Writing Joins in MySQL

Presented by:
Sheeri K. Cabral

Pythian
love your data

Topics Covered

- JOINs
  - OUTER
    - LEFT, RIGHT, FULL OUTER
  - INNER
    - INNER, NATURAL, comma (,)
  - CROSS
- Subqueries
  - DEPENDENT SUBQUERY
  - DERIVED TABLE
- Changing a subquery to a JOIN

Example

- 6-week intensive course
- Homework every Friday
  - Each assignment is 6% of your grade
  - Lowest grade is dropped
  - 30% of your grade, total
- Weekly tests every Monday
  - Same grading structure as hw
- Midterm – Wed. 1/20 – 15% of your grade
- Final exam – Friday 2/12 – 25% of your grade

Sample data

- work table

```
CREATE TABLE work (
 work_id tinyint(3) unsigned NOT NULL AUTO_INCREMENT,
 wname varchar(255) DEFAULT NULL,
 given date DEFAULT NULL,
 pct_of_grade tinyint(3) unsigned NOT NULL,
 PRIMARY KEY (work_id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Sample data

```
mysql> SELECT * FROM work;

+---------+---------+------------+--------------+
| work_id | wname   | given      | pct_of_grade |
+---------+---------+------------+--------------+
|       1 | hw1     | 2010-01-01 |            6 |
|       2 | test1   | 2010-01-04 |            6 |
|       3 | hw2     | 2010-01-08 |            6 |
|       4 | test2   | 2010-01-11 |            6 |
|       5 | hw3     | 2010-01-15 |            6 |
|       6 | test3   | 2010-01-18 |            6 |
|       7 | midterm | 2010-01-20 |           15 |
|       8 | hw4     | 2010-01-22 |            6 |
|       9 | test4   | 2010-01-25 |            6 |
|      10 | hw5     | 2010-01-29 |            6 |
|      11 | test5   | 2010-02-01 |            6 |
|      12 | hw6     | 2010-02-05 |            6 |
|      13 | test6   | 2010-02-08 |            6 |
|      14 | final   | 2010-02-12 |           25 |
+---------+---------+------------+--------------+
```

Sample data

- student table

```
CREATE TABLE student (
 student_id tinyint(3) unsigned NOT NULL AUTO_INCREMENT
 name varchar(255) DEFAULT NULL,
 email varchar(255) DEFAULT NULL,
 PRIMARY KEY (student_id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

- Entries

```
+------------+-----------------+------------------+
| student_id | name            | email            |
+------------+-----------------+------------------+
|          1 | Sheeri Cabral   | sheeri@foo.edu   |
|          2 | Giuseppe Maxia  | giuseppe@foo.edu |
|          3 | Colin Charles   | colin@foo.edu    |
|          4 | Ronald Bradford | ronald@foo.edu   |
+------------+-----------------+------------------+
```

Sample data

- student_work table

```
Create Table: CREATE TABLE student_work (
  student_id tinyint(3) unsigned NOT NULL,
  work_id tinyint(3) unsigned NOT NULL,
  grade_num tinyint(3) unsigned DEFAULT NULL,
  grade_letter char(2) DEFAULT NULL,
  for_grade enum('y','n') DEFAULT 'y',
  KEY student_id (student_id),
  KEY work_id (work_id),
  CONSTRAINT student_work_ibfk_1 FOREIGN KEY (student_id)
REFERENCES student (student_id),
  CONSTRAINT student_work_ibfk_2 FOREIGN KEY (work_id)
REFERENCES work (work_id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Sample data

- student_work entries

```
INSERT INTO student_work (student_id,work_id,grade_num)
VALUES
-- Sheeri had 88 for each hw/test except hw6 (72),
-- midterm 88, final 90, and she did not take test3.
(1,1,88),(1,2,88),(1,3,88),(1,4,88),(1,5,88),(1,7,88),
(1,9,88),(1,10,88),(1,11,88),(1,12,72),(1,13,88),
(1,14,90),
-- Giuseppe completed all assignments/tests:
(2,1,100),(2,2,100),(2,3,90),(2,4,88),(2,5,88),(2,6,85),
(2,7,95),(2,8,100),(2,9,100),(2,10,82),(2,11,85),
(2,12,89),(2,13,90),(2,14,96);
```

Sample data

- student_work entries

```
INSERT INTO student_work (student_id,work_id,grade_num)
VALUES
-- Colin is busy planning 2010 User Conference, and
-- did not complete any hw assignments, and as a result
-- did not do well on the tests
(3,2,75),(3,4,77),(3,6,89),(3,7,85),(3,9,72),(3,11,89),
(3,13,70),(3,14,80)
-- Ronald knew his stuff but got busy as the course
-- went on....
(4,1,100),(4,2,100),(4,3,95),(4,4,95),(4,5,90),(4,6,90),
(4,7,95),(4,8,85),(4,9,85),(4,10,80),(4,11,80),(4,12,75),
(4,13,75),(4,14,83);
```

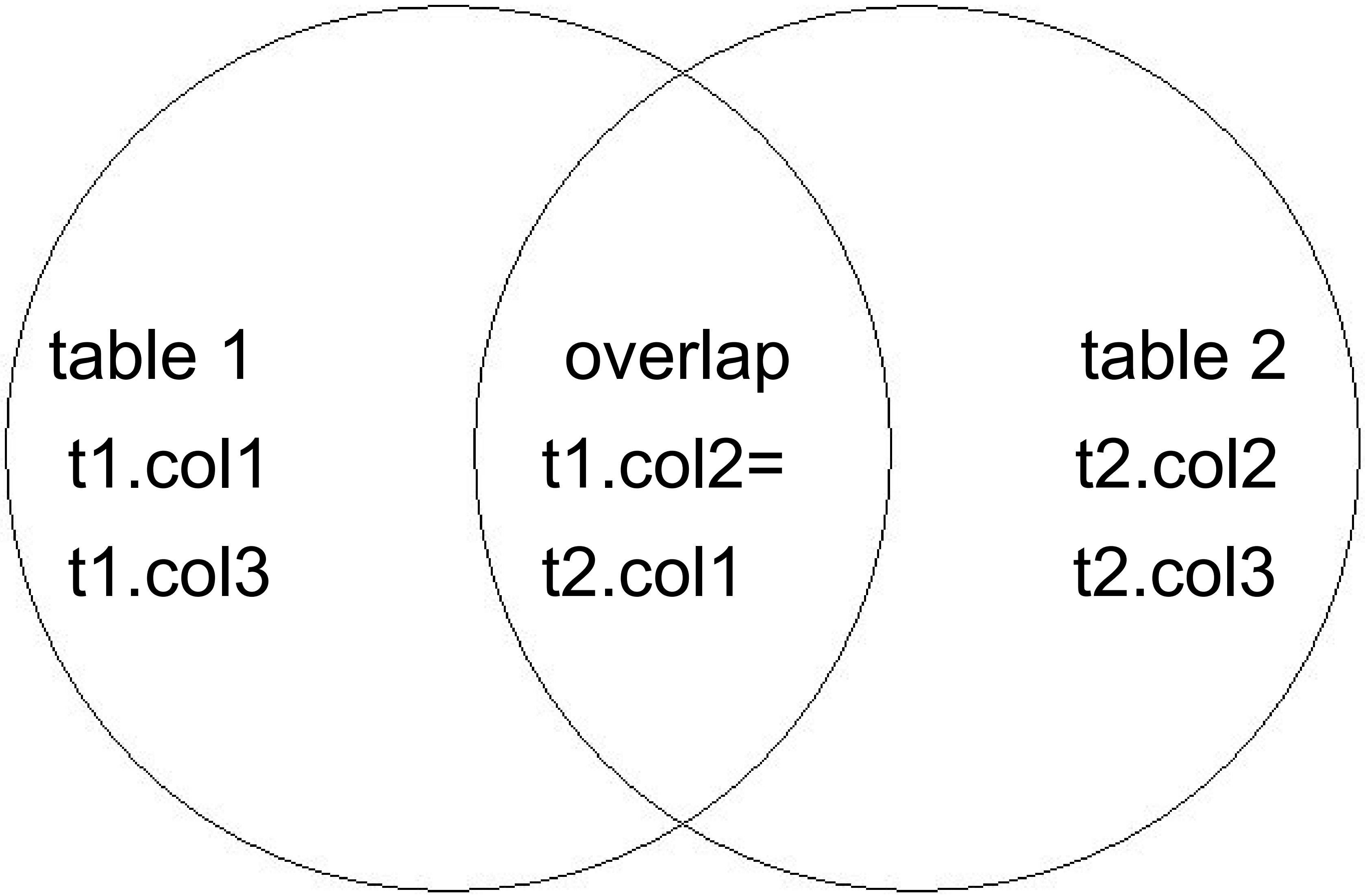
Sample data

- Global grade_num_letter table

