



CS425 – Fall 2016

Boris Glavic

Chapter 4: Introduction to SQL

Modified from:
Database System Concepts, 6th Ed.
©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use



Chapter 4: Introduction to SQL

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



Textbook: Chapter 3

CS425 – Fall 2016 – Boris Glavic

4.2

©Silberschatz, Korth and Sudarshan



History

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

CS425 – Fall 2016 – Boris Glavic

4.3

©Silberschatz, Korth and Sudarshan



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.

CS425 – Fall 2016 – Boris Glavic

4.4

©Silberschatz, Korth and Sudarshan



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.

CS425 – Fall 2016 – Boris Glavic

4.5

©Silberschatz, Korth and Sudarshan



Create Table Construct

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

```
create table r (A1 D1, A2 D2, ..., An Dn,
              (integrity-constraint1),
              ...,
              (integrity-constraintk))
```

- r is the name of the relation
- each A_i is an attribute name in the schema of relation r
- D_i is the data type of values in the domain of attribute A_i

- Example:

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

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

CS425 – Fall 2016 – Boris Glavic

4.6

©Silberschatz, Korth and Sudarshan



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**



And a Few More Relation Definitions

- **create table** *student* (
 - ID varchar(5),
 - $name$ varchar(20) not null,
 - $dept_name$ varchar(20),
 - tot_cred numeric(3,0),
 - primary key** (ID),
 - foreign key** ($dept_name$) **references** *department*);
- **create table** *takes* (
 - ID varchar(5),
 - $course_id$ varchar(8),
 - sec_id varchar(8),
 - $semester$ varchar(6),
 - $year$ numeric(4,0),
 - $grade$ varchar(2),
 - primary key** ($ID, course_id, sec_id, semester, year$),
 - foreign key** (ID) **references** *student*,
 - foreign key** ($course_id, sec_id, semester, year$) **references** *section*);
- Note: sec_id can be dropped from primary key above, to ensure a student cannot be registered for two sections of the same course in the same semester



Even more

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



Drop and Alter Table Constructs

- **drop table** *student*
 - Deletes the table and its contents
- **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
 - And more ...



Basic Query Structure

- The SQL **data-manipulation language (DML)** provides the ability to query information, and insert, delete and update tuples
- A typical SQL **query** has the form:

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

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



The select Clause

- The **select** clause list the attributes desired in the result of a query
 - corresponds to the projection operation of the relational algebra
- Example: find the names of all instructors:


```
select name
from instructor
```
- NOTE: SQL keywords are case insensitive (i.e., you may use upper- or lower-case letters.)
 - E.g. $Name \equiv NAME \equiv name$
 - Some people use upper case wherever we use bold font.

The select Clause (Cont.)

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after select.
- Find the names of all departments with instructor, and remove duplicates


```
select distinct dept_name
from instructor
```
- The (redundant) keyword **all** specifies that duplicates not be removed.


```
select all dept_name
from instructor
```

CS425 – Fall 2016 – Boris Glavic 4.13 ©Silberschatz, Korth and Sudarshan

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.
 - Most systems also support additional functions
 - E.g., substring
 - Most systems allow user defined functions (UDFs)
- 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.

CS425 – Fall 2016 – Boris Glavic 4.14 ©Silberschatz, Korth and Sudarshan

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)

CS425 – Fall 2016 – Boris Glavic 4.15 ©Silberschatz, Korth and Sudarshan

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.
- SQL standard: any valid expression that returns a boolean result
 - Vendor specific restrictions may apply!

CS425 – Fall 2016 – Boris Glavic 4.16 ©Silberschatz, Korth and Sudarshan

Cartesian Product: *instructor X teaches*

instructor				teaches				
ID	name	dept_name	salary	ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2009
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2010
32343	El Said	History	60000	15151	MU-199	1	Spring	2010
...	22222	PHY-101	1	Fall	2009

inst.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2009
...

