم دیتا بیس برash `null` در نظر میگیره ما می تونیم دوتا ستون رو با هم جمع کنیم حالا اگر با یک `null` جمع کنیم بازم مقدارش `null` میشه نکته: بهتره بعضی وقتا این مقدار `null` رو مدیریت بکنیم با `is null` می تونیم بررسی بکنیم که یک مقداری `null` است یا نه با مقدار `isnull` که پیوسته است ینی بگیم:  
...select `isnull(salary,0)` sa form

این `isnull` اینجا به این معنی که اگر `salary` مقدار داره ک مقدار خودشو توی خروجی چاپ کن و اگر نداره به جاش صفر قرار بده --> یک مثالی زد که توی برگه نوشتم اون مثال ظرفیت انبار رو من نفهمیدم؟؟؟؟  
نکته: هر چیزی با `null` جمع بشه یا کم بشه یا از این موارد بازم به ما مقدار `null` برمی گردونه



# Null Values (Cont.)

- SQL treats as **unknown** the result of any comparison involving a null value (other than predicates **is null** and **is not null**).
  - Example: **5 < null** or **null <> null** or **null = null**
- The predicate in a **where** clause can involve Boolean operations (**and**, **or**, **not**); thus the definitions of the Boolean operations need to be extended to deal with the value **unknown**.
  - **and** :  $(\text{true and unknown}) = \text{unknown}$ ,  
 $(\text{false and unknown}) = \text{false}$ ,  
 $(\text{unknown and unknown}) = \text{unknown}$
  - **or**:  $(\text{unknown or true}) = \text{true}$ ,  
 $(\text{unknown or false}) = \text{unknown}$   
 $(\text{unknown or unknown}) = \text{unknown}$
- 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

بعضی وقتا نیاز داریم براساس یک یا چندتا فیچر داده هامون رو دسته بندی بکنیم توى group by تعداد رکورد ها تغییر میکنه

مثالا فرض می کنیم 6 تا موژیک داریم دوتا ابی و دوتا مشکی و دوتا سبز --> مثال این در برگه نوشته شده است  
ما اینجا مثلا 6 تا رکورد داریم و می گیم بیا رکورد های موژیک هایی که در اختیار داری رو براساس رنگ دسته بندی بکن که در این حالت ما 3 تا رکورد داریم دیگه یعنی اینجا برای هر رنگ یک رکورد در نظر می گیره اگر براساس رنگ و برنز گروه بندی کنیم به این صورت میشود که موژیک های ابی ما از هم جدا میشوند چون دوتا موژیک ابی ما برندهای متفاوتی داشتند پس در نهایت ما 4 تا رکورد داریم  
نکته: وقتی گروه بندی رو زیاد میکنیم یعنی بهش باز ستون اضافه میکنیم رکوردهای ما هم زیاد میشوند اگر گروه بندی براساس id بود در این حالت ما 6 تا رکورد داریم چون id یونیک است  
نکته: توى group by حداقل یک رکورد و حداقل مثلث توى این مثال 6 رکورد است



# Aggregate Functions Examples

- 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 2018 semester
  - **select count (distinct ID)  
from teaches  
where semester = 'Spring' and year = 2018;**
- Find the number of tuples in the *course* relation
  - **select count (\*)** این تعداد همه تاپل ها را به ما میده  
**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;`

<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

-  
توى اين مثال 7 تا رکورد داريم

```

select * from [Customer] t order by t.Branch_Cod desc;
-----  

SELECT t.Branch_Cod, count(*) cnt_all, count(salary) cnt_sal, avg(t.salary) sal_avg, sum(t.salary) sal_sum,  

       count(cod_personeli) cnt_per, avg(t.cod_personeli) per_avg, sum(t.cod_personeli) per_sum  

  FROM [Customer] t  

 group by t.Branch_Cod  

order by 1 desc;

```

براساس اولین ستون توی select میاد مرتب میکنه ینی براساس t.Branch\_code

نکته: sum مقدار null رو صفر در نظر میگیره ولی count مقدار null رو اصلا در نظر نمیگیره چون اگه صفر در نظر بگیره مجبوره اونو بشماره

Results

	customer_number	name	L_name	birth_date	city	National_cod	cod_personeli	Branch_Cod	salary
1	2	Saeed	mohammadi	2006-02-03	tehran	222222222	12	60	2000
2	16	ali	navidi	2000-04-16	shiraz	1111111111	64	32	4000
3	21	arash	mofidi	2046-08-08	tehran	1234567890	2	20	4000
4	20	ali	mohammadi	2000-04-16	shiraz	1111111111	NULL	20	4000
5	1	mohammad	mohammadi	2004-02-03	Isahan	2121212121	123	16	1000
6	18	ali	ahmadi	NULL	NULL	NULL	NULL	NULL	1000

