CS157A: Introduction to Database Management Systems

Chapter 6: The Database Language SQL-Part II

Suneuy Kim

Join Variants

- CROSS JOIN → Cartesian product
- (INNER) JOIN ON → Theta Join
- NATURAL JOIN → Natural Join
- LEFT|RIGHT|FULL OUTER JOIN ON

  : augment the result of a join by the dangling tuples

MySQL Join Variants

- In MySQL, Join, Cross Join, Inner Join are the same, working as theta join.

```
select uName, title, age
from User  INNER JOIN Loan ON User.uID = Loan.uID
and age < 20;
```

- Replacing INNER JOIN with Cross Join or Join will not change the result.

MySQL: Inner Join, Cross Join, and Join (Alternative Syntaxes)

`using(a1, a2, …)` Clause

- This is similar to on, but the name of the join attribute(s) must be the same in each table.
- The join attribute(s) only appears *once* in the result set.

```
select uName, title
 from User INNER JOIN Loan using(uID);
```

On or Where?

```
select uName, age
from User join Loan on User.uID=Loan.uID
where loaned > 3 and title = 'Bambi';
```

```
select uName, age
from User join Loan  on User.uID=Loan.uID  and
                  loaned > 3 and title = 'Bambi';
```

On or Where?

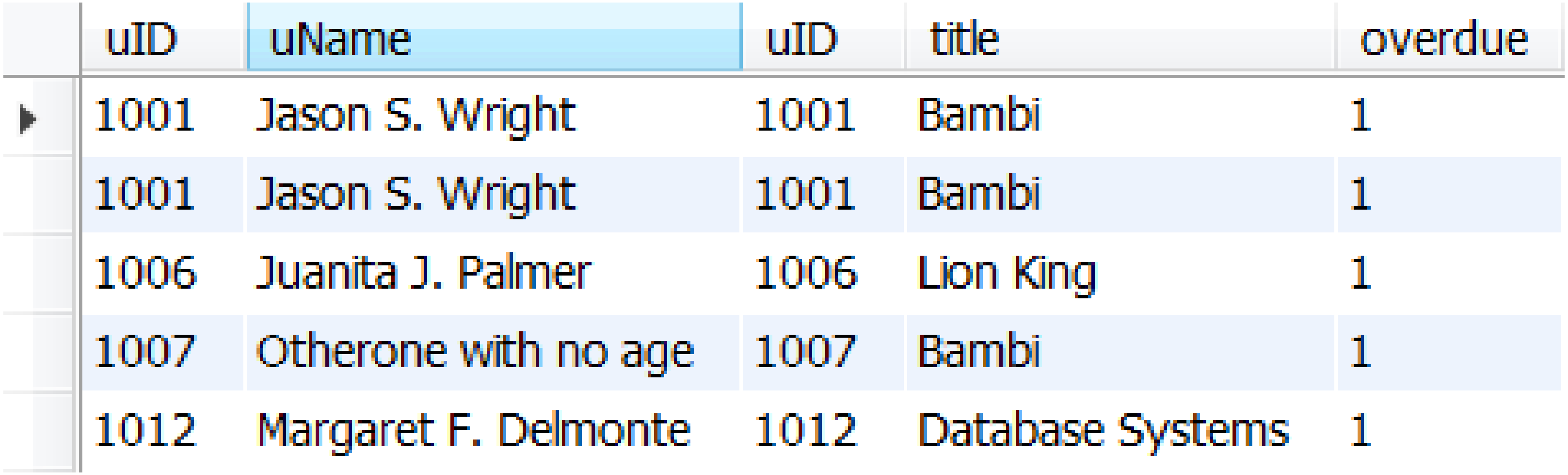
- With an **Inner Join**, the clauses are *effectively* equivalent.
- With an **Outer Join**, they are not the same.

```
select * from User left outer join Loan on
User.uID = Loan.uID where Loan.overdue = true ;
```

vs.

