


Databases – Introduction to SQL (Part 2)



Jaeyong Choi
Dept. of Software, Gachon University



DML



❑ Data Manipulation Language

- ❑ The SQL commands that deals with the manipulation of data present in database belong to DML
 - ❑ **SELECT** – is used to retrieve data from the a database.
 - ❑ **INSERT** – is used to insert data into a table.
 - ❑ **UPDATE** – is used to update existing data within a table.
 - ❑ **DELETE** – is used to delete records from a database table.



Insertion to the Database

❑ Insertion: Add a new tuple to database

- ❑ all data with the same order

- ❑ **insert into** course

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

- ❑ specify all columns, order can be changed

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

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

- ❑ specify some columns

- insert into** course (**title**, **credits**)

- values** ('Database Systems', 4);

- ❑ Add a new tuple to student with tot_creds set to null

- ❑ **insert into** student **values** ('3003', 'Green', 'Finance', **null**);

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



Insertion to the Database

❑ Insertion *cont'd*

- ❑ 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** statement is evaluated fully before any of its results are inserted into the relation

- ❑ Add several data
 - ❑ **insert into** student **values**
(**'1234'**, **'ee'**, **'finance'**, **4**),
(**'3456'**, **'gg'**, **'comp. sci'**, **2**),
(**'5678'**, **'ff'**, **'comp. sci'**, **2**);

ID	name	dept_name	tot_cred
10101	srinivsan	comp. sci	0
1012	cc	finanace	NULL
12121	wu	finance	0
1365	dd	comp. sci	NULL
2003	aa	comp. sci	NULL
2027	bb	comp. sci	NULL
3003	Green	comp. sci	NULL
45565	katz	comp. sci	0
76543	singh	finance	0
NULL	NULL	NULL	NULL



Exercise

❑ Insert some information by referring the table



ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
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

Figure 2.1 The *instructor* relation.

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

Figure 2.2 The *course* relation.

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

Figure 4.1 The *student* relation.

dept_name	building	budget
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
PhysicsI	Watson	70000

Figure 2.5 The *department* relation.

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

Figure 2.6 The *section* relation.

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

Figure 2.7 The *teaches* relation.



Update the Database

- ❑ Drop table (테이블이나 DB제거)
 - ❑ **Drop table r ;**
- ❑ Alter table (테이블의 이름, 컬럼 등 변경)
 - ❑ **alter table r add A D , add primary key (A);**
 - ❑ 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
 - ❑ You can add any constraint such as primary key
 - ❑ **alter table r drop A ;**
 - ❑ A is the name of an attribute of relation r
 - ❑ Dropping of attributes not supported by many databases
 - ❑ **alter table r_1 rename r_2 ;**
 - ❑ Rename table r_1 to table r_2
 - ❑ **Alter table r modify column A int default 1000;**
 - ❑ **Alter table r change column A new_ A int default 1000;**
 - ❑ Modify and change can change the state of a column. change can also change column names.
 - ❑ **Alter table r Add constraint A Foreign key(ID) references r_2 (ID);**



Update the Database

- ❑ Delete (테이블의 데이터를 제거)
 - ❑ Delete all instructors
 - ❑ **delete from** instructor;
 - ❑ Delete all instructors from the Finance department
 - ❑ **delete from** instructor
where dept_name= 'Finance';
 - ❑ Delete all 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');