	Branch_Cod	cnt_all	cnt_sal	sal_avg	sal_sum	cnt_per	per_avg	per_sum
1	60	1	1	2000	2000	1	12	12
2	32	1	1	4000	4000	1	64	64
3	20	2	2	4000	8000	1	2	2
4	16	1	1	1000	1000	1	123	123
5	NULL	1	1	1000	1000	0	NULL	NULL





# 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;**

---



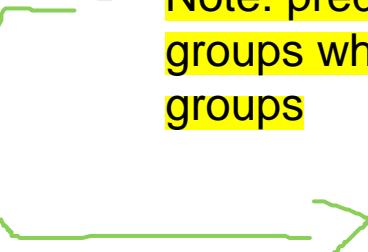
# Aggregate Functions – Having Clause

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

```
select dept_name, avg (salary) as 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

نکته: بعد از group by اجرا میشے ولی where قبلاش



- با این میتوانیم روی ستون هایی که توی سلکت group by است شرط بذاریم نکته: روی group by شرط می ذاره having



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

as follows:

- **From clause:**  $r_i$  can be replaced by any valid subquery
- **Where clause:**  $P$  can be replaced with an expression of the form:

$B <\text{operation}> (\text{subquery})$

$B$  is an attribute and  $<\text{operation}>$  to be defined later.

- **Select clause:**  
 $A_i$  can be replaced by a subquery that generates a single value.

می تونه توی select from or where or subqueries بیاد



# Set Membership





# Set Membership

- Find courses offered in Fall 2017 and in Spring 2018

```
select distinct course_id  
from section  
where semester = 'Fall' and year= 2017 and  
course_id in (select course_id  
from section  
where semester = 'Spring' and year= 2018);
```

- Find courses offered in Fall 2017 but not in Spring 2018

```
select distinct course_id  
from section  
where semester = 'Fall' and year= 2017 and  
course_id not in (select course_id  
from section  
where semester = 'Spring' and year= 2018);
```





# Set Membership (Cont.)

- Name all instructors whose name is neither “Mozart” nor Einstein”

```
select distinct name  
from instructor  
where name not in ('Mozart', 'Einstein')
```

- 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

بعد از `not in or in` می تونیم توی پرانتز یا مقدار های ثابت بنویسیم یا سلکت ولی ترکیب این دو تا رو نمیتوانیم  
داشته باشیم

نکته: کلید اصلی باید یکتا باشد و `null` هم نمیتواند باشد  
تعداد کل دانشآموزان (متمايز) که بخشهاي درسي تدریس شده توسط مربی را با شناسه 10101 گزاردهاند را  
بیابید.



# Set Comparison





# Set Comparison – “some” Clause

- 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

بنی حداقل یکی some

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

برای مقایسه some است

اسامی مربیانی را بیابید که حقوق آنها بیشتر از برخی (حداقل یک) مربی در بخش زیست شناسی است.



# Definition of “some” Clause

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

بنی 5 کمتر از یکی از عدای توی ستون است پس درست است  
 $(5 < \text{some } \quad ) = \text{true}$  (read: 5 < some tuple in the relation)

$(5 < \text{some } \quad ) = \text{false}$

$(5 = \text{some } \quad ) = \text{true}$  +

حافل یکی توی این مجموعه باشد که با 5 یکی نباشد  
 $(5 \neq \text{some } \quad ) = \text{true}$  (since  $0 \neq 5$ )

$(= \text{some}) \equiv \text{in } \quad$  بنی 5 حافل با یکی از این اعداد توی مجموعه یکی باشد  
However,  $(\neq \text{some}) \neq \text{not in}$  بنی 5 داخل این مجموعه باشد



بنی با هیچکدام یکی نباشد ولی مخالف some بنی حافل با یکی، یکی نباشد not in

وقتی مساوی **some** داریم ینی هم ارز با **in** است  
داره میگه توی هیچ کدوم از اینا نباشه برای همین با مخالف **some** یکی نیست **not in**



# 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');
```

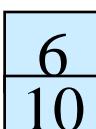
**all** : باید تمامشون برقرار باشه  
**some** میگه یکیش هم برقرار بود او کیه  
اسامی تمام مربیانی که حقوق آنها از حقوق همه مربیان گروه زیست شناسی بیشتر است را بیابید.



# Definition of “all” Clause

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

(5 < all  ) = false      5 باید کمتر از تمامی مقادیر باشد

(5 < all  ) = true

(5 = all  ) = false

(5 ≠ all  ) = true (since  $5 \neq 4$  and  $5 \neq 6$ )

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

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



# 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**  $r \Leftrightarrow r = \emptyset$

یعنی میاد بررسی میکنه که وجود داره این خاصیته برای این مورد یا نه و به مقدارش هیچ کاری نداره **exists**



# 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**  $r \Leftrightarrow r = \emptyset$

## Use of “exists” Clause

- Yet another way of specifying the query “Find all courses taught in both the Fall 2017 semester and in the Spring 2018 semester”

```
select course_id
  from section as S
 where semester = 'Fall' and year = 2017 and
       exists (select *
                  from section as T
                 where semester = 'Spring' and year= 2018
                   and S.course_id = T.course_id);
```

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

یہ ردپائی از بالائی باید توی پابینی باشے

روش دیگری برای تعیین پرس و جو «**همه دروس تدریس شده در ترم پاییز 2017 و در ترم بهار 2018 را نمیاد ستون ها رو ببینه بلکه فقط سطرها رو چک میکنه بباید.**



# Use of “not exists” Clause

- 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

همه دانش آموزانی را که تمام دروس ارائه شده در بخش زیست شناسی را گذرانده اند، بیایید.



