

# **Chapter 3: Introduction to SQL**

**Database System Concepts, 7th Ed.** 

©Silberschatz, Korth and Sudarshan See <a href="https://www.db-book.com">www.db-book.com</a> 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
- 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.



## **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 stores 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.



### **Create Table Construct**

• An SQL relation is defined using the create table command:

#### create table r

```
(A<sub>1</sub> D<sub>1</sub>, A<sub>2</sub> D<sub>2</sub>, ..., A<sub>n</sub> D<sub>n</sub>, (integrity-constraint<sub>1</sub>), ..., (integrity-constraint<sub>k</sub>))
```

- r is the name of the relation
- each A<sub>i</sub> is an attribute name in the schema of relation r
- D<sub>i</sub> is the data type of values in the domain of attribute A<sub>i</sub>
- Example:

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



## **Integrity Constraints in Create Table**

- Types of integrity constraints
  - primary key  $(A_1, ..., A_n)$
  - foreign key (A<sub>m</sub>, ..., A<sub>n</sub>) references r
  - not null
- SQL prevents any update to the database that violates an integrity constraint.
- Example:



### **And a Few More Relation Definitions**

create table takes (



#### 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
  - Remove all tuples from the student relation
    - delete from student
- Drop Table
  - drop table r
- Alter
  - 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 exiting tuples in the relation are assigned null as the value for the new attribute.
  - alter table r drop A
    - where A is the name of an attribute of relation r
    - Dropping of attributes not supported by many databases.



# **Basic Query Structure**

A typical SQL query has the form:

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

- A<sub>i</sub> represents an attribute
- R<sub>i</sub> represents a relation
- P is a predicate.
- The result of an SQL query is a relation.



#### 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 ≡ NAME ≡ 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 should not be removed.

**select all** dept\_name **from** instructor

#### dept\_name

Comp. Sci.

Finance

Music

Physics

History

**Physics** 

Comp. Sci.

History

Finance

**Biology** 

Comp. Sci.

Elec. Eng.



# 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** *F*00

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"



# 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



#### 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

name

Katz Brandt



#### 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.
- 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).



# **Examples**

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

name	course_id
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	рну-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	вю-101
Crick	вю-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
     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



# **Self Join Example**

Relation emp-super

person	supervisor
Bob	Alice
Mary	Susan
Alice	David
David	Mary

- Find the supervisor of "Bob"
- Find the supervisor of the supervisor of "Bob"



## **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 in alphabetic order the names of all instructors

**select distinct** *name* **from** *instructor* **order by** *name* 

- We may specify desc for descending order or asc for ascending order, for each attribute; ascending order is the default.
  - Example: order by name desc
- Can sort on multiple attributes
  - Example: order by dept\_name, name



### **Where Clause Predicates**

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



## **Set Operations**

Find courses that ran in Fall 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)
```



# **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.



#### **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

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



# **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: (true and unknown) = unknown,
     (false and unknown) = false,
     (unknown and unknown) = unknown
  - or: (unknown or true) = true,
     (unknown or false) = unknown
     (unknown or unknown) = unknown
- Result of where clause predicate is treated as false if it evaluates to unknown



## **Aggregate Functions**

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

avg: average value

min: minimum value

max: maximum value

sum: sum of values

count: number of values



# **Aggregate Functions 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 (\*)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;

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

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



# **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



# **Nested Subqueries**

- SQL provides a mechanism for the nesting of subqueries. A subquery is a select-from-where expression that is nested within another query.
- The nesting can be done in the following SQL query

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

#### as follows:

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

B is an attribute and operation> to be defined later.

Select clause:

 $A_i$  can be replaced be a subquery that generates a single value.



# **Set Membership**



#### **Set Membership**

Find courses offered in Fall 2017 and in Spring 2018

Find courses offered in Fall 2017 but not in Spring 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

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



# **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



#### **Definition of "some" Clause**

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

$$(5 < \mathbf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(\mathsf{read}: \ 5 < \mathsf{some tuple in the relation})$$

$$(5 < \mathbf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{false}$$

$$(5 = \mathbf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathbf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathbf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathsf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathsf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathsf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathsf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathsf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathsf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathsf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathsf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathsf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathsf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathsf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathsf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathsf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathsf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathsf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathsf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathsf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$

$$(5 \neq \mathsf{some} \quad \boxed{0} \\ 5 \\ ) = \mathsf{true}$$



# **Set Comparison – "all" Clause**

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



#### **Definition of "all" Clause**

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

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

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

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

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

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



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



#### **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"

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



#### Use of "not exists" Clause

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

- First nested query lists all courses offered in Biology
- Second nested query lists all courses a particular student took
- Note that  $X Y = \emptyset \Leftrightarrow X \subset Y$
- 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" if a given subquery contains no duplicates.
- Find all courses that were offered at most once in 2017



# **Subqueries in the From Clause**



## **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."

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



#### With Clause

- The with clause provides a way of defining a temporary relation whose definition is available only to the query in which the with clause occurs.
- Find all departments with the maximum budget

```
with max_budget (value) as
          (select max(budget)
          from department)
select department.name
from department, max_budget
where department.budget = max_budget.value;
```



## **Complex Queries using With Clause**

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



## **Scalar Subquery**

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

Runtime error if subquery returns more than one result tuple



#### **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.



## **Deletion (Cont.)**

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

- Problem: as we delete tuples from instructor, the average salary changes
- Solution used in SQL:
  - 1. First, compute **avg** (salary) and find all tuples to delete
  - 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);
```



# **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;</li>

Give a 5% salary raise to instructors whose salary is less than average



## **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;</pre>
```

- 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</pre>
```



## **Updates with Scalar Subqueries**

Recompute and update tot\_creds value for all students

- Sets tot\_creds to null for students who have not taken any course
- Instead of sum(credits), use:

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



# **End of Chapter 3**