CS425 – Fall 2016 – Boris Glavic 4.17 ©Silberschatz, Korth and Sudarshan

Joins

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


```
select name, course_id
from instructor, teaches
where instructor.ID = teaches.ID
```
- Find the course ID, semester, year and title of each course offered by the Comp. Sci. department


```
select section.course_id, semester, year, title
from section, course
where section.course_id = course.course_id and
dept_name = 'Comp. Sci.'
```

section
course_id
sec_id
semester
year
building
room_no
time_slot_id

→

course
course_id
title
dept_name
credits

CS425 – Fall 2016 – Boris Glavic 4.18 ©Silberschatz, Korth and Sudarshan



Try Writing Some Queries in SQL

- Suggest queries to be written.....

CS425 – Fall 2016 – Boris Glavic

4.19

©Silberschatz, Korth and Sudarshan



Joined Relations

- **Join operations** take two relations and return as a result another relation.
- A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition). It also specifies the attributes that are present in the result of the join
- The join operations are typically used as subquery expressions in the **from** clause

CS425 – Fall 2016 – Boris Glavic

4.20

©Silberschatz, Korth and Sudarshan



Join operations – Example

- Relation *course*

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

- Relation *prereq*

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- Observe that
prereq information is missing for CS-315 and
course information is missing for CS-437

CS425 – Fall 2016 – Boris Glavic

4.21

©Silberschatz, Korth and Sudarshan



Natural Join

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column
 - This is the natural join from relational algebra
- **select ***

from instructor natural join teaches;

ID	name	dept_name	salary	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Bioloev	72000	BIO-301	1	Summer	2010

CS425 – Fall 2016 – Boris Glavic

4.22

©Silberschatz, Korth and Sudarshan



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;

CS425 – Fall 2016 – Boris Glavic

4.23

©Silberschatz, Korth and Sudarshan



Natural Join (Cont.)

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

CS425 – Fall 2016 – Boris Glavic

4.24

©Silberschatz, Korth and Sudarshan



Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values.

CS425 – Fall 2016 – Boris Glavic

4.25

©Silberschatz, Korth and Sudarshan



Left Outer Join

- *course natural left outer join prereq*

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	null

CS425 – Fall 2016 – Boris Glavic

4.26

©Silberschatz, Korth and Sudarshan



Right Outer Join

- *course natural right outer join prereq*

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	null	null	null	CS-101

CS425 – Fall 2016 – Boris Glavic

4.27

©Silberschatz, Korth and Sudarshan



Joined Relations

- **Join operations** take two relations and return as a result another relation.
- These additional operations are typically used as subquery expressions in the **from** clause
- **Join condition** – defines which tuples in the two relations match, and what attributes are present in the result of the join.
- **Join type** – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

Join types
inner join
left outer join
right outer join
full outer join

Join Conditions
natural
on <predicate>
using (A ₁ , A ₁ , ..., A _n)

CS425 – Fall 2016 – Boris Glavic

4.28

©Silberschatz, Korth and Sudarshan



Full Outer Join

- *course natural full outer join prereq*

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	null
CS-347	null	null	null	CS-101

CS425 – Fall 2016 – Boris Glavic

4.29

©Silberschatz, Korth and Sudarshan



Joined Relations – Examples

- *course inner join prereq on course.course_id = prereq.course_id*

course_id	title	dept_name	credits	prereq_id	course_id
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190

- What is the difference between the above, and a natural join?
- *course left outer join prereq on course.course_id = prereq.course_id*

course_id	title	dept_name	credits	prereq_id	course_id
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190
CS-315	Robotics	Comp. Sci.	3	null	null

CS425 – Fall 2016 – Boris Glavic

4.30

©Silberschatz, Korth and Sudarshan

Joined Relations – Examples

- course natural right outer join prereq

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	null	null	null	CS-101

- course full outer join prereq using (course_id)

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	null
CS-347	null	null	null	CS-101

CS425 – Fall 2016 – Boris Glavic 4.31 ©Silberschatz, Korth and Sudarshan

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

CS425 – Fall 2016 – Boris Glavic 4.32 ©Silberschatz, Korth and Sudarshan

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 '\'
```

CS425 – Fall 2016 – Boris Glavic 4.33 ©Silberschatz, Korth and Sudarshan

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.

CS425 – Fall 2016 – Boris Glavic 4.34 ©Silberschatz, Korth and Sudarshan

Case Construct

- Like case, if, and ? Operators in programming languages