# 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

نکته: هر چیزی که توی سلکت نوشته شد باید توی **group by** باشد به غیر از توابع تجمعی ولی اگر چیزی توی **group by** بود می تواند توی سلکت نباشد (:))



# 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)
```

رو می ریزه توی ستون value توی رابطه (max(budget  
max\_budget

- ```
select department.name  
      from department, max_budget  
     where department.budget = max_budget.value;
```

میاد یک رابطه موقت ایجاد میکنه with



# 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),
```

جمع حقوق هر دانشکده

```
dept_total_avg(value) as  
  (select avg(value)  
   from dept_total)
```

این هم یک with دیگر است و کارش این است که میاد میانگین  
جمع های مرحله قبلی رو به دست میاره  
و این هم یک تک عدد است

```
select dept_name  
from dept_total, dept_total_avg  
where dept_total.value > dept_total_avg.value;
```

جمع حقوق دانشکده از میانگین جمع حقوق همه دانشکده ها بیشتر باشد



# 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

تعداد کل سطرها را می‌دهد (\*)  
count

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

اگر subquery توى سلکت استفاده شود حتماً یک مقدار برمی‌گرداند

- Runtime error if subquery returns more than one result tuple

نکته: اول from department اجرا می‌شود و بعد می‌رود توى سلکت



# 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

- SQL expresses a deletion by:
  - **delete from  $r$  where  $P$ ;**
- where  $P$  represents a predicate and  $r$  represents a relation. The **delete** statement first finds all tuples  $t$  in  $r$  for which  $P(t)$  is true, and then deletes them from  $r$ . The **where** clause can be omitted, in which case all tuples in  $r$  are deleted.
- Note that a **delete** command operates on only **one** relation.
- The predicate in the **where** clause may be as complex as a **select** command's **where** clause.

نکته:

- 1- از توابع رابطه  $r$  اونجایی که شرط  $p$  برقرار بود اون سطر یا تاپل را حذف میکنه
- 2- اگر **where** رو ننویسیم کل تاپل هارو حذف می کنه و یه رابطه خالی داریم
- 3- دیلیت فقط روی یک رابطه کار میکنه ینی جوین اینا نداره



# Deletion (Cont.)

- 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.

اون *deptname* هایی که توی اون *in* بودن سطرشون حذف میشه

```
delete from instructor  
where dept_name in (select dept_name  
from department  
where building = 'Watson');
```

این فقط اون *deptname* هایی رو به ما میده که اون شرط روش برقرار باشه



# 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 *instructor*, the average salary changes
- Solution used in SQL:
  - First, compute **avg** (salary) and find all tuples to delete
  - Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)



# Insertion

ینی یه سطر به جدول اضافه کن

- 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.)

- Make each student in the Music department who has earned more than 144 credit hours an instructor in the Music department with a salary of \$18,000.

```
insert into instructor
```

```
    select ID, name, dept_name, 18000
          from student
        where dept_name = 'Music' and total_cred > 144;
```

- 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

- Give a 5% salary raise to all instructors  
**update** *instructor*  
    **set** *salary* = *salary* \* 1.05
- Give a 5% salary raise to those instructors who earn less than 70000  
**update** *instructor*  
    **set** *salary* = *salary* \* 1.05  
    **where** *salary* < 70000;
- Give a 5% salary raise to instructors whose salary is less than average  
**update** *instructor*  
    **set** *salary* = *salary* \* 1.05  
    **where** *salary* < (**select avg** (*salary*)  
                **from** *instructor*);



# Updates (Cont.)

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

- set the *tot\_cred* attribute of each *student* tuple to the sum of the credits of courses successfully completed by the student. We assume that a course is successfully completed if the student has a grade that is neither 'F' nor null.

```
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);
```

| <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 3.7 Result of “For all instructors in the university who have taught some course, find their names and the course ID”



# Updates with Scalar Subqueries

- In case a student has not successfully completed any course, the preceding statement would set the `tot_cred` attribute value to null. To set the value to 0 instead, we could use another **update** statement to replace null values with 0;

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

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

Or

- isnull(sum(credits),0)**
- SQL Server: **ISNULL()**, Oracle: **NVL()**, MySQL: **IFNULL()**



# End of Chapter 3