UMBC

AN HONORS UNIVERSITY IN MARYLAND

CMSC 461, Database Management Systems
Spring 2018

Lecture 6 - More SQL

These slides are based on “Database System Concepts” book and slides, 6th edition, and the 2009/2012 CMSC 461 slides by Dr. Kalpakis

Dr. Jennifer Sleeman

Logistics

- Project Phase 1 due Thursday 2/15/2018
- Homework 2 due on 2/26/2018

Today we will wrap up the SQL discussion today

Lecture Outline

- ***Additional Operations***
- Set Operations
- Aggregate Functions
- Nested Queries
- Modification of the database
- Joins
- Data Types

Rename Operation

The SQL allows renaming relations and attributes using the as clause:

- old-name as new-name

Example:

select ID, name, salary/12 ***as monthly_salary***
from instructor

Rename Operation

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

Rename Operation

select distinct ***T.****name*
from instructor ***as T,*** instructor ***as S***
where ***T.****salary* > ***S.****salary and*
S.*dept_name* = *'Comp. Sci.';*

Also known as table alias, correlation variable or tuple variable

Why Rename?

- Relations in from clause may have attributes with same attribute name
- If an arithmetic expression used, resulting attribute no name
- May want to change attribute name

String Operation

- 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.
- Patterns are case sensitive

String Operation

- Enclosed by single quotes
  - Case sensitive

    'comp. Sci.' ='Comp. Sci.' is false
- Concatenation ||
- Extraction of substring
- Length of string
- Convert to upper or lower case
- Removal of white space (trim(s))

String Operation

- 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.

String Operation

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 '\'

Ordering Display of Tuples

- List in alphabetical order the names of all instructors

```
select distinct name
from instructor
order by name;
```

Ordering Display of Tuples

- 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***

Ordering Display of Tuples

- Can sort on multiple attributes
  - Example: ***order by*** dept_name, name
- Can order by multiple attributes specifying desc/asc order for each

```
select * from instructor
order by salary desc, name asc;
```

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, ≥ $90,000 and ≤ $100,000)

```
select name
from instructor
where salary between 90000 and 100000
```

Where Clause Predicates

```
select name
from instructor
where salary between 90000 and 100000
```

INSTEAD OF…

```
select name
from instructor
where salary >= 90000 and salary <=
100000
```

Where Clause Predicates

- Tuple comparison $(v_1, v_2, \ldots v_n)$ denotes a tuple of arity n
- Comparison operators
  - $(a_1, a_2) <= (b_1, b_2)$ is true if $a_1 <= b_1$ and $a_2 <= b_2$

```
select name, course_id
from instructor, teaches
where (instructor.ID, dept_name) =
(teaches.ID, 'Biology');
```

Lecture Outline

- Additional Operations
- ***Set Operations***
- Aggregate Functions
- Nested Queries
- Modification of the database
- Joins
- Data Types

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***.

Set Operations

```
select dept_name from instructor_L5 where dept_name='Finance'
union all select dept_name from instructor_L5 where
dept_name='Computer Science';
```

```
+-----------------+
| dept_name       |
+-----------------+
| Finance         |
| Computer Science |
| Computer Science |
+-----------------+
```

```
select dept_name from instructor_L5 where dept_name='Finance'
union select dept_name from instructor_L5 where
dept_name='Computer Science';
```

```
+-----------------+
| dept_name       |
+-----------------+
| Finance         |
| Computer Science |
+-----------------+
```

Set Operations

Find courses that ran in Fall 2009 or in Spring 2010

Set Operations

(**select** *course_id* **from** *section* **where** *sem* = ‘Fall’ **and** *year* = 2009)
union
(**select** *course_id* **from** *section* **where** *sem* = ‘Spring’ **and** *year* = 2010);

Set Operations

Find courses that ran in Fall 2009 and in Spring 2010

Set Operations

(**select** *course_id* **from** *section* **where** *sem* = ‘Fall’ **and** *year* = 2009)
intersect
(**select** *course_id* **from** *section* **where** *sem* = ‘Spring’ **and** *year* = 2010);

Set Operations

Find courses that ran in Fall 2009 but not in Spring 2010

Set Operations

(**select** *course_id* **from** *section* **where** *sem* = ‘Fall’ **and** *year* = 2009)

except

(**select** *course_id* **from** *section* **where** *sem* = ‘Spring’ **and** *year* = 2010);