```
case
  when c1 then e1
  when c2 then e2
  ...
  [else en]
end
```

- Each c_i is a condition
- Each e_i is an expression
- Returns the first e_i for which c_i evaluates to true
 - If none of the c_i is true, then return e_n (**else**)
 - If there is no else return null

CS425 – Fall 2016 – Boris Glavic 4.35 ©Silberschatz, Korth and Sudarshan

Case Construct Example

- Like case, if, and ? Operators in programming languages

```
select
  name,
  case
    when salary > 1000000 then 'premium'
    else 'standard'
  end as customer_group
from customer
```

CS425 – Fall 2016 – Boris Glavic 4.36 ©Silberschatz, Korth and Sudarshan



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**
- Order is not expressible in the relational model!**

CS425 - Fall 2016 - Boris Glavic

4.37

©Silberschatz, Korth and Sudarshan



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');**

CS425 - Fall 2016 - Boris Glavic

4.38

©Silberschatz, Korth and Sudarshan



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)
```

CS425 - Fall 2016 - Boris Glavic

4.39

©Silberschatz, Korth and Sudarshan



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**

CS425 - Fall 2016 - Boris Glavic

4.40

©Silberschatz, Korth and Sudarshan



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 and comparisons involving *null* evaluate to *null*
 - Example: $5 + null$ returns *null*
 - $null > 5$ returns *null*
 - $null = null$ 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
```

CS425 - Fall 2016 - Boris Glavic

4.41

©Silberschatz, Korth and Sudarshan



Null Values and Three Valued Logic

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

CS425 - Fall 2016 - Boris Glavic

4.42

©Silberschatz, Korth and Sudarshan



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
- Most DBMS support user defined aggregation functions

CS425 – Fall 2016 – Boris Glavic

4.43

©Silberschatz, Korth and Sudarshan



Aggregate Functions (Cont.)

- Find the average salary of instructors in the Computer Science department
 - select avg (salary)**
from instructor
where dept_name= 'Comp. Sci.' ;
- Find the total number of instructors who teach a course in the Spring 2010 semester
 - select count (distinct ID)**
from teaches
where semester = 'Spring' and year = 2010
- Find the number of tuples in the *course* relation
 - select count (*)**
from course;

CS425 – Fall 2016 – Boris Glavic

4.44

©Silberschatz, Korth and Sudarshan



Aggregate Functions – Group By

- Find the average salary of instructors in each department
 - select dept_name, avg (salary)**
from instructor
group by dept_name;
 - Note: departments with no instructor will not appear in result

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

CS425 – Fall 2016 – Boris Glavic

4.45

©Silberschatz, Korth and Sudarshan



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;

CS425 – Fall 2016 – Boris Glavic

4.46

©Silberschatz, Korth and Sudarshan



Aggregate Functions – Having Clause

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