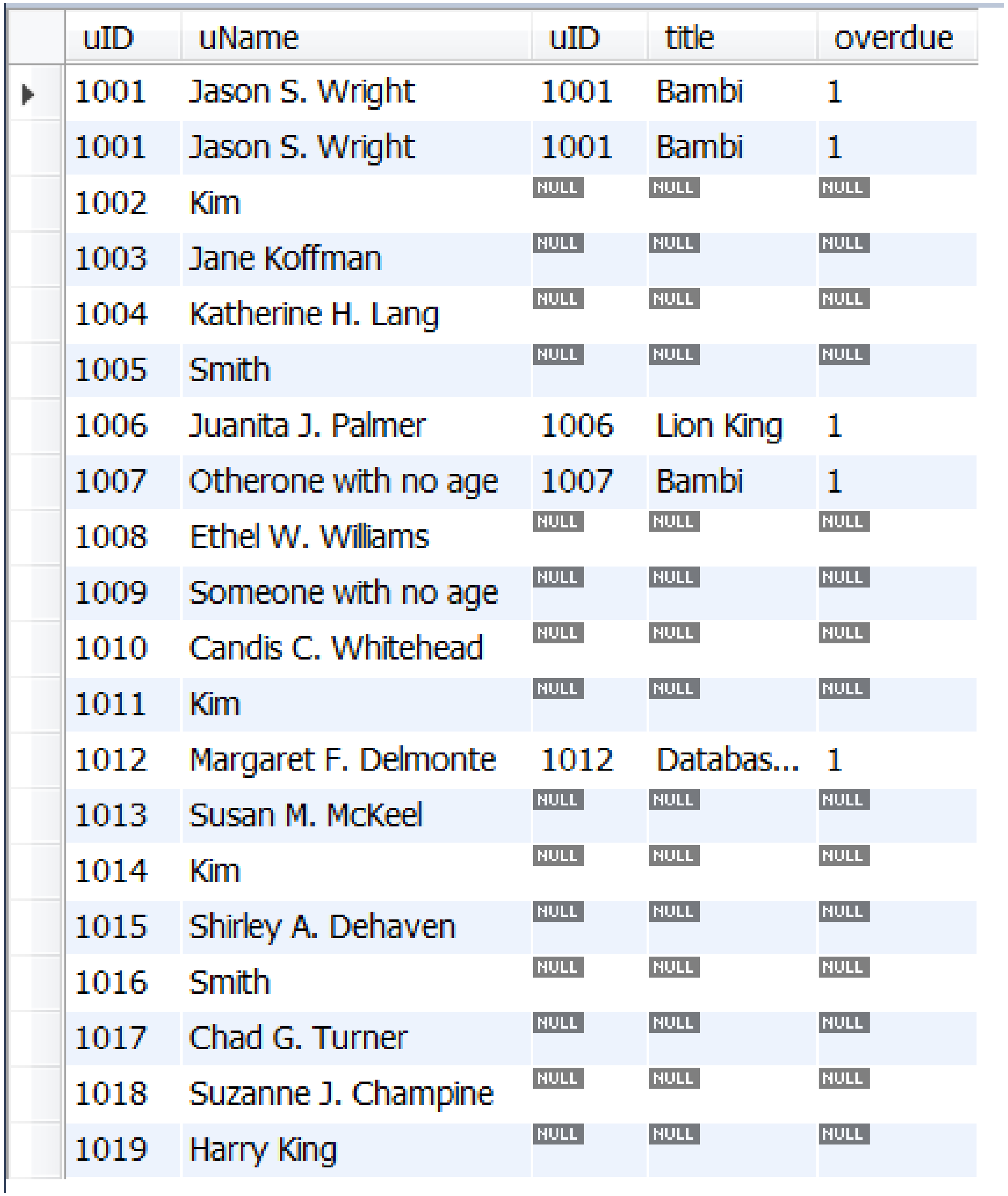
```
select * from User left outer join Loan
on User.uID = Loan.uID and Loan.overdue = true;
```

```
select User.uID, uName, Loan.uID, title, overdue
from User left join Loan on User.uID = Loan.uID
where Loan.overdue = true;
```