Recall 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

Recall Null Values

- 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;
```

Lecture Outline

- Review
- Finish In-Class Exercise
- Additional Operations
- Set Operations
- ***Aggregate Functions***
- Nested Queries
- Modification of the database
- Joins
- Data Types

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 - Average

Find the average salary of instructors in the Computer Science department

Aggregate Functions - Average

```
select avg (salary)
from instructor
where dept_name= ’Comp. Sci.’;
```

Aggregate Functions - Average

```
select avg (salary)
from instructor
where dept_name= ’Comp. Sci.’;
```

What is going to be the name of the attribute returned?

Aggregate Functions - Average

```
select avg (salary) as avg_salary
from instructor
where dept_name= ’Comp. Sci.’;
```

Aggregate Functions - Count

Find the total number of instructors who teach a course in the Spring 2010 semester

Aggregate Functions - Count

Why use distinct?

```
select count (distinct ID)
from teaches
where semester = 'Spring' and
year = 2010
```

Aggregate Functions - Count

Find the number of tuples in the course relation

Aggregate Functions - Count

```
select count (*) from course;
```

Aggregate Functions – Group By

Find the average salary of instructors in each department

Aggregate Functions – Group By

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

Aggregate Functions – Group By

Find the number of instructors in each department who teach a course in the Spring 2010 semester

Aggregate Functions – Group By

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

Instructor

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

Teaches

Aggregate Functions – Group By

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

Instructor

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

Teaches

Find the number of instructors in each department who teach a course in the Spring 2010 semester

Aggregate Functions – Group By

What are we grouping by?

Aggregate Functions – Group By

```
from instructor natural join teaches where
semester='Spring' and year=2010
group by dept_name;
```

Aggregate Functions – Group By

What are we counting?

Aggregate Functions – Group By

```
select dept_name, count(distinct ID) as
instr_count from instructor natural join
teaches where semester='Spring' and
year=2010 group by dept_name;
```

Aggregate Functions – Group By

```
mysql> select dept_name, count(distinct ID) as instr_count from instructor  natural join teaches
where semester='Spring' and year=2010 group by dept_name;
+------------+-------------+
| dept_name  | instr_count |
+------------+-------------+
| Comp. Sci. |           3 |
| Finance    |           1 |
| History    |           1 |
| Music      |           1 |
+------------+-------------+
4 rows in set (0.00 sec)

mysql> select dept_name, count(ID) as instr_count from instructor  natural join teaches where
semester='Spring' and year=2010 group by dept_name;
+------------+-------------+
| dept_name  | instr_count |
+------------+-------------+
| Comp. Sci. |           4 |
| Finance    |           1 |
| History    |           1 |
| Music      |           1 |
+------------+-------------+
4 rows in set (0.00 sec)
```

Can I do this? (MySQL)

```
mysql> select dept_name, ID, avg (salary)
    -> from instructor
    -> group by dept_name;
+------------+-------+--------------+
| dept_name  | ID    | avg (salary) |
+------------+-------+--------------+
| Biology    | 76766 | 72000.000000 |
| Comp. Sci. | 10101 | 77333.333333 |
| Elec. Eng. | 98345 | 80000.000000 |
| Finance    | 12121 | 85000.000000 |
| History    | 32343 | 61000.000000 |
| Music      | 15151 | 40000.000000 |
| Physics    | 22222 | 91000.000000 |
+------------+-------+--------------+
7 rows in set (0.00 sec)
```

Aggregate Functions – Erroneous Query

Attributes in select clause outside of aggregate functions must appear in group by list

```
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

Aggregate Functions – Having Clause

- Use the having clause to state a condition that applies to groups constructed by the group by clause rather than single tuples
- Predicates in having clause are applied after the formation of groups whereas predicates in the where clause are applied before forming groups

Aggregate Functions – Having Clause

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

Aggregate Functions – Having Clause

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

VALID

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

VALID

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

INVALID

WARNING: THESE ARE NOT THE SAME!

Null Values and Aggregate Functions

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

Lecture Outline

- Review
- Finish In-Class Exercise
- Additional Operations
- Set Operations
- Aggregate Functions
- ***Nested Queries***
- Modification of the database
- Joins
- Data Types

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.