```
CREATE TABLE grade_num_letter (
  grade_num tinyint(3) unsigned NOT NULL,
  grade_letter char(2) NOT NULL DEFAULT '',
  PRIMARY KEY (grade_num)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

mysql> select min(grade_num),max(grade_num),count(*) from
grade_num_letter;
+----------------+----------------+----------+
| min(grade_num) | max(grade_num) | count(*) |
+----------------+----------------+----------+
|              0 |            100 |      101 |
+----------------+----------------+----------+
1 row in set (0.00 sec)
```

Venn Diagram



MySQL JOINs

- A JOIN clause is optional
  - ON (tbl1.col1 ? tbl2.col2 AND ...)
    - Can also specify tbl1.col1 ? expr, (tbl1.col1<10)
  - USING (col1,col2,...)
    - Same as tbl1.col1=tbl2.col1 AND tbl1.col2=tbl2.col2
- INNER, CROSS, comma (,) all do the same thing
  - As of MySQL 5.0, COMMA join is lower precedence than other JOINS
  - SELECT...FROM tbl1, tbl2 ON (tbl1.col=tbl2.col) INNER JOIN tbl3 ON (tbl2.col=tbl3.col) WHERE tbl1.col=tbl2.col;

CROSS JOIN

- Cartesian product
    - All combinations

```
SELECT name, wname
FROM student CROSS JOIN work;
```

Usually not desired

INNER JOIN

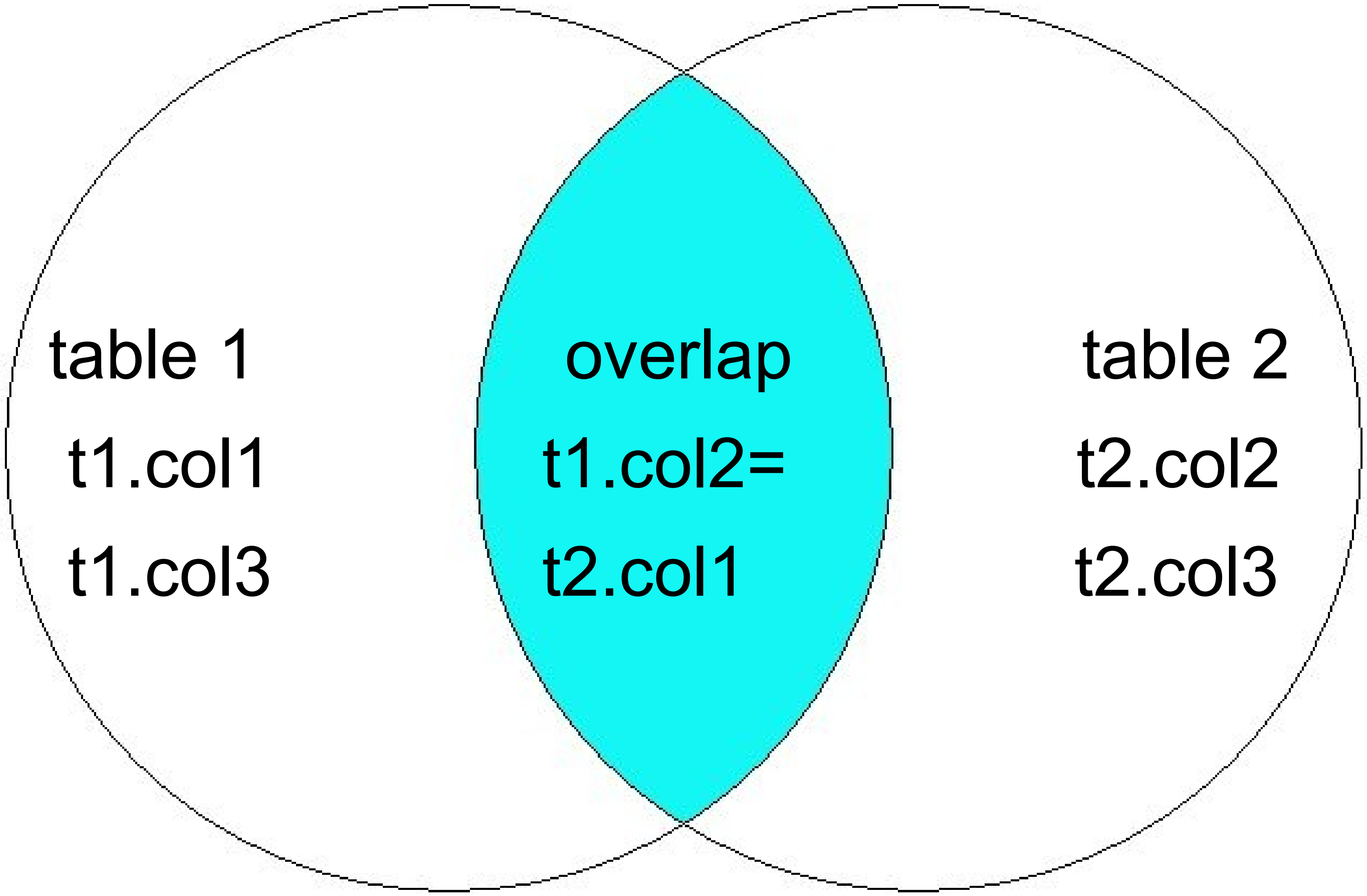


table 1
t1.col1
t1.col3
overlap
t1.col2=
t2.col1
table 2
t2.col2
t2.col3

INNER JOIN

- Show only rows that matches on both sides