uID	uName	uID	title	overdue
1001	Jason S. Wright	1001	Bambi	1
1001	Jason S. Wright	1001	Bambi	1
1006	Juanita J. Palmer	1006	Lion King	1
1007	Otherone with no age	1007	Bambi	1
1012	Margaret F. Delmonte	1012	Database Systems	1

```
select User.uID, uName,
Loan.uID, title, overdue
from User left join Loan
on User.uID = Loan.uID
and Loan.overdue = true;
```



uID	uName	uID	title	overdue
1001	Jason S. Wright	1001	Bambi	1
1001	Jason S. Wright	1001	Bambi	1
1002	Kim	NULL	NULL	NULL
1003	Jane Koffman	NULL	NULL	NULL
1004	Katherine H. Lang	NULL	NULL	NULL
1005	Smith	NULL	NULL	NULL
1006	Juanita J. Palmer	1006	Lion King	1
1007	Otherone with no age	1007	Bambi	1
1008	Ethel W. Williams	NULL	NULL	NULL
1009	Someone with no age	NULL	NULL	NULL
1010	Candis C. Whitehead	NULL	NULL	NULL
1011	Kim	NULL	NULL	NULL
1012	Margaret F. Delmonte	1012	Databas...	1
1013	Susan M. McKeel	NULL	NULL	NULL
1014	Kim	NULL	NULL	NULL
1015	Shirley A. Dehaven	NULL	NULL	NULL
1016	Smith	NULL	NULL	NULL
1017	Chad G. Turner	NULL	NULL	NULL
1018	Suzanne J. Champine	NULL	NULL	NULL
1019	Harry King	NULL	NULL	NULL

Join using and on together

Most of the system doesn't allow join-using-on together.

```
select U1.uID, U1.uName, U1.age, U2.uID,
U2.uName, U2.age
from User U1 join User U2 using (age)
on U1.uID < U2.uID;
```

How to fix it ?

```
select U1.uID, U1.uName, U1.age, U2.uID,
U2.uName, U2.age
from User U1 join User U2 using (age)
where U1.uID < U2.uID;
```

or

```
select U1.uID, U1.uName, U1.age, U2.uID,
U2.uName, U2.age
from User U1 join User U2 on U1.age = U2.age and
U1.uID < U2.uID;
```

Changing three way join to binary join

```
select *
from Loan join User join Book
on Loan.uID = User.uID and Loan.title = Book.title ;
```

→

```
select *
from (Loan join User on Loan.uID = User.uID)
          join Book on Loan.title = Book.title;
```

Natural Join

// Suppose uID is the only common attribute in // User and Loan

```
select distinct uName, title
from User natural join Loan;
```

→

```
select distinct uName, title
from User join Loan
on User.uID = Loan.uID;
```

Note: select * will return relations with a different schema.

Outer Join

- Dangling tuple

  A tuple that fails to join with any tuple of the other relation.

- Outer join augments the result of join by the dangling tuples, padded with NULL.

Outer Join

- LEFT|RIGHT|FULL OUTER JOIN pads dangling tuples from LEFT, RIGHT, or BOTH.
- Theta Join

  R LEFT|RIGHT|FULL OUTER JOIN S ON <condition>
- Natural Join

  R NATURAL LEFT|RIGHT|FULL OUTER JOIN S

R

a	b
10	20
50	5

S

c	d
7	40
10	5
30	8

R LEFT|RIGHT|FULL OUTER JOIN S ON b > c

LEFT

a	b	c	d
10	20	7	40
10	20	10	5
50	5	N	N

RIGHT

a	b	c	d
10	20	7	40
10	20	10	5
N	N	30	8

FULL

a	b	c	d
10	20	7	40
10	20	10	5
50	5	N	N
N	N	30	8

Rewriting left outer join

```
select uName, User.uID, title, overdue
from User, Loan
where User.uID = Loan.uID
union all
select uName, uID, NULL, NULL
from User
where uID not in (select uID from Loan);
```

right outer join

```
select uID, uName, title, overdue
from User right outer join Loan using (uID);
```

full outer join

```
select uID, uName, title, overdue
from User full outer join Loan using (uID);
```