Set Membership

Find courses offered in Fall 2009 and in Spring 2010

Set Membership

```
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);
```

Set Membership

Find courses offered in Fall 2009 but not in Spring 2010

Set Membership

```
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);
```

Set Membership

Find the total number of (distinct) students who have taken course sections taught by the instructor with ID 10101

Set Membership

```
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);
```

Set Comparison

Nested subqueries can be used to compare sets.

Correlation Variables

- **Correlated subquery –** uses a correlation name from an outer query
- **Correlation name** or **correlation variable** – variables from outer query that are used in nested subquery

Subqueries in 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;
```

Scalar subqueries

- Scalar subquery is one which is used where a single value is expected
- Runtime error if subquery returns more than one result tuple

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

Scalar subqueries

```
select name
        from instructor
        where  salary * 10 >
            (select budget  from department
              where department.dept_name =
instructor.dept_name)
```

Lecture Outline

- Review
- Finish In-Class Exercise
- Additional Operations
- Set Operations
- Aggregate Functions
- Nested Queries
- ***Modification of the database***
- Joins
- Data Types

Modifications 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

Deletions

- Expressed similarly to queries
- Delete whole tuples

delete *from* *r* *where* *P;*

- *P* is the predicate
- *r* is the relation
- First finds all tuples *t* in *r* where *P(t)* is true
- Then deletes them from *r*

Deletions

Delete all instructors

delete from instructor

Delete all instructors from the Finance department

delete from instructor
where dept_name= ’Finance’;

Deletions

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');

Can I do this?

*delete * from* instructor;

ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '* from instructor' at line 1

Can I do this?

delete ID from instructor;

ERROR 1109 (42S02): Unknown table 'ID' in MULTI DELETE

Can I do this?

delete from instructor, courses
where dept_name= ’Finance’;

ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'where dept_name= ’Finance’' at line 2

Deletions – What is wrong with this statement?

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)

Deletions – What is wrong with this statement?

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

delete from instructor
where salary< (select avg (salary) from instructor);

Insertions

- To insert:
  - Specify a tuple to be inserted
  - Use a set of tuples that results from a query
- Attribute values must be members of attribute's domain
- Tuples inserted must have correct number of attributes

Insertions

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);

Insertions

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.

Updates

- To change a value in a tuple without changing all values in the tuple
- Use update statement
  - Alternative is to delete tuple and insert with new value

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

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);
```

Lecture Outline

- Review
- Finish In-Class Exercise
- Additional Operations
- Set Operations
- Aggregate Functions
- Nested Queries
- Modification of the database
- ***Joins***
- Data Types

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

Joined Relations

What types of joins have we seen so far?

Joined Relations

Cartesian with where clause

Select *name, course_id*
from *instructor, teaches*
where *instructor.ID = teaches.ID;*

Joined Relations

Natural Join

Select name, course_id
*from instructor **natural join** teaches;*

Joined Relations

- There is also ***join*** with ***using*** clause
  Select name, course_id
  *from instructor **join** teaches **using** (ID);*
- You must specify list of attributes to join upon
- Both relations must have the same name
- Similar to natural join except:
  - Not all attributes that are the same are joined upon

Joined Relations

- There is also ***join*** with ***on*** condition
  - *Select name, course_id*
  - *from instructor **join** teaches **on** (instructor.ID = teaches.ID);*
- Arbitrary join condition
- Similar to using ***where*** clause to specify join condition
  - The ***on*** condition behaves differently for outer joins

Join Example

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

course

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

prereq

What happens when we join on these two tables?

Join Example

```
select * from course, prereq where
course.course_id = prereq.course_id;
```

```
+----------+------------+-----------+--------+----------+----------+
| course_id | title      | dept_name  | credits | course_id | prereq_id |
+----------+------------+-----------+--------+----------+----------+
| BIO-301   | Genetics   | Biology    |       4 | BIO-301   | BIO-101   |
| CS-190    | Game Design | Comp. Sci. |       4 | CS-190    | CS-101    |
+----------+------------+-----------+--------+----------+----------+
2 rows in set (0.00 sec)
```

Join Example

```
select * from course natural
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    |
+----------+------------+-----------+--------+----------+
2 rows in set (0.00 sec)
```

Join Example

```
select * from course 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    |
+----------+------------+-----------+--------+-----------+
2 rows in set (0.00 sec)
```