```
SELECT s.grade_num, g.grade_letter
FROM student_work AS s
 INNER JOIN grade_num_letter AS g
  ON (s.grade_num=g.grade_num);
```

CROSS, INNER are semantic

- CROSS JOIN acting as an INNER JOIN:

```
SELECT s.grade_num, g.grade_letter
FROM student_work AS s
 CROSS JOIN grade_num_letter AS g
  ON (s.grade_num=g.grade_num);
```

- INNER JOIN acting as a CROSS JOIN:

```
SELECT name, wname FROM student INNER JOIN
work;
```

In fact you do not need either!

- JOIN acting as an INNER JOIN:

```
SELECT s.grade_num, g.grade_letter
FROM student_work AS s
 JOIN grade_num_letter AS g
  ON (s.grade_num=g.grade_num);
```

- JOIN acting as a CROSS JOIN:

```
SELECT name, wname FROM student JOIN work;
```

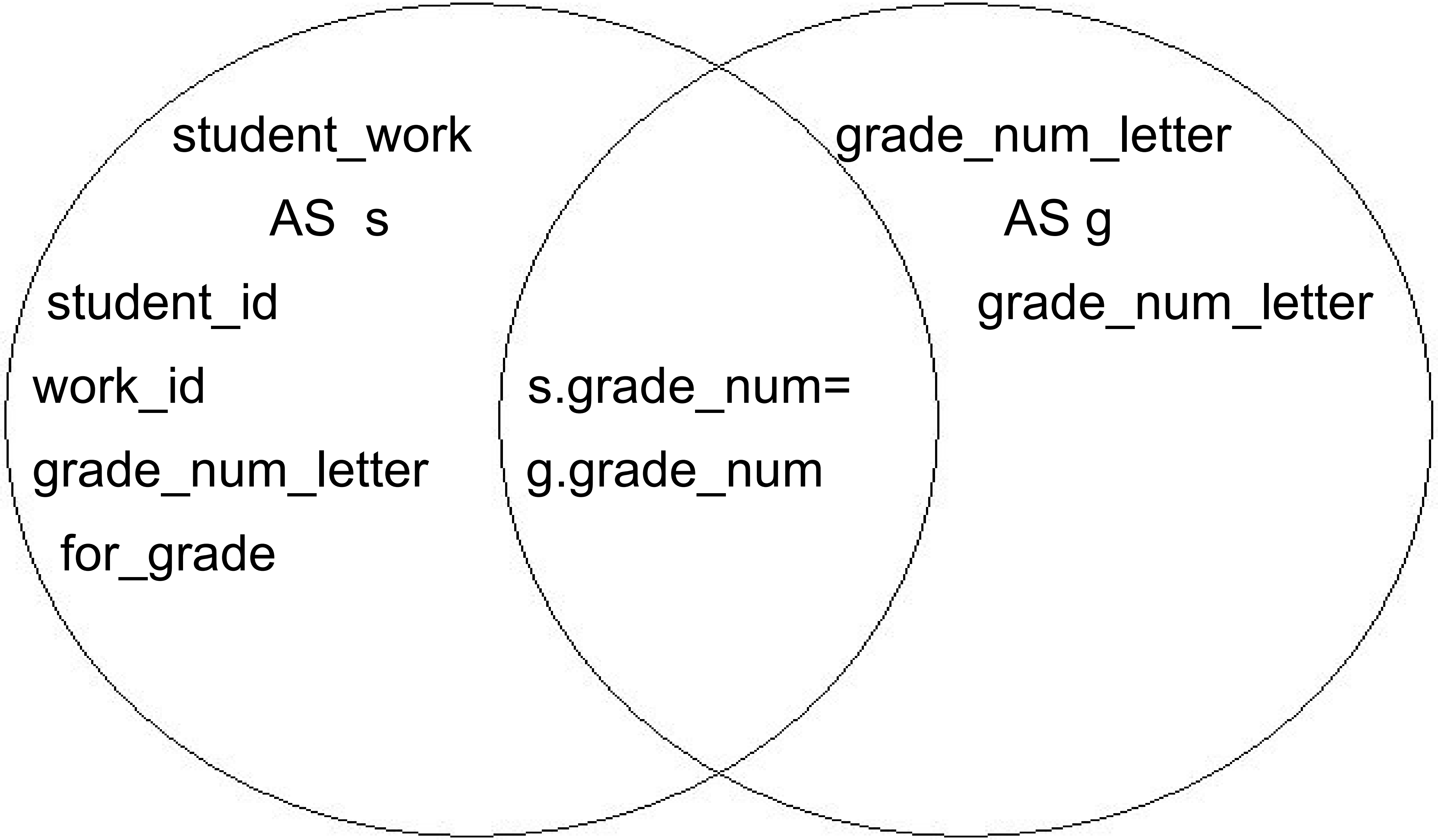
JOIN clause

- ON (…) or USING(...)
- Can specify in the WHERE clause
- Same results

My Best Practices

- Don't use a comma to join
  - Unexpected behavior with other JOINs in a query
- Never use JOIN; always use INNER JOIN or CROSS JOIN
  - Whoever debugs will know your intention
- Use a JOIN clause instead of a WHERE clause
  - More clear what is a filter and what is a join

Getting the letter grades



OUTER JOIN

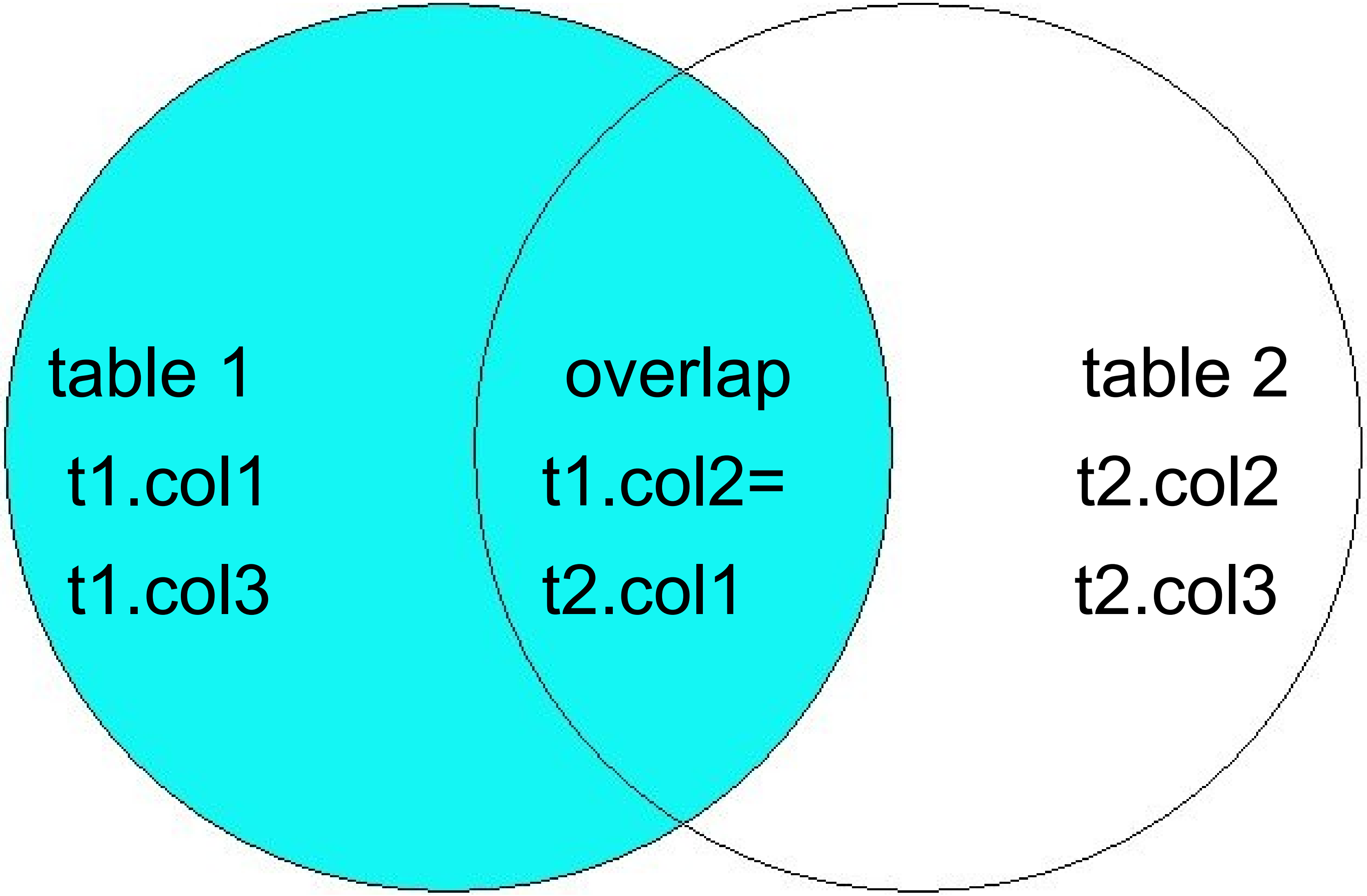
- Show all rows that match on one side
- “get all grades for test3”