[Q] MySQL does not support full outer join. Can you rewrite full outer join without using join ?

full outer join without using join

```
select uName, User.uID, title, overdue
from User, Loan
where User.uID = Loan.uID
union all
select uName, uID, NULL, NULL
from User
where uID not in (select uID from Loan)
union all
select NULL, uID, title, overdue
from Loan
where uID not in (select uID from User);
```

- Commutativity : (A op B) = (B op A)
- Associativity: (A op B) op C = A op (B op C)

Left|Right outer joins are not commutative.

Full outer join is commutative.

Left|Right|Full outer joins are not associative.

It is important to think of the order of ( ).

Example: Violation of Associativity

- SELECT * FROM (A LEFT OUTER JOIN B ON A.b_id = B.id) LEFT OUTER JOIN C ON B.id IS NULL;
- SELECT * FROM A LEFT OUTER JOIN (B LEFT OUTER JOIN C ON B.id IS NULL) ON A.b_id = B.id;
- See this in action

http://sqlfiddle.com/#!2/0d462/3

MySQL doesn’t support except (or minus) and intersect

A

x	y
1	a
2	b
3	c
4	d

B

x	y
1	a
3	c

```
DROP TABLE IF EXISTS A;
CREATE TABLE A (x INT, y VARCHAR(5));
DROP TABLE IF EXISTS B;
CREATE TABLE B (x INT, y VARCHAR(5));
INSERT INTO A(x,y) VALUES(1,'a');
INSERT INTO A(x,y) VALUES(2,'b');
INSERT INTO A(x,y) VALUES(3,'c');
INSERT INTO A(x,y) VALUES(4,'d');
INSERT INTO B(x,y) VALUES(1,'a');
INSERT INTO B(x,y) VALUES(3,'c');
```

Difference in MySQL

A

x	y
1	a
2	b
3	c
4	d

B

x	y
1	a
3	c

```
SELECT * FROM A
WHERE (x,y) NOT IN (SELECT *
FROM B);
```

```
SELECT * FROM A
WHERE NOT EXISTS
(SELECT * FROM B WHERE B.x =
A.x AND B.y = A.y);
```

A - B

x	y
2	b
4	d

```
SELECT DISTINCT A.x AS x, A.y AS y
FROM A LEFT OUTER JOIN B USING (x,
y) WHERE B.x IS NULL;
```

Intersection in MySQL

A

x	y
1	a
2	b
3	c
4	d

B

x	y
1	a
3	c

```
SELECT * FROM A WHERE (x,y)
IN (SELECT * FROM B);
```

```
SELECT * FROM A WHERE EXISTS
(SELECT * FROM B
 WHERE B.x=A.x AND B.y =A.y);
```

A ∩B

x	y
1	a
3	c

```
SELECT DISTINCT A.x AS x, A.y AS y
FROM A INNER JOIN B USING (x,y);
```

Aggregation

- min, max, sum, avg, count

select A1, A2, …, An ← aggregation appears here.

from R1, R2, …, Rn

where ← apply to the single tuple at a time

group by

having ← filter the group

```
select avg(age)
from User;

select min(age)
from User, Loan
where User.uID = Loan.uID and title = 'Bambi';

select count(*)
from Loan
where title = 'Bambi';
```

```
select avg(age)
from User, Loan
where User.uID = Loan.uID and title = 'Bambi';
```

VS

```
select avg(age)
from User
where uID in
(select  uID   from Loan  where title = 'Bambi');
```

Eliminating Duplicates in an Aggregation

```
select count(distinct uID)
from Loan
where title = 'Bambi';
```

group by

- We may follow a SELECT-FROM-WHERE expression by GROUP BY and a list of grouping attributes.
- The relation that results from the FROM-WHERE is grouped according to the values of all those attributes, and any aggregation in SELECT is applied only within each group.