```
select dept_name, avg (salary)
from instructor
group by dept_name
having avg (salary) > 42000;
```

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

CS425 – Fall 2016 – Boris Glavic

4.47

©Silberschatz, Korth and Sudarshan



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

CS425 – Fall 2016 – Boris Glavic

4.48

©Silberschatz, Korth and Sudarshan

Empty Relations and Aggregates

- What if the input relation is empty
- Conventions:
 - **sum**: returns *null*
 - **avg**: returns *null*
 - **min**: returns *null*
 - **max**: returns *null*
 - **count**: returns 0

CS425 – Fall 2016 – Boris Glavic 4.49 ©Silberschatz, Korth and Sudarshan

Duplicates

- In relations with duplicates, SQL can define how many copies of tuples appear in the result.
- **Multiset (bag semantics)** 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 σ_θ , then there are c_1 copies of t_1 in $\sigma_\theta(r_1)$.
 2. $\Pi_A(r)$: For each copy of tuple t_i in r_1 , there is a copy of tuple $\Pi_A(t_i)$ in $\Pi_A(r_1)$ where $\Pi_A(t_i)$ denotes the projection of the single tuple t_i .
 3. $r_1 \times r_2$: If there are c_1 copies of tuple t_1 in r_1 and c_2 copies of tuple t_2 in r_2 , there are $c_1 \times c_2$ copies of the tuple $t_1 \times t_2$ in $r_1 \times r_2$.

CS425 – Fall 2016 – Boris Glavic 4.50 ©Silberschatz, Korth and Sudarshan

Multiset Relational Algebra

- Pure relational algebra operates on **set-semantics** (no duplicates allowed)
 - e.g. after projection
- Multiset (**bag-semantics**) relational algebra retains duplicates, to match SQL semantics
 - SQL duplicate retention was initially for efficiency, but is now a feature
- Multiset relational algebra defined as follows
 - **selection**: has as many duplicates of a tuple as in the input, if the tuple satisfies the selection
 - **projection**: one tuple per input tuple, even if it is a duplicate
 - **cross product**: If there are m copies of t_1 in r , and n copies of t_2 in s , there are $m \times n$ copies of $t_1.t_2$ in $r \times s$
 - Other operators similarly defined
 - ↳ E.g. **union**: $m + n$ copies, **intersection**: $\min(m, n)$ copies
 - difference**: $\max(0, m - n)$ copies

CS425 – Fall 2016 – Boris Glavic 4.51 ©Silberschatz, Korth and Sudarshan

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 A1, A2, ..., An
from r1, r2, ..., rm
where P
      
```

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

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

CS425 – Fall 2016 – Boris Glavic 4.52 ©Silberschatz, Korth and Sudarshan

SQL and Relational Algebra

- ```

select A1, A2, .. An
from r1, r2, ..., rm
where P

```

 is equivalent to the following expression in multiset relational algebra
 
$$\Pi_{A_1, \dots, A_n}(\sigma_P(r_1 \times r_2 \times \dots \times r_m))$$
- ```

select A1, A2, sum(A3)
from r1, r2, ..., rm
where P
group by A1, A2
      
```

 is equivalent to the following expression in multiset relational algebra

$$A_1, A_2 \int \text{sum}(A_3) (\sigma_P(r_1 \times r_2 \times \dots \times r_m))$$

CS425 – Fall 2016 – Boris Glavic 4.53 ©Silberschatz, Korth and Sudarshan

SQL and Relational Algebra

- More generally, the non-aggregated attributes in the **select** clause may be a subset of the **group by** attributes, in which case the equivalence is as follows:


```

select A1, sum(A3) AS sumA3
from r1, r2, ..., rm
where P
group by A1, A2
      
```

 is equivalent to the following expression in multiset relational algebra

$$\Pi_{A_1, \text{sum}A_3}(A_1, A_2 \int \text{sum}(A_3) \text{ as } \text{sum}A_3 (\sigma_P(r_1 \times r_2 \times \dots \times r_m)))$$

CS425 – Fall 2016 – Boris Glavic 4.54 ©Silberschatz, Korth and Sudarshan



Subqueries in the From Clause

- SQL allows a subquery expression to be used in the **from** clause
- Find the average instructors' salaries of those departments where the average salary is greater than \$42,000.

```
select dept_name, avg_salary
from (select dept_name, avg (salary) as avg_salary
      from instructor
      group by dept_name)
where avg_salary > 42000;
```

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

```
select dept_name, avg_salary
from (select dept_name, avg (salary)
      from instructor
      group by dept_name)
as dept_avg (dept_name, avg_salary)
where avg_salary > 42000;
```

CS425 - Fall 2016 - Boris Glavic 4.55 ©Silberschatz, Korth and Sudarshan



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.

CS425 - Fall 2016 - Boris Glavic 4.56 ©Silberschatz, Korth and Sudarshan



Example Query

- Find courses offered in Fall 2009 and in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
       course_id in (select course_id
                    from section
                    where semester = 'Spring' and year= 2010);
```

- Find courses offered in Fall 2009 but not in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
       course_id not in (select course_id
                        from section
                        where semester = 'Spring' and year=
2010);
```

CS425 - Fall 2016 - Boris Glavic 4.57 ©Silberschatz, Korth and Sudarshan



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.