```
SELECT name, wname, grade_num
FROM student CROSS JOIN work
LEFT OUTER JOIN student_work
USING (student_id,work_id)
WHERE wname='test3';
```

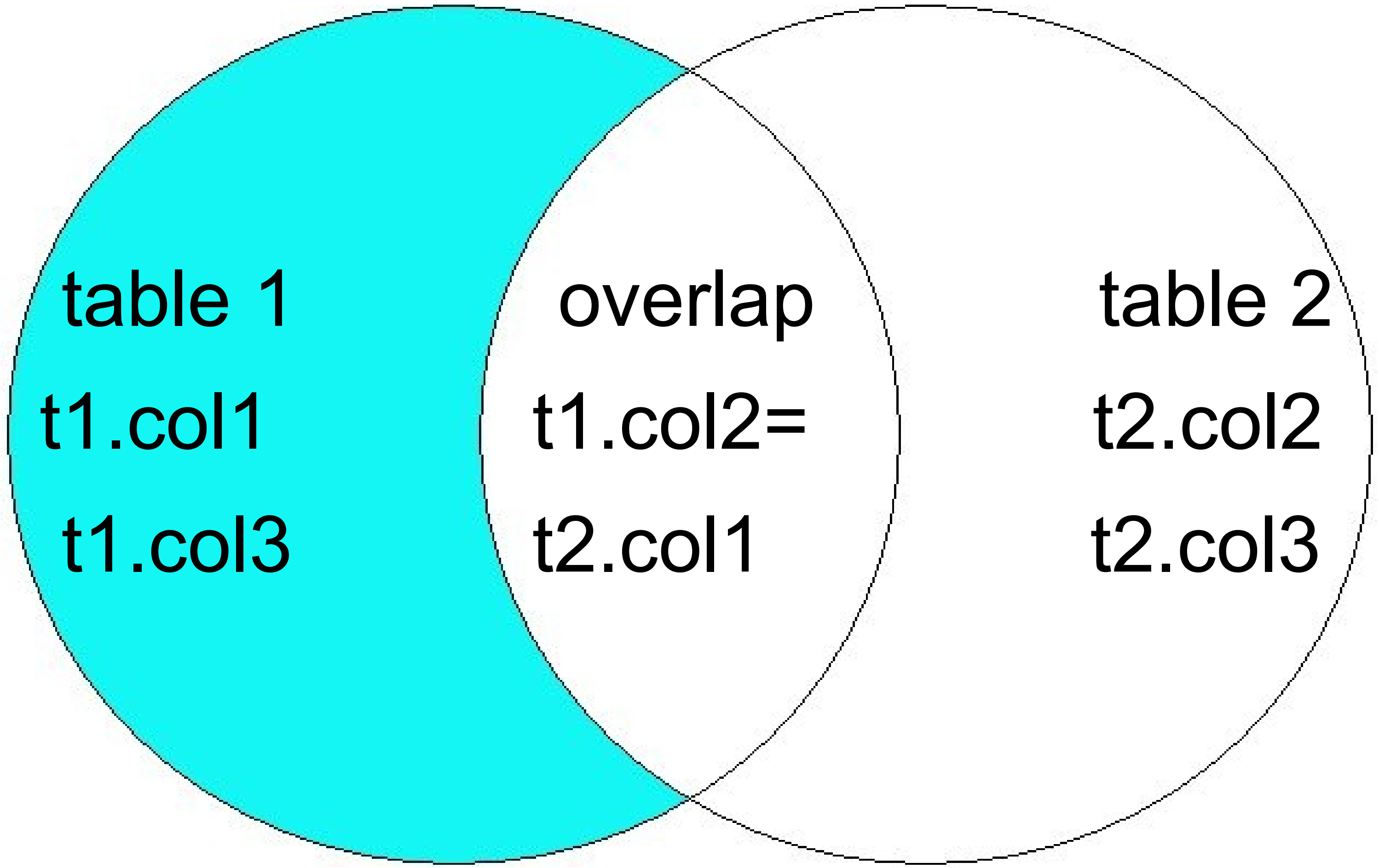
OUTER JOIN

- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- “OUTER” is redundant

LEFT JOIN, inclusive



LEFT JOIN, exclusive

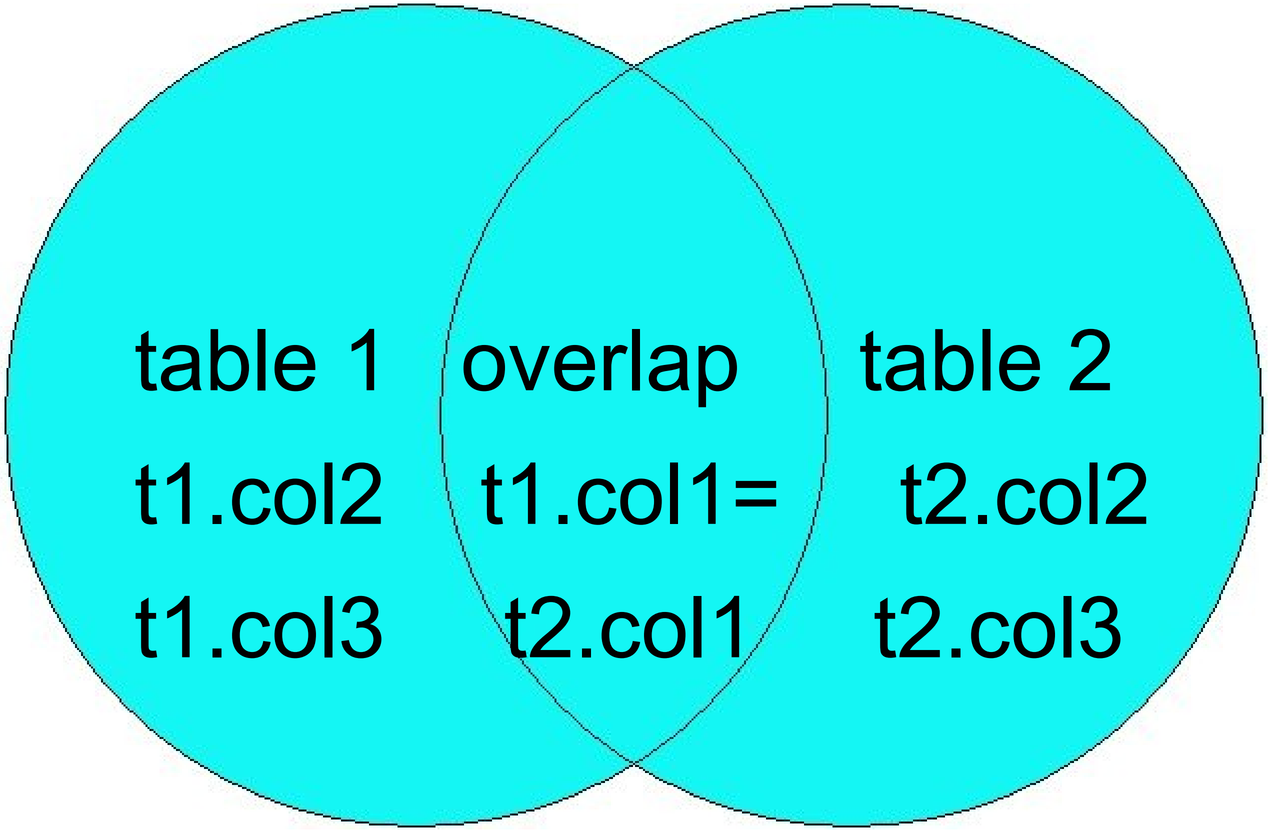