Update the Database

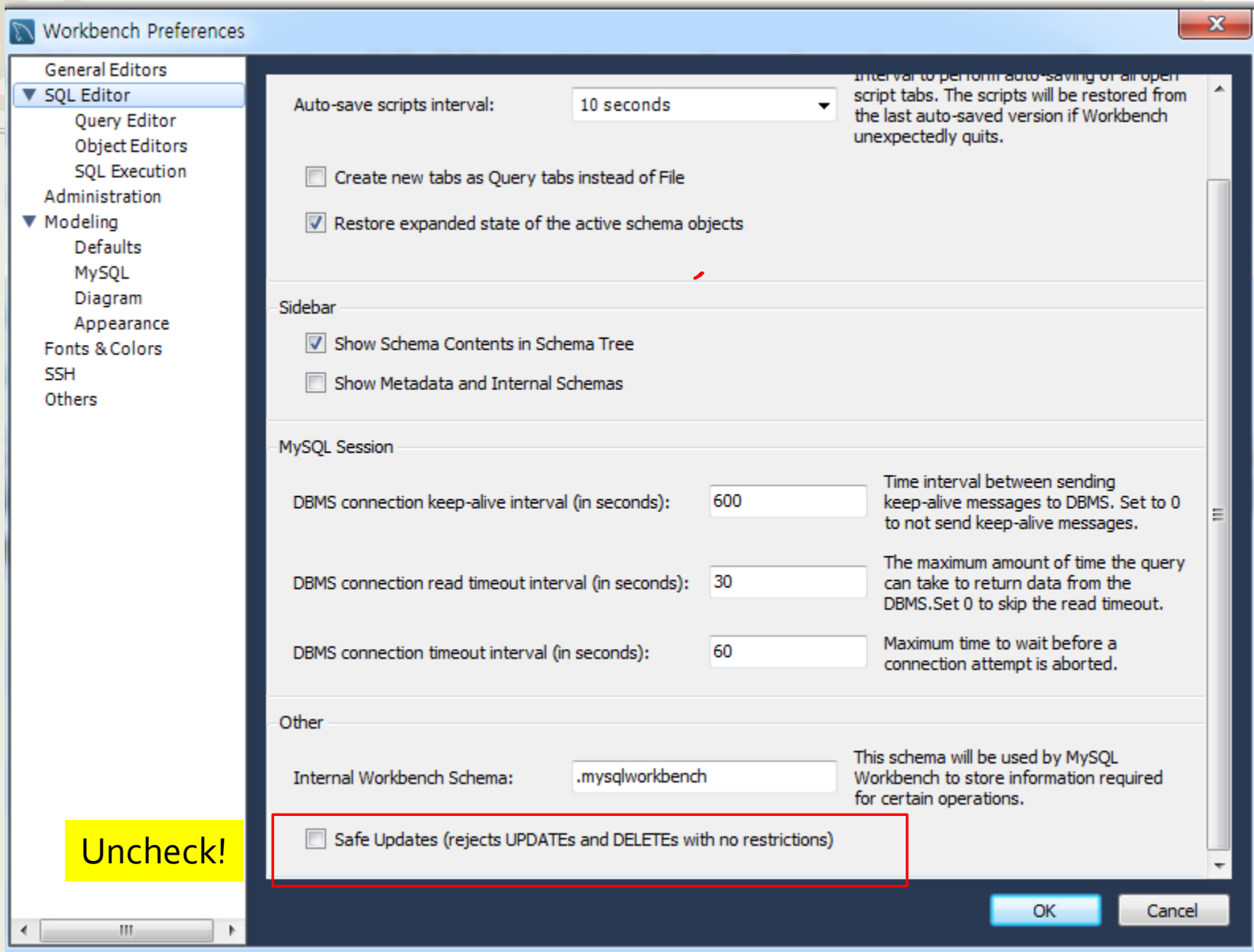
❑ Updates (데이터 변경)

- ❑ Increase salaries of instructors whose salary is over \$100,000 by 3%

- ❑ **update** instructor
set salary = salary * 1.03
where salary > 100000;

ID	name	dept_name	salary
10101	srinivasan	comp. sci	70963.33
12121	wu	biology	98256.92
14141	wu2	biology	109174.36
15151	choi	comp. sci	98256.92
NULL	NULL	NULL	NULL

- ❑ **update** instructor **set** ID=10102 **where** ID =10101;
- ❑ Unsafe option issue
 - ❑ Edit → preference → SQL editor → uncheck safe option





Basic Structure of SQL Queries

❑ Basic query structure

❑ A typical SQL query has the form:

❑ **select** A_1, A_2, \dots, A_n
from r_1, r_2, \dots, r_m
where P_i

❑ A_i represents an attribute

❑ R_i represents a relation

❑ P is a predicate

■ $<, >, \leq, \geq, =, \neq, \text{AND}, \text{OR}, \text{NOT}$

❑ NOTE: the result of an SQL query is a **relation**

❑ Examples

❑ **select** * **from** student;

❑ **select** * **from** course;



Select clause



- ❑ Lists the attributes desired in the result of a query
- ❑ Corresponds to the relational projection operation
- ❑ E.g., Find the names of all instructors:
 - ❑ **select** name **from** instructor;
- ❑ NOTE: SQL names are case insensitive
 - ❑ Some people use upper case wherever we use bold font



Select clause cont'd

- ❑ SQL allows duplicates in relations as well as query results
- ❑ For elimination of duplicates, insert the keyword **distinct** after **select**
- ❑ E.g., Find the department names of all instructors, and **remove duplicates**
 - ❑ **select distinct** dept_name **from** instructor;
- ❑ **all** specifies that duplicates should not be removed
 - ❑ **select all** dept_name **from** instructor;