CS425 - Fall 2016 - Boris Glavic 4.58 ©Silberschatz, Korth and Sudarshan



Quantification

- 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');
```

CS425 - Fall 2016 - Boris Glavic 4.59 ©Silberschatz, Korth and Sudarshan



Definition of Some Clause

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

$(5 < \text{some } \begin{matrix} 0 \\ 5 \\ 6 \end{matrix}) = \text{true}$ (read: 5 < some tuple in the relation)

$(5 < \text{some } \begin{matrix} 0 \\ 5 \end{matrix}) = \text{false}$

$(5 = \text{some } \begin{matrix} 0 \\ 5 \end{matrix}) = \text{true}$

$(5 \neq \text{some } \begin{matrix} 0 \\ 5 \end{matrix}) = \text{true}$ (since $0 \neq 5$)

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

CS425 - Fall 2016 - Boris Glavic 4.60 ©Silberschatz, Korth and Sudarshan



Example Query

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

```
select name
from instructor
where salary > all (select salary
                   from instructor
                   where dept_name = 'Biology');
```

CS425 – Fall 2016 – Boris Glavic

4.61

©Silberschatz, Korth and Sudarshan



Definition of all Clause

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

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

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

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

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

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

CS425 – Fall 2016 – Boris Glavic

4.62

©Silberschatz, Korth and Sudarshan



Test for Empty Relations

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

CS425 – Fall 2016 – Boris Glavic

4.63

©Silberschatz, Korth and Sudarshan



Correlation Variables

- Yet another way of specifying the query “Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester”

```
select course_id
from section as S
where semester = 'Fall' and year = 2009 and
exists (select *
        from section as T
        where semester = 'Spring' and year = 2010
        and S.course_id = T.course_id);
```

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

CS425 – Fall 2016 – Boris Glavic

4.64

©Silberschatz, Korth and Sudarshan



Not Exists

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

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

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

CS425 – Fall 2016 – Boris Glavic

4.65

©Silberschatz, Korth and Sudarshan



Test for Absence of Duplicate Tuples

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

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

CS425 – Fall 2016 – Boris Glavic

4.66

©Silberschatz, Korth and Sudarshan



Correlated Subqueries in the From Clause

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


```
select name, salary, avg_salary
from instructor I1,
     lateral (select avg(salary) as avg_salary
             from instructor I2
             where I2.dept_name= I1.dept_name);
```
- Lateral clause permits later part of the **from** clause (after the lateral keyword) to access correlation variables from the earlier part.
- Note: lateral is part of the SQL standard, but is not supported on many database systems; some databases such as SQL Server offer alternative syntax

CS425 – Fall 2016 – Boris Glavic

4.67

©Silberschatz, Korth and Sudarshan



With Clause

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

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

CS425 – Fall 2016 – Boris Glavic

4.68

©Silberschatz, Korth and Sudarshan



Complex Queries using With Clause

- With clause is very useful for writing complex queries
- Supported by most database systems, with minor syntax variations
- Find all departments where the total salary is greater than the average of the total salary at all departments

```
with dept_total (dept_name, value) as
(select dept_name, sum(salary)
 from instructor
 group by dept_name),
dept_total_avg(value) as
(select avg(value)
 from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value >= dept_total_avg.value;
```

CS425 – Fall 2016 – Boris Glavic

4.69

©Silberschatz, Korth and Sudarshan



Scalar Subquery

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


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


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

CS425 – Fall 2016 – Boris Glavic

4.70

©Silberschatz, Korth and Sudarshan



Query Features Recap - Syntax

- An SQL query is either a Select-from-where block or a set operation
- An SQL query block is structured like this:


```
SELECT [DISTINCT] select_list
[FROM from_list]
[WHERE where_condition]
[GROUP BY group_by_list]
[HAVING having_condition]
[ORDER BY order_by_list]
```
- Set operations