....WHERE t2.col1 IS NULL

FULL OUTER JOIN

- Does not exist in MySQL
- Can be simulated

FULL OUTER JOIN, inclusive



SELECT … FROM tbl1 LEFT JOIN tbl2 ...

UNION [ALL] … FROM tbl1 RIGHT JOIN tbl2 …

Pythian
love your data

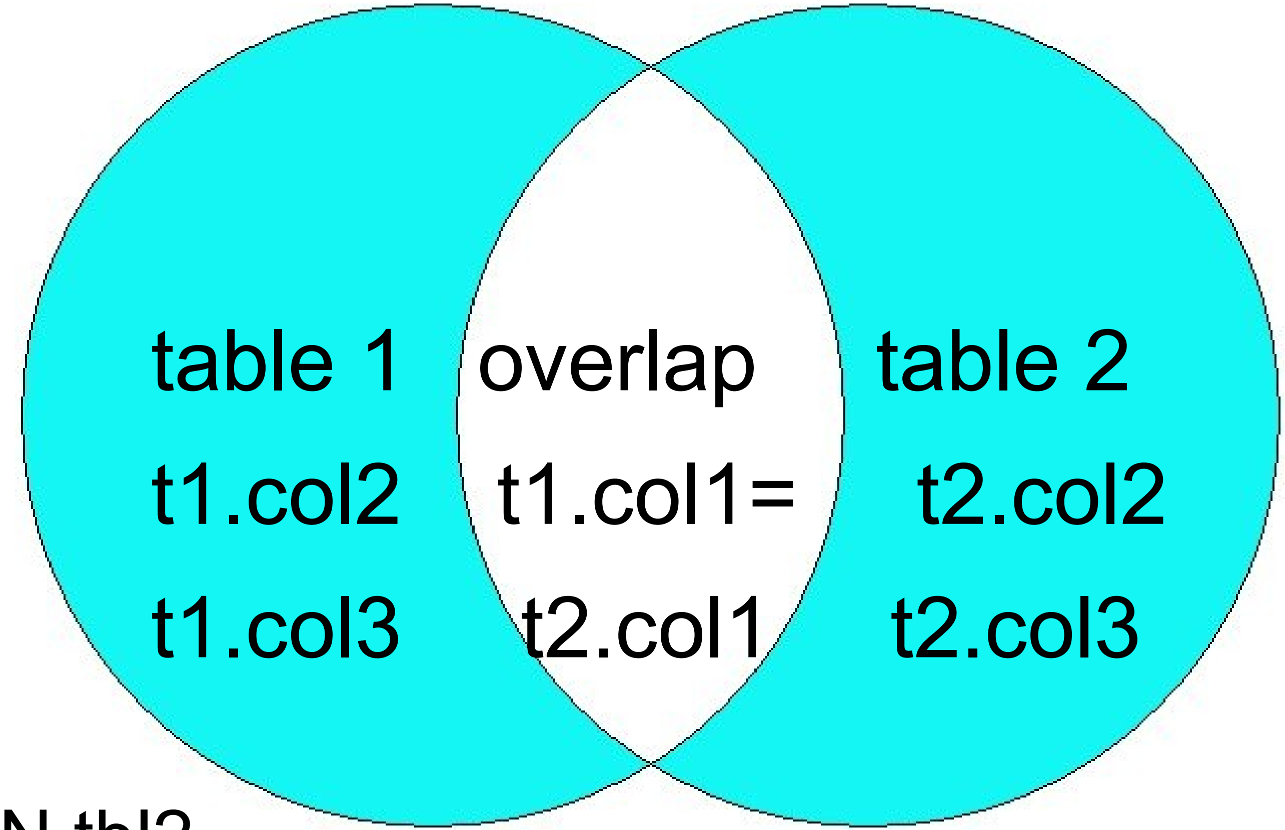
FULL OUTER JOIN

- If it is already in the first result, do not put it in the second result:

```
SELECT … FROM tbl1 LEFT JOIN tbl2 ...
UNION ALL … FROM tbl1 RIGHT JOIN tbl2 …
WHERE tbl1.col IS NULL
```

- In the 2nd query in the union, only rows that have no match in tbl1 are taken.

FULL OUTER JOIN, exclusive



SELECT …
FROM tbl1 LEFT JOIN tbl2 ...
WHERE tbl2.col1 IS NULL
UNION ALL … SELECT … FROM tbl1 RIGHT JOIN tbl2 …
WHERE tbl1.col IS NULL

NATURAL modifier

- Does not use a JOIN clause
  - JOIN clause is **all** matching field names
- Works for:
  - NATURAL JOIN
  - NATURAL LEFT JOIN
  - NATURAL RIGHT JOIN

NATURAL JOIN example

- Instead of:

```
SELECT name,grade_num
FROM student INNER JOIN student_work USING
(student_id)
WHERE name='Sheeri Cabral';
```

- Write:

```
SELECT name,grade_num
FROM student NATURAL JOIN student_work
WHERE name='Sheeri Cabral';
```

NATURAL JOIN gone awry

- Having the same field names when the fields are not equal:

```
SELECT sw.grade_num, gnl.grade_letter FROM
student_work AS sw INNER JOIN
grade_num_letter AS gnl USING (grade_num);
```

- Is NOT equivalent to:

```
SELECT sw.grade_num, gnl.grade_letter FROM
student_work AS sw NATURAL JOIN
grade_num_letter AS gnl;
```

This is why the field is called work.wname!

Subqueries

- A subquery is a query within a query
- More “natural” way of thinking for procedural thinkers
- SQL is declarative, and optimized that way

Procedural Thinking

- How to get the names and grades for test1

Procedural Thinking: Get the names and grades for test1

- “Get names and grades”

```
SELECT name, grade_num
```

Procedural Thinking: Get the names and grades for test1

SELECT name, grade_num

- “start with the join table”

FROM student_work

Procedural Thinking: Get the names and grades for test1

SELECT name, grade_num

FROM student_work

- “get the name”

INNER JOIN student USING (student_id)

Procedural Thinking: Get the names and grades for test1

```
SELECT name, grade_num
FROM student_work
INNER JOIN student USING (student_id)
```

- “But only get test1”

```
WHERE work_id IN (SELECT work_id FROM work
WHERE wname='test1')
```

Procedural Thinking: Get the names and grades for test1

```
SELECT name, grade_num
FROM student_work
INNER JOIN student USING (student_id)
WHERE work_id IN (SELECT work_id FROM work
WHERE wname='test1');
```

Procedural Thinking: Get the names and grades for test1

```
SELECT name, grade_num
FROM student_work
INNER JOIN student USING (student_id)
WHERE work_id IN (SELECT work_id FROM work
WHERE wname='test1');
```

Declarative Thinking: Get the names and grades for test1

- I have 3 sets of data
- student has the name
- student_work has the grades
- work has the name of the assignment

SELECT name, grade_num FROM .... WHERE wname='test1'

Declarative Thinking: Get the names and grades for test1

- student and student_work relate by student_id

```
SELECT name, grade_num FROM
student INNER JOIN student_work USING
(student_id)
....
WHERE wname='test1';
```

Declarative Thinking: Get the names and grades for test1

- work and student_work relate by work_id

SELECT name, grade_num FROM

student INNER JOIN student_work USING (student_id)

INNER JOIN work USING (work_id)

WHERE wname='test1';

```
SELECT name, wname, grade_num
FROM student CROSS JOIN work
LEFT OUTER JOIN student_work
USING (student_id,work_id)
WHERE wname='test3';
```

But.....

- That falls apart for test3, because Sheeri did not take test3.

- So now what?

Get all grades for test3

- Start with:

```
SELECT name, grade_num FROM....
WHERE wname='test3';
```

Get all grades for test3

- We want a listing for each row in “work” against each row in “student”

```
SELECT name, grade_num
FROM student CROSS JOIN work
....
WHERE wname='test3';
```

Get all grades for test3

- **We want a listing for each row in “work” against each row in “student”**

- What we want, not how to get it. That's declarative!

Get all grades for test3

- Grades might not exist for all the rows
- ...so we'll need an outer join
- Fill in the values from student_work for the grades that do exist, joining on student_id and work_id:

```
 SELECT name, grade_num
 FROM student CROSS JOIN work
 LEFT JOIN student_work USING (student_id,
work_id) WHERE wname='test3';
```

Drop the lowest test score

How?

Drop the lowest test score

For our purposes, NULL = 0.

So we'll need to keep the CROSS JOIN as in the previous query:

```
SELECT name, grade_num
FROM student CROSS JOIN work
LEFT JOIN student_work USING (student_id,
work_id)
```

Drop the lowest test score

Get all tests:

```
SELECT name, grade_num
FROM student CROSS JOIN work
LEFT JOIN student_work USING (student_id,
work_id) WHERE wname like 'test_';
```

We expect 24 rows returned....

Drop the lowest test score

Get minimum grade from all tests **per** person:

```
SELECT name, min(grade_num)
FROM student CROSS JOIN work
LEFT JOIN student_work USING (student_id,
work_id) WHERE wname like 'test_'
GROUP BY student_id;
```

We expect 4 rows

Drop the lowest test score

Convert to an UPDATE statement.....

```
SELECT name, min(grade_num)
FROM student CROSS JOIN work
LEFT JOIN student_work USING (student_id,
work_id) WHERE wname like 'test_'
GROUP BY student_id;
```

This is hard!

Drop the lowest test score

Now add in the rest...

```
UPDATE student_work as upd
INNER JOIN student_work as sel USING
(student_id, work_id)
RIGHT JOIN student USING (student_id, work_id)
CROSS JOIN work
SET upd.for_grade='n' WHERE wname like 'test_'
AND upd.grade_num=min(sel.grade_num)
GROUP BY sel.student_id;
```

That doesn't work....

Sometimes you need a subquery.....

```
UPDATE student_work

SET for_grade='n' WHERE
CONCAT(student_id,work_id) IN (SELECT
CONCAT(student_id,work_id) FROM

student CROSS JOIN work

LEFT JOIN student_work USING (student_id,
work_id)  WHERE wname like 'test_'

GROUP BY student_id);
```

That doesn't work either....

- Sometimes you need to do it in >1 query!
- Sometimes it's not necessary, but more optimal
- Problem is the min(grade_num)....GROUP BY
- So use a temporary table:
- CREATE TEMPORARY TABLE grade_to_drop SELECT min(coalesce(grade_num,0)) FROM student CROSS JOIN work LEFT JOIN student_work USING (student_id,work_id) WHERE wname like 'test_' group by student_id;

Temporary table

```
CREATE TEMPORARY TABLE grade_to_drop (
student_id tinyint unsigned not null,
work_id tinyint unsigned default null,
grade_num tinyint unsigned default null
);
```

Temporary table

```
INSERT INTO grade_to_drop (student_id,
grade_num)
SELECT student_id,min(coalesce(grade_num,0))
FROM student CROSS JOIN work
LEFT JOIN student_work USING
(student_id,work_id)
WHERE wname like 'test_'GROUP BY student_id;
```

Temporary table

```
UPDATE grade_to_drop AS gtd INNER JOIN
student_work AS sw USING
(student_id,grade_num)
SET gtd.work_id = sw.work_id ;

UPDATE student_work AS sw INNER JOIN
grade_to_drop AS gtd USING
(student_id,work_id)
SET sw.for_grade='n' ;
```

Temporary table