Join Example

```
select * from course join prereq on
course.course_id = prereq.course_id;
```

```
+----------+------------+-----------+--------+----------+----------+
| course_id | title      | dept_name  | credits | course_id | prereq_id |
+----------+------------+-----------+--------+----------+----------+
| BIO-301   | Genetics    | Biology    |       4 | BIO-301   | BIO-101   |
| CS-190    | Game Design | Comp. Sci. |       4 | CS-190    | CS-101    |
+----------+------------+-----------+--------+----------+----------+
2 rows in set (0.01 sec)
```

Outer Joins

- 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.
- ***inner join*** – join operations that do not preserve non-matched tuples

Left Outer Join

select * from 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      |
+----------+------------+-----------+--------+----------+
3 rows in set (0.00 sec)
```

Right Outer Join

select * from course ***natural right outer join*** prereq;

```
+----------+----------+------------+-----------+---------+
| course_id | prereq_id | title      | dept_name  | credits |
+----------+----------+------------+-----------+---------+
| BIO-301   | BIO-101   | Genetics   | Biology    |       4 |
| CS-190    | CS-101    | Game Design | Comp. Sci. |       4 |
| CS-347    | CS-101    | NULL       | NULL       |    NULL |
+----------+----------+------------+-----------+---------+
3 rows in set (0.00 sec)
```

Full Outer Join

select * from 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     |
| BIO-301  | BIO-101    | Genetics   | Biology   | 4        |
| CS-190   | CS-101     | Game Design | Comp. Sci. | 4       |
| CS-347   | CS-101     | NULL       | NULL      | NULL     |
+----------+------------+------------+-----------+----------+
6 rows in set (0.00 sec)
```

Full Outer Join in MySQL Alternative

select * from course natural ***left outer join*** prereq

union

select * from course natural ***right outer join*** prereq;

Join Types and Conditions

- ***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 Type
Inner join
Left outer join
Right outer join
Full outer join

Join Conditions
natural
on <predicate>
Using (A_1, A_2,....A_n)

Join Types and Conditions

select * from course ***right outer join*** prereq ***on*** course.course_id=prereq.course_id;

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

select * from course ***right outer join*** prereq ***using*** (course_id);

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

Join Types and Conditions

Select * from course ***inner join*** prereq ***on*** course.course_id = prereq.course_id;

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

Select * from course ***natural 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    |
+----------+------------+-----------+--------+----------+
```

Lecture Outline

- Review
- Finish In-Class Exercise
- Additional Operations
- Set Operations
- Aggregate Functions
- Nested Queries
- Modification of the database
- Joins
- ***Data Types***

Date and Time Data Types

- ***date***: Dates, containing a (4 digit) year, month and date
  - Example: date ‘2005-7-27’
- ***time***: Time of day, in hours, minutes and seconds.
  - Example: time ‘09:00:30’ , time ‘09:00:30.75’

Date and Time Data Types

- ***timestamp***: date plus time of day
  - Example: timestamp '2005-7-27 09:00:30.75'
- ***interval***: period of time
  - Example: interval '1' day
  - Subtracting a date/time/timestamp value from another gives an interval value
  - Interval values can be added to date/time/timestamp values

Default Types

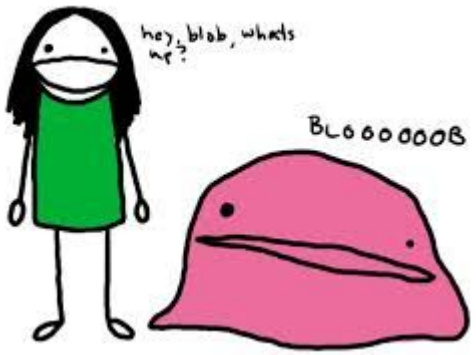
- You may specify a default type for an attribute

Example:

```
create table student
(ID              varchar(5),
name             varchar(20) not null,
dept_name        varchar(20),
tot_cred         numeric(3,0) default 0,
primary key(ID));
```

Large Object Types

- Large objects (photos, videos, CAD files, etc.) are stored as a large object:
  - blob: binary large object -- object is a large collection of uninterpreted binary data (whose interpretation is left to an application outside of the database system)



Large Object Types

- clob: character large object -- object is a large collection of character data

When a query returns a large object, a pointer is returned rather than the large object itself.

User Defined Types

create type construct in SQL creates user-defined type

create type Dollars as numeric (12,2) final

create table department
(dept_name varchar (20),
building varchar (15),
budget Dollars);

Coming Next Week NoSQL