Select clause cont'd

- ❑ An asterisk (*) in the select clause denotes "all attributes"
 - ▣ **select * from instructor;**
- ❑ An attribute can be a literal(상수) with from clause
 - ▣ **select 'A' from instructor;**
 - ▣ Results in a table with one column and N rows (number of tuples in the *instructor* table), each row with value "A"
- ❑ An attribute can be a literal without from clause
 - ▣ **select '437';**
 - ▣ Results in a table with one column and a single row with value "437"
- ❑ Can give the column a name
 - ▣ **select '437' as FOO;**



Select clause cont'd

- ❑ Can contain arithmetic expressions involving the operations $+$, $-$, $*$, and $/$ on constants or tuple attributes
- ❑ Example:
 - ❑ **select** ID, name, salary/12 **from** instructor;
 - ❑ Returns 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 **as** clause:
 - ❑ **select** ID, name, salary/12 **as** monthly_salary **from** instructor;



Where clause

- ❑ Specifies the conditions that the result must satisfy
- ❑ Corresponds to the selection predicate of the relational algebra
- ❑ E.g., Find all instructors in Comp. Sci. department
 - ❑ **select** name
 from instructor
 where dept_name = "Comp. Sci.";
- ❑ Comparison results can be combined using the logical connectives *and, or, and not*
- ❑ E.g., Find all instructors in Comp. Sci. dept with salary > 80000
 - ❑ **select** name
 from instructor
 where dept_name = 'Comp. Sci.' **and** salary > 80000;
- ❑ Comparisons can be applied to results of arithmetic expressions



Where clause predicates

- ❑ SQL includes **between** comparison operator
 - ▣ E.g., Find the names of all instructors with the salary between \$90,000 and \$100,000 (i.e., $\geq \$90,000$ 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');**

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
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

Figure 2.1 The *instructor* relation.

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

Figure 2.7 The *teaches* relation.



From clause







- ❑ Lists the relations involved in the query
- ❑ Corresponds to the Cartesian product operation of the relational algebra
 - ▣ E.g., Find the **Cartesian product** instructor × 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 is not very useful directly, but useful combined with where-clause condition



From clause

instructor





Result Grid			 Filter Rows:	
	ID	name	dept_name	tot_cred
	2011	mint	cs	50
	2012	white	management	200
	3003	Green	Finance	NULL
	3011	red	cs	300
	437	yello	cs	350
	NULL	NULL	NULL	NULL



`select * from instructor, teaches;`

Result Grid		Filter Rows:			Export:		Wrap Cell Content:	
	ID	name	dept_name	tot_cred	course_id	title	dept_name	credits
▶	2011	mint	cs	50	CS-111	db	comp. sci	2
	2011	mint	cs	50	CS-437	Database Systems	comp. sci	4
	2012	white	management	200	CS-111	db	comp. sci	2
	2012	white	management	200	CS-437	Database Systems	comp. sci	4
	3003	Green	Finance	NULL	CS-111	db	comp. sci	2
	3003	Green	Finance	NULL	CS-437	Database Systems	comp. sci	4
	3011	red	cs	300	CS-111	db	comp. sci	2
	3011	red	cs	300	CS-437	Database Systems	comp. sci	4
	437	yello	cs	350	CS-111	db	comp. sci	2
	437	yello	cs	350	CS-437	Database Systems	comp. sci	4

teaches

Result Grid		 Filter Rows:	<input type="text"/>	 Edit
	course_id	title	dept_name	credits
	CS-111	db	comp. sci	2
	CS-437	Database Systems	comp. sci	4
	NULL	NULL	NULL	NULL



Additional Basic Operations

❑ Rename operation

- ❑ SQL allows renaming relations and attributes using **as** clause:

- ❑ *old-name as new-name*

- ❑ E.g., 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.';**

- ❑ T and S can be thought of as copies of the relation *instructor*; they are declared as *aliases*, that is as alternative names, for the relation *instructor*

T	ID	name	dept_name	salary
	10101	srinivasan	comp. sci	65000.00
	12121	wu	biology	90000.00
	14141	wu2	biology	100000.00
	15151	choi	comp. sci	90000.00
*	NULL	NULL	NULL	NULL