```
Repeat for dropping the lowest homework

DROP TABLE IF EXISTS grade_to_drop;
```

More best practices

- EXPLAIN all your queries, and get the best “type” possible
- Avoid JOIN hints (index hints, STRAIGHT_JOIN)
- Try to optimize subqueries into JOINs if possible

That's it!

- Questions?
- Comments?

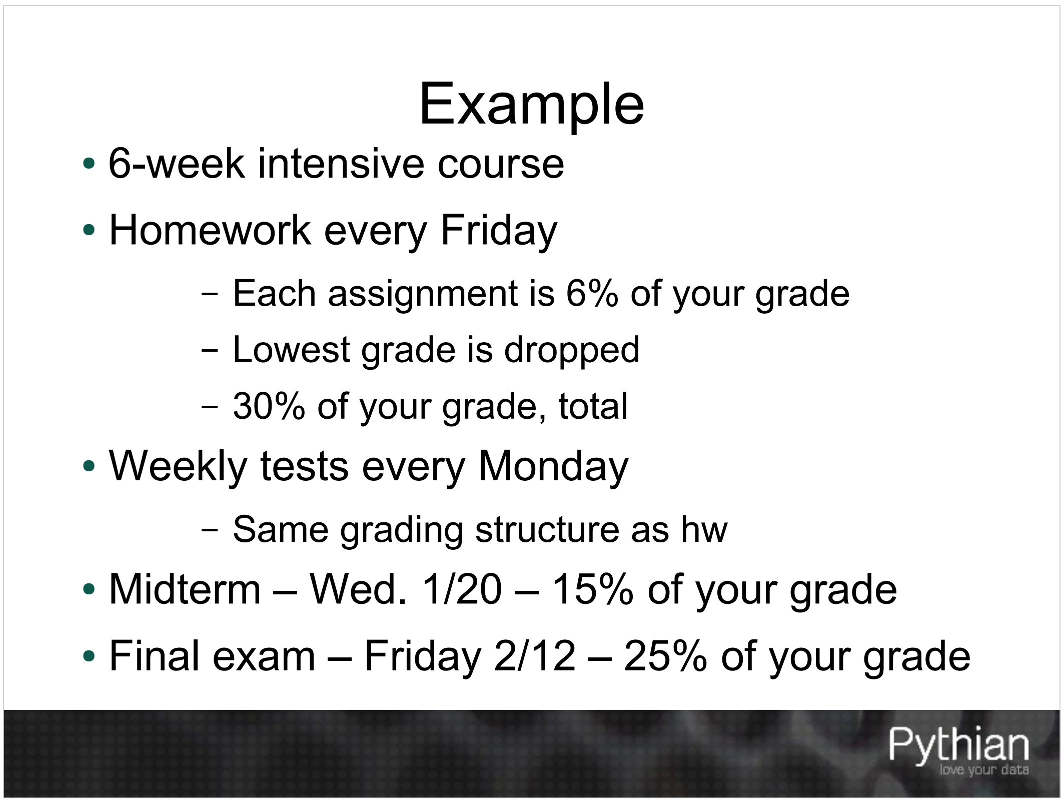
Writing Joins in MySQL

Presented by:
Sheeri K. Cabral

Pythian
love your data

Topics Covered

- JOINs
  - OUTER
    - LEFT, RIGHT, FULL OUTER
  - INNER
    - INNER, NATURAL, comma (,)
  - CROSS
- Subqueries
  - DEPENDENT SUBQUERY
  - DERIVED TABLE
- Changing a subquery to a JOIN



Example

- 6-week intensive course
- Homework every Friday
  - Each assignment is 6% of your grade
  - Lowest grade is dropped
  - 30% of your grade, total
- Weekly tests every Monday
  - Same grading structure as hw
- Midterm – Wed. 1/20 – 15% of your grade
- Final exam – Friday 2/12 – 25% of your grade

Pythian
love your data

Hw due the first day of class, how cruel!

Sample data

- work table

```
CREATE TABLE work (
 work_id tinyint(3) unsigned NOT NULL AUTO_INCREMENT,
 wname varchar(255) DEFAULT NULL,
 given date DEFAULT NULL,
 pct_of_grade tinyint(3) unsigned NOT NULL,
 PRIMARY KEY (work_id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Sample data

```
mysql> SELECT * FROM work;
+---------+---------+------------+--------------+
| work_id | wname   | given      | pct_of_grade |
+---------+---------+------------+--------------+
|       1 | hw1     | 2010-01-01 |            6 |
|       2 | test1   | 2010-01-04 |            6 |
|       3 | hw2     | 2010-01-08 |            6 |
|       4 | test2   | 2010-01-11 |            6 |
|       5 | hw3     | 2010-01-15 |            6 |
|       6 | test3   | 2010-01-18 |            6 |
|       7 | midterm | 2010-01-20 |           15 |
|       8 | hw4     | 2010-01-22 |            6 |
|       9 | test4   | 2010-01-25 |            6 |
|      10 | hw5     | 2010-01-29 |            6 |
|      11 | test5   | 2010-02-01 |            6 |
|      12 | hw6     | 2010-02-05 |            6 |
|      13 | test6   | 2010-02-08 |            6 |
|      14 | final   | 2010-02-12 |           25 |
+---------+---------+------------+--------------+
```

Sample data

- student table

```
CREATE TABLE student (
 student_id tinyint(3) unsigned NOT NULL AUTO_INCREMENT
 name varchar(255) DEFAULT NULL,
 email varchar(255) DEFAULT NULL,
 PRIMARY KEY (student_id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

- Entries

```
+------------+-----------------+------------------+
| student_id | name            | email            |
+------------+-----------------+------------------+
|          1 | Sheeri Cabral   | sheeri@foo.edu   |
|          2 | Giuseppe Maxia  | giuseppe@foo.edu |
|          3 | Colin Charles   | colin@foo.edu    |
|          4 | Ronald Bradford | ronald@foo.edu   |
+------------+-----------------+------------------+
```

Pythian
love your data

Sample data

- student_work table

```
Create Table: CREATE TABLE student_work (
  student_id tinyint(3) unsigned NOT NULL,
  work_id tinyint(3) unsigned NOT NULL,
  grade_num tinyint(3) unsigned DEFAULT NULL,
  grade_letter char(2) DEFAULT NULL,
  for_grade enum('y','n') DEFAULT 'y',
  KEY student_id (student_id),
  KEY work_id (work_id),
  CONSTRAINT student_work_ibfk_1 FOREIGN KEY (student_id)
REFERENCES student (student_id),
  CONSTRAINT student_work_ibfk_2 FOREIGN KEY (work_id)
REFERENCES work (work_id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Sample data

- student_work entries

```
INSERT INTO student_work (student_id,work_id,grade_num)
VALUES
-- Sheeri had 88 for each hw/test except hw6 (72),
-- midterm 88, final 90, and she did not take test3.
(1,1,88),(1,2,88),(1,3,88),(1,4,88),(1,5,88),(1,7,88),
(1,9,88),(1,10,88),(1,11,88),(1,12,72),(1,13,88),
(1,14,90),
-- Giuseppe completed all assignments/tests:
(2,1,100),(2,2,100),(2,3,90),(2,4,88),(2,5,88),(2,6,85),
(2,7,95),(2,8,100),(2,9,100),(2,10,82),(2,11,85),
(2,12,89),(2,13,90),(2,14,96);
```