```
[Query Block] set_op [Query Block]
set_op: [ALL] UNION | INTERSECT | EXCEPT
```

CS425 – Fall 2016 – Boris Glavic

4.71

©Silberschatz, Korth and Sudarshan



Query Features Recap - Syntax

- Almost all clauses are optional
- Examples:
 - SELECT * FROM r;**
 - SELECT 1;**
 - Convention: returns single tuple
 - SELECT 'ok' FROM accounts HAVING sum(balance) = 0;**
 - SELECT 1 GROUP BY 1;**
 - SELECT 1 HAVING true;**
 - Let *r* be a relation with two attributes *a* and *b*
 - SELECT a,b FROM r**
 - WHERE a IN (SELECT a FROM r) AND b IN (SELECT b FROM r)**
 - GROUP BY a,b HAVING count(*) > 0;**
- Note:
 - Not all systems support all of this "non-sense"

CS425 – Fall 2016 – Boris Glavic

4.72

©Silberschatz, Korth and Sudarshan

Syntax - SELECT

- **SELECT** [DISTINCT [ON (distinct_list)]] select_list
- select_list
 - List of projection expressions
 - [expr] [AS name]
 - expr
 - Expression over attributes, constants, arithmetic operators, functions, **CASE**-construct, aggregation functions
- distinct_list
 - List of expressions
- Examples:
 - **SELECT DISTINCT ON** (a % 2) a **FROM** r;
 - **SELECT** substring(a, 1,2) **AS** x **FROM** r;
 - **SELECT CASE WHEN** a = 2 **THEN** a **ELSE null END AS** b **FROM** r;
 - **SELECT** a = b **AS is_a_equal_to_b FROM** r;

CS425 – Fall 2016 – Boris Glavic 4.73 ©Silberschatz, Korth and Sudarshan

Syntax - FROM

- **FROM** from_list
- from_list
 - List of from clause expressions
 - subquery | relation | constant_relation | join_expr [alias]
 - subquery
 - Any valid SQL query – alias is not optional
 - relation
 - A relation in the database
 - constant_relation
 - (**VALUES** tuples) – alias is not optional
 - join_expr
 - joins between from_clause entries
 - alias
 - [AS] b [(attribute_name_list)]

CS425 – Fall 2016 – Boris Glavic 4.74 ©Silberschatz, Korth and Sudarshan

Syntax – FROM (cont.)

- Examples (relation r with attributes a and b):
 - **SELECT * FROM** r;
 - **SELECT * FROM** r **AS** g(v,w);
 - **SELECT * FROM** r x;
 - **SELECT * FROM (VALUES (1,2), (3,1)) AS** s(u,v);
 - **SELECT * FROM** r **NATURAL JOIN** s, t;
 - **SELECT * FROM ((r JOIN s ON (r.a = s.c)) NATURAL JOIN (SELECT * FROM t) AS new);**
 - **SELECT * FROM (SELECT * FROM r) AS** r;
 - **SELECT * FROM (SELECT * FROM (SELECT * FROM r) AS r) AS** r;

CS425 – Fall 2016 – Boris Glavic 4.75 ©Silberschatz, Korth and Sudarshan

Syntax - WHERE

- **WHERE** where_condition
- where_condition: A boolean expression over
 - Attributes
 - Constants: e.g., true, 1, 0.5, 'hello'
 - Comparison operators: =, <, >, **IS DISTINCT FROM**, **IS NULL**, ...
 - Arithmetic operators: +, -, /, %
 - Function calls
 - Nested subquery expressions
- Examples
 - **SELECT * FROM** r **WHERE** a = 2;
 - **SELECT * FROM** r **WHERE** true **OR** false;
 - **SELECT * FROM** r **WHERE NOT**(a = 2 **OR** a = 3);
 - **SELECT * FROM** r **WHERE** a **IS DISTINCT FROM** b;
 - **SELECT * FROM** r **WHERE** a < **ANY** (SELECT c **FROM** s);
 - **SELECT * FROM** r **WHERE** a = (SELECT count(*) **FROM** s);

CS425 – Fall 2016 – Boris Glavic 4.76 ©Silberschatz, Korth and Sudarshan

Syntax – GROUP BY

- **GROUP BY** group_by_list
- group_by_list
 - List of expressions
 - Expression over attributes, constants, arithmetic operators, functions, **CASE**-construct, aggregation functions
- Examples:
 - **SELECT** sum(a), b **FROM** r **GROUP BY** b;
 - **SELECT** sum(a), b, c **FROM** r **GROUP BY** b, c;
 - **SELECT** sum(a), b/2 **FROM** r **GROUP BY** b/2;