S	ID	name	dept_name	salary
	10101	srinivasan	comp. sci	65000.00
	12121	wu	biology	90000.00
	14141	wu2	biology	100000.00
	15151	choi	comp. sci	90000.00
*	NULL	NULL	NULL	NULL

name
wu
wu2
wu2
choi



String operations

- ❑ SQL includes a string-matching operator **like** for comparisons on character strings
 - ❑ It uses patterns that are described using two special characters.
 - ❑ **percent (%)** – matches any substring
 - ❑ **underscore (_)** – matches any single character
- ❑ E.g., Find the names of all instructors whose name includes the substring "dar".
 - ❑ **select name from instructor where name like '%dar%';**
- ❑ E.g., Find the names that begin with the letter m and have the letter d as the third letter
 - ❑ **select name from instructor where name like 'm_d%';**
- ❑ E.g., Match the string "100%"
 - ❑ **... like '100\%' escape '\';**
 - ❑ We use backslash (\) as the escape character



String operations



- ❑ Patterns are **case sensitive**
 - ❑ '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 '||' in Oracle)
 - ❑ `select concat(LastName, ', ' , FirstName) AS Name FROM Person`
 - ❑ Converting from upper to lower case (and vice versa)
 - ❑ `select UPPER('louder please');`
 - ❑ `select lower('AABB');`
 - ❑ Finding string length, extracting substrings, etc.
 - ❑ `select length("SQL Tutorial") AS LengthOfString;`
 - ❑ `select substring("MySql test", 1,5);`
 - (expression, start, length)
 - `select substring("MySql test", 1,5); → MySql`



Ordering the display of tuples

- ❑ List the names of all instructors in an alphabetic order
 - ❑ `select distinct name from instructor order by name;`
 - ❑ `select distinct name from instructor order by name desc;`
- ❑ We may specify **desc** or **asc** for descending or ascending order, for each attribute
 - ❑ Ascending order is the default
 - ❑ E.g., ... **order by** name **desc**;
- ❑ Can sort on multiple attributes
 - ❑ E.g., ... **order by** name **asc**, dept_name **desc**;

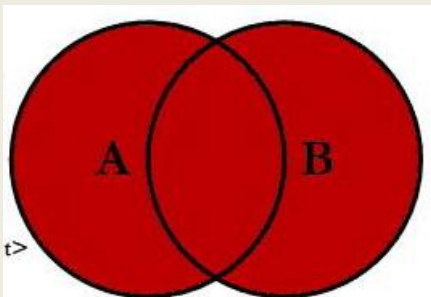
	name
▶	choi
	srinivasan
	wu
	wu2

	name
▶	wu2
	wu
	srinivasan
	choi

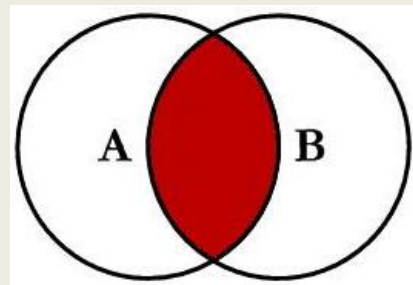


Set Operations

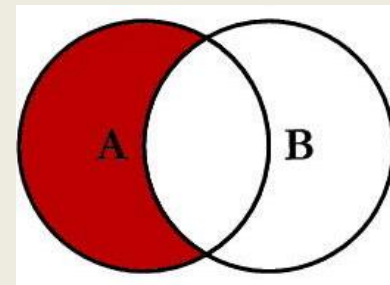
- ❑ Set operations : union, intersect, except
 - ❑ Link records from multiple tables
 - ❑ Each of the above operations automatically eliminates duplicates
 - ❑ To retain all duplicates, use the corresponding multiset versions: **union all, intersect all, except all**



union



intersect



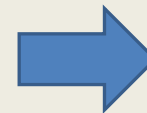
except



Set Operations (Union)

- ❑ 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);

	course_id	sec_id	semester	year	building	room_no	time_slot_id
▶	cs-101	1	fall	2009	painter	514	1
	cs-101	1	spring	2010	painter	514	1
	cs-110	2	fall	2009	it315	514	4
	cs-200	2	spring	2010	it315	514	4
	cs-250	2	spring	2009	it315	514	4
	cs-315	1	spring	2010	painter	514	2
	cs-319	2	spring	2020	it315	514	4
	fin-101	2	fall	2019	it314	514	3
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL



Result Grid	
	course_id
▶	cs-101
	cs-110
	cs-200
	cs-315



Set Operations (Intersect, Except)

- ❑ 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);



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 a value does not exist
- ❑ Any arithmetic expression involving null returns null
 - ▣ E.g., $5 + \text{null}$ returns null
- ❑ The predicate **is null** can be used to check for null values
 - ▣ E.g., Find all instructors whose salary is unknown
 - ▣ **select name from instructor where salary is null;**