group by

When there is an aggregation in SELECT clause, there are only two types of terms the SELECT clause can have:

1. Aggregations - these terms are evaluated for each group.
2. Grouping attributes can be unaggregated.

Example: group by

```
select title, count(*)
from Loan
group by title;
```

group by

```
select title, min(age),  max(age)
from User, Loan
where User.uID = Loan.uID
group by title;
```

group by

[Q] To find the Largest span of ages of users who borrowed the same book

```
select max(mx-mn)
from
   (select title, min(age) as mn, max(age) as mx
    from User, Loan
    where User.uID = Loan.uID
    group by title) ST;
```

group by

[Q] To find the number of **different** books a user loaned

```
select User.uID, uName, count(distinct title)
from  User, Loan
where User.uID = Loan.uID
group by User.uID;
```

Does it work ?

```
select User.uID, uName, count(distinct title), title
from  User, Loan
where User.uID = Loan.uID
group by User.uID
```

[A] There maybe different titles per group. Then system chooses a random title among the titles the user loaned (e.g. MySql) or generates error (e.g. Postgres).

Quiz

Number of books loaned by each user. If a user did not loan any book, show 0 for the number of loaned books.

```
select User.uID, uName, count(distinct title)
from  User, Loan
where User.uID = Loan.uID
group by User.uID
union
select User.uID, uName, 0
from User
where uID not in (select uID from Loan);
```

HAVING Clauses

- HAVING <condition> may follow a GROUP BY clause.
- If so, the condition applies to each group, and groups not satisfying the condition are eliminated.

Requirements on HAVING Conditions

1. An aggregation in a HAVING clause applies only to the group being tested.

```
select loaned, count(*)
from User
group by loaned
having count(*) >2
```

Tests if the current group has more than 2 counts

Requirements on HAVING Conditions

2. Any attributes of relations in the FROM clause may be aggregated in the HAVING clause, but only grouping attributes may appear unaggregated in the HAVING clause.

```
select loaned, count(*), avg(age)
from User
group by loaned
having  avg(age) > 40 and count(*) >=3 ;
```

Example: Violation

```
select loaned, count(*), age
from User
group by loaned
having age > 40 and count(*) >=3 ;
```

In MySQL, an age is randomly chosen within each group and returned

MySQL generates an error.

Having

[Q] To find books loaned at least three times

```
select title
from Loan
group by title
having count(*) >=3;
```

[Q] To find a book loaned by at least three different users.

```
select title
from Loan
group by title
having count(distinct uID) >= 3;
```

Without using group by and having

To find books with fewer than 3 borrowers

```
select title
from Loan
group by title
having count(*) < 3;
```

```
select distinct title
from Loan L1
where
( select count(*)
 from Loan L2
  where L2.title = L1.title) < 3;
```

[Q] To find books whose loaner’s maximum age is below the average age of users

```
select title
from User, Loan
where User.uID = Loan.uID
group by title
having max(age) <(select avg(age) from User);
```

Null values and Aggregation

- NULL is ignored in any aggregation except for count(*)

  select sum(age) from User: null is ignored

  count(*) counts all tuples including null

  count(age) counts non-null ages.

  count(distinct age) counts non-null unique ages.
- NULL is treated as an ordinary when forming groups.

  e.g.) select age, count(*) from user group by age;
- Any aggregation except count over an empty bag of values returns NULL. The count of an empty bag is 0.

Example: Null values and Aggregation

```
1. select count(*)
   from User
   where age is not null;
```

```
2. select count(distict  age)
   from User
   where age is not null;
```

```
3. select count(distinct  age)
   from User
```

```
4. select distinct  age
   from User
```

Notes:

2 and 3 return the same result.

4 returns distinct ages including null.