 - **SELECT** sum(a), b **FROM** r **GROUP BY** b > 5;
 - Incorrect, cannot select b, because it is not an expression in the group by clause
 - **SELECT** sum(a), b **FROM** r **GROUP BY** b **IN** (SELECT c **FROM** s);

CS425 – Fall 2016 – Boris Glavic 4.77 ©Silberschatz, Korth and Sudarshan

Syntax – HAVING

- **HAVING** having_condition
- having_condition
 - Like where_condition except that expressions over attributes have either to be in the **GROUP BY** clause or are aggregated
- Examples:
 - **SELECT** sum(a), b **FROM** r **GROUP BY** b **HAVING** sum(a) > 10;
 - **SELECT** sum(a), b **FROM** r **GROUP BY** b **HAVING** sum(a) + 5 > 10;
 - **SELECT** sum(a), b **FROM** r **GROUP BY** b **HAVING** true;
 - **SELECT** sum(a), b **FROM** r **GROUP BY** b **HAVING** count(*) = 50;
 - **SELECT** b **FROM** r **GROUP BY** b **HAVING** sum(a) > 10;

CS425 – Fall 2016 – Boris Glavic 4.78 ©Silberschatz, Korth and Sudarshan



Syntax – ORDER BY

- **ORDER BY** order_by_list
- order_by_list
 - Like select_list minus renaming
 - Optional [**ASC** | **DESC**] for each item
- Examples:
 - **SELECT * FROM r ORDER BY a;**
 - **SELECT * FROM r ORDER BY b, a;**
 - **SELECT * FROM r ORDER BY a * 2;**
 - **SELECT * FROM r ORDER BY a * 2, a;**
 - **SELECT * FROM r ORDER BY a + (SELECT count(*) FROM s);**

CS425 – Fall 2016 – Boris Glavic

4.79

©Silberschatz, Korth and Sudarshan



Query Semantics

- Evaluation Algorithm (you can do it manually – sort of)
- 1. Compute **FROM** clause
 1. Compute cross product of all items in the **FROM** clause
 - ▶ Relations: nothing to do
 - ▶ Subqueries: use this algorithm to recursively compute the result of subqueries first
 - ▶ Join expressions: compute the join
- 2. Compute **WHERE** clause
 1. For each tuple in the result of 1. evaluate the **WHERE** clause condition
- 3. Compute **GROUP BY** clause
 1. Group the results of step 2. on the **GROUP BY** expressions
- 4. Compute **HAVING** clause
 1. For each group (if any) evaluate the **HAVING** condition

CS425 – Fall 2016 – Boris Glavic

4.80

©Silberschatz, Korth and Sudarshan



Query Semantics (Cont.)

5. Compute **ORDER BY** clause
 5. Order the result of step 4 on the **ORDER BY** expressions
 6. Compute **SELECT** clause
 5. Project each result tuple from step 5 on the **SELECT** expressions
- If the **WHERE**, **SELECT**, **GROUP BY**, **HAVING**, **ORDER BY** clauses have any nested subqueries
 - For each tuple *t* in the result of the **FROM** clause
 - ▶ Substitute the correlated attributes with values from *t*
 - ▶ Evaluate the resulting query
 - ▶ Use the result to evaluate the expression in the clause the subquery occurs in

CS425 – Fall 2016 – Boris Glavic

4.81

©Silberschatz, Korth and Sudarshan



Query Semantics (Cont.)

- Equivalent relational algebra expression
 - **ORDER BY** has no equivalent, because relations are unordered
 - Nested subqueries: need to extend algebra (not covered here)
- Each query block is equivalent to

$$\pi(\sigma(\mathcal{G}(\pi(\sigma(F_1 \times \dots \times F_n))))))$$
 - Where F_i is the translation of the i^{th} **FROM** clause item
 - Note: we leave out the arguments

CS425 – Fall 2016 – Boris Glavic

4.82

©Silberschatz, Korth and Sudarshan



Modification of the Database

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

CS425 – Fall 2016 – Boris Glavic

4.83

©Silberschatz, Korth and Sudarshan



Modification of the Database – Deletion

- Delete all instructors