Sample data

- student_work entries

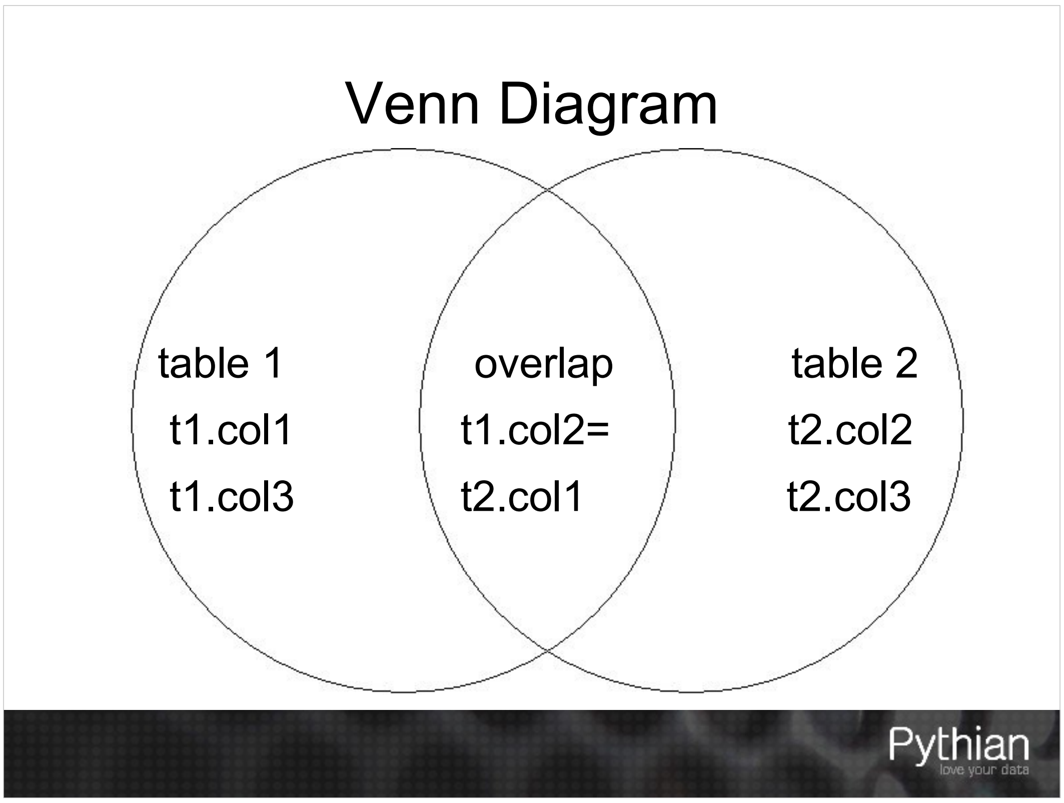
```
INSERT INTO student_work (student_id,work_id,grade_num)
VALUES
-- Colin is busy planning 2010 User Conference, and
-- did not complete any hw assignments, and as a result
-- did not do well on the tests
(3,2,75),(3,4,77),(3,6,89),(3,7,85),(3,9,72),(3,11,89),
(3,13,70),(3,14,80)
-- Ronald knew his stuff but got busy as the course
-- went on....
(4,1,100),(4,2,100),(4,3,95),(4,4,95),(4,5,90),(4,6,90),
(4,7,95),(4,8,85),(4,9,85),(4,10,80),(4,11,80),(4,12,75),
(4,13,75),(4,14,83);
```

Sample data

- Global grade_num_letter table

```
CREATE TABLE grade_num_letter (
  grade_num tinyint(3) unsigned NOT NULL,
  grade_letter char(2) NOT NULL DEFAULT '',
  PRIMARY KEY (grade_num)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

mysql> select min(grade_num),max(grade_num),count(*) from
grade_num_letter;
+----------------+----------------+----------+
| min(grade_num) | max(grade_num) | count(*) |
+----------------+----------------+----------+
|              0 |            100 |      101 |
+----------------+----------------+----------+
1 row in set (0.00 sec)
```



Venn Diagram

The overlap is only what is equal, even though there may be 2 fields with the same name.

MySQL JOINs

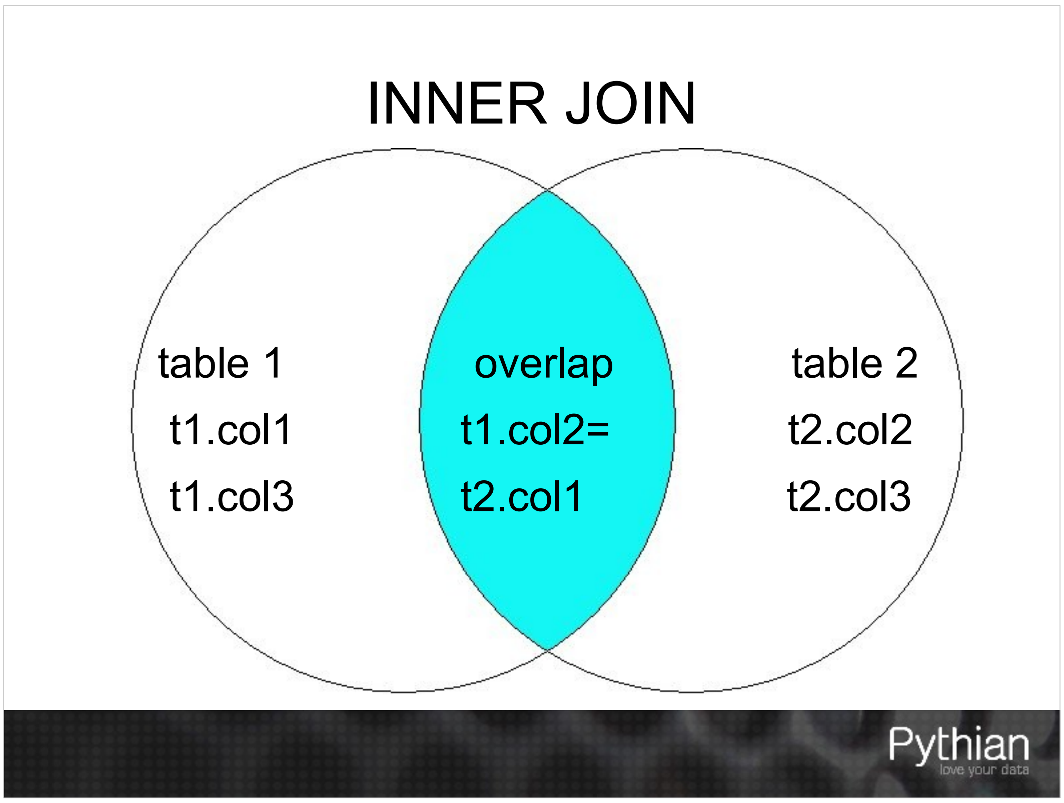
- A JOIN clause is optional
  - ON (tbl1.col1 ? tbl2.col2 AND ...)
    - Can also specify tbl1.col1 ? expr, (tbl1.col1<10)
  - USING (col1,col2,...)
    - Same as tbl1.col1=tbl2.col1 AND tbl1.col2=tbl2.col2
- INNER, CROSS, comma (,) all do the same thing
  - As of MySQL 5.0, COMMA join is lower precedence than other JOINS
  - SELECT...FROM tbl1, tbl2 ON (tbl1.col=tbl2.col) INNER JOIN tbl3 ON (tbl2.col=tbl3.col) WHERE tbl1.col=tbl2.col;

CROSS JOIN

- Cartesian product
  - All combinations

```
SELECT name, wname
FROM student CROSS JOIN work;
```

Usually not desired



INNER JOIN

- Show only rows that matches on both sides
- Note that the SELECT may show the columns, but the venn diagram is showing the **matching**.