Data Modification

- `insert into` R `values (V1, V2,.., Vn)`
- `insert into` R `select` statement
- `delete from` R `where` condition
- `update` R

  `set` attribute = expression

  `where` condition
- `update` R

  `set` A1 = expr1, A2 = expr2, …, An = exprn

  `where` condition

Insert

- To insert a single tuple:

  INSERT INTO R (A1, …, An)

  VALUES (v1, …, vn);

- We may add to the relation name a list of attributes. Two reasons to do so:

1. We forget the standard order of attributes for the relation.
2. We don’t have values for all attributes, and we want the system to fill in missing components with NULL or a default value.

Insert

```
insert into Book(title,author,copies)
values('This Book', 'That Author', 40);
```

=

```
insert into Book values
('This Book', 'That Author', 40);

insert into Book(title, author) values
('This Book', 'That Author');
```

→ The system will initialize the value of copies to null.

```
Error: insert into Book(title,author,copies)
        values('This Book', 'That Author');
Error: insert into Book
        values ('This Book', 'That Author');
```

Insert

User table defines auto-incremented field (uId) and on-updated field (updatedon)

```
insert into user (uname, age, loaned,
updatedon) values('John Smith', 23,4, null);
```

:assigns an auto-incremented id to this tuple and updatedon value will set to the current time stamp.

Note: the sequence is reset.

Insert

```
insert into user (uid, uname, age,
loaned, updatedon) values(2000, 'John
Smith', 23,4, null);
```

Note: When you insert any other value into a AUTO_INCREMENT column, the column is set to the value and the sequence is reset so that the next auto generated value follows sequentially from the inserted value.

Inserting Many Tuples

- We may insert the entire result of a query into a relation, using the form:

    INSERT INTO <relation>

    (<subquery>);

```
insert into Loan
(
 select uID,‘Let’s Read!','0000-00-00',false
 from User
 where uID not in (select uID from Loan)
);
```

[Q] To have users, who loaned Bambi and not being overdue, loan ‘Bambi II’.

```
insert Loan
(select uid, 'Bambi II', UTC_DATE(), false
from USER
where uid in
   (select uid  from Loan  where title='Bambi'
     and overdue = false));
```

Delete

- To delete tuples satisfying a condition from some relation:

```
DELETE FROM <relation>
WHERE <condition>;
```

Delete

[Q] To delete a user who borrowed the same books

```
delete from User
where  uID in
(select uID
 from Loan
 group by uID
 having count(title) <> count(distinct title))  ;
```

Delete – Error

```
delete
from Loan
where  uID in
( select uID
  from Loan
  group by uID
  having count(title) <>
count(distinct title)) ;
```

Note: You can’t specify the relation for update in From clause.

Update

- To change certain attributes in certain tuples of a relation:

```
UPDATE <relation>
SET <list of attribute assignments>
WHERE <condition on tuples>;
```

Update

[Q] To find a user with age < 15, and turn their overdue to false.

```
update Loan
set overdue = false
where overdue = true and
          uID in  (select  uID   from user    where age
< 15);
```

Update - Error

[Q] To find the oldest user(s) who borrowed ‘Bambi’ and if the loan is being overdue, set it to false.

```
update loan
set overdue = false
where loan.title = 'Bambi' and
      overdue = true and uid in
     (select uid from user where age =
            (select max(age)
             from user natural join loan
             group by title
             having title = 'Bambi'));
Error: Can’t specify the target loan for
update in FROM clause
```

```
DROP VIEW IF EXISTS OldestBambiUser;

CREATE VIEW OldestBambiUser AS
  select distinct uID
  from user natural join Loan
  where title = 'Bambi' and age =
          (select max(age)
           from user natural join loan
           where title = 'Bambi');
update Loan
set overdue = false
where overdue = true and uID  in
       (select * from OldestBambiUser);
```

```
update loan
set overdue = false
where loan.title = 'Bambi'
and overdue = true and uid in
(select uid from user where age  =
(select max(age)
 from (select uid, age, title from
       user natural join loan) R
 group by title
 having title = 'Bambi'));
```