```
delete from instructor
```
- Delete all instructors from the Finance department


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


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

CS425 – Fall 2016 – Boris Glavic

4.84

©Silberschatz, Korth and Sudarshan



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:
 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)

CS425 – Fall 2016 – Boris Glavic

4.85

©Silberschatz, Korth and Sudarshan



Modification of the Database – Insertion

- Add a new tuple to *course*

```
insert into course
values (' CS-437', ' Database Systems', ' Comp. Sci.', 4);
```
- or equivalently


```
insert into course (course_id, title, dept_name, credits)
values (' CS-437', ' Database Systems', ' Comp. Sci.', 4);
```
- Add a new tuple to *student* with *tot_creds* set to null


```
insert into student
values (' 3003', ' Green', ' Finance', null);
```

CS425 – Fall 2016 – Boris Glavic

4.86

©Silberschatz, Korth and Sudarshan



Insertion (Cont.)

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


```
insert into student
select ID, name, dept_name, 0
from instructor
```
- The **select from where** statement is evaluated fully before any of its results are inserted into the relation (otherwise queries like **insert into table1 select * from table1** would cause problems, if *table1* did not have any primary key defined.

CS425 – Fall 2016 – Boris Glavic

4.87

©Silberschatz, Korth and Sudarshan



Modification of the Database – Updates

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


```
update instructor
set salary = salary * 1.03
where salary > 100000;
update instructor
set salary = salary * 1.05
where salary <= 100000;
```
 - The order is important
 - Can be done better using the **case** statement (next slide)

CS425 – Fall 2016 – Boris Glavic

4.88

©Silberschatz, Korth and Sudarshan



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
```

CS425 – Fall 2016 – Boris Glavic

4.89

©Silberschatz, Korth and Sudarshan



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
```
- Or **COALESCE**(sum(credits),0)
 - **COALESCE** returns first non-null arguments

CS425 – Fall 2016 – Boris Glavic

4.90


©Silberschatz, Korth and Sudarshan



Recap


- SQL queries
 - Clauses: **SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY**
 - Nested subqueries
 - Equivalence with relational algebra
- SQL update, inserts, deletes
 - Semantics of referencing updated relation in **WHERE**
- SQL DDL
 - Table definition: **CREATE TABLE**

CS425 – Fall 2016 – Boris Glavic 4.91 ©Silberschatz, Korth and Sudarshan



End of Chapter 4


Modified from:
Database System Concepts, 6th Ed.
©Silberschatz, Korth and Sudarshan
See www.db-book.com for conditions on re-use



Outline

- Introduction
- Relational Data Model
- Formal Relational Languages (relational algebra)
- **SQL - Intermediate**
- Database Design
- Transaction Processing, Recovery, and Concurrency Control
- Storage and File Structures
- Indexing and Hashing
- Query Processing and Optimization

CS425 – Fall 2016 – Boris Glavic 4.93 ©Silberschatz, Korth and Sudarshan



Advanced SQL Features**

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

CS425 – Fall 2016 – Boris Glavic 4.94 ©Silberschatz, Korth and Sudarshan




Figure 3.02

name
Srinivasan
Wu
Mozart
Einstein
El Said
Gold
Katz
Califiori
Singh
Crick
Brandt
Kim

CS425 – Fall 2016 – Boris Glavic 4.95 ©Silberschatz, Korth and Sudarshan




Figure 3.03

dept_name
Comp. Sci.
Finance
Music
Physics
History
Physics
Comp. Sci.
History
Finance
Biology
Comp. Sci.
Elec. Eng.

CS425 – Fall 2016 – Boris Glavic 4.96 ©Silberschatz, Korth and Sudarshan