Null Values



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



Aggregate 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

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



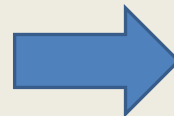
Aggregate Functions

□ Group By

■ Find the *average* salary of instructors in each department

■ **select** dept_name, **avg**(salary) **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



Aggregate Functions

❑ *Having* clause

- ❑ Find the names and average salaries of all departments whose average salary is greater than 42,000
 - ❑ `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
- ❑ Note: Aggregate functions cannot be used in the **where** clause

❑ General syntax

- ❑ `select column from table where condition group by column having condition order by column1, column2..;`



Exercise

idx	type	name
1	1	안중근
2	1	윤봉길
3	2	김유신
4	2	이순신
5	3	이성계
6	3	왕건
7	4	반갑수

- ❑ type으로 그룹화 해서 데이터 개수 조회

- ❑ `SELECT type, COUNT(name) AS cnt FROM hero_collection GROUP BY type;`

- ❑ type이 2 이상인 경우에만 그룹화 해서 데이터 개수 조회

- ❑ `select name, type group by type where type >= 2`

- ❑ type 그룹화하여 데이터 개수 출력 후, 그룹내 데이터 개수가 2개 이상인 데이터 조회

- ❑ `SELECT type, COUNT(name) AS cnt FROM hero_collection GROUP BY type HAVING cnt >= 2;`

- ❑ type 2 이상인 사람들을 그룹화하여 그룹내 데이터 개수를 가져온 후, 그룹내 데이터 개수가 2개 이상인 데이터 조회

- ❑ `SELECT type, COUNT(name) AS cnt FROM hero_collection WHERE type > 1 GROUP BY type HAVING cnt >= 2;`

- ❑ type 2 이상인 사람들을 그룹화하여 그룹내 데이터 개수를 가져온 후, 그 중에 수가 2개 이상인 데이터를 type 내림차순 정렬로 조회

- ❑ `SELECT type, COUNT(name) AS cnt FROM hero_collection WHERE type > 1 GROUP BY type HAVING cnt >= 2 ORDER BY type DESC;`



Exercise

idx	type	name
1	1	안중근
2	1	윤봉길
3	2	김유신
4	2	이순신
5	3	이성계
6	3	왕건
7	4	반갑수

- ❑ type으로 그룹화 해서 데이터 개수 조회
 - ❑ `SELECT type, COUNT(name) AS cnt FROM hero_collection GROUP BY type;`
- ❑ type이 2 이상인 경우에만 그룹화 해서 데이터 개수 조회
 - ❑ `SELECT type, COUNT(name) AS cnt FROM hero_collection WHERE type > 1 GROUP BY type;`
- ❑ type 그룹화하여 데이터 개수 출력 후, 그룹내 데이터 개수가 2개 이상인 데이터 조회
 - ❑ `SELECT type, COUNT(name) AS cnt FROM hero_collection GROUP BY type HAVING cnt >= 2;`
- ❑ type 2 이상인 사람들을 그룹화하여 그룹내 데이터 개수를 가져온 후, 그룹내 데이터 개수가 2개 이상인 데이터 조회
 - ❑ `SELECT type, COUNT(name) AS cnt FROM hero_collection WHERE type > 1 GROUP BY type HAVING cnt >= 2;`
- ❑ type 2 이상인 사람들을 그룹화하여 그룹내 데이터 개수를 가져온 후, 그 중에 수가 2개 이상인 데이터를 type 내림차순 정렬로 조회
 - ❑ `SELECT type, COUNT(name) AS cnt FROM hero_collection WHERE type > 1 GROUP BY type HAVING cnt >= 2 ORDER BY type DESC;`



Aggregate Functions



- ❑ Null values and aggregates
 - ❑ Total all salaries
 - ❑ `select sum(salary) from instructor;`
 - ❑ Above statement ignores null amounts
 - ❑ Result is null if there is null amount
 - ❑ All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes



Assignment #3 (150pt)



☐ Do Exercises

- ☐ 3.11, 3.13, 3.14, 3.21, 3.25, 3.26, 3.28
- ☐ 데이터가 없는 경우 임의로 넣으세요.
- ☐ Self Study : Chapter 3.8

☐ Due: Two Weeks Later

- ☐ Before the lecture – 9/28 (Wed)

☐ Submission form

- ☐ *.doc or *.sql both okay

☐ Method: upload your report in Cyber Campus

- ☐ Questions are uploaded in Assignment 3 folder
- ☐ Answers must be written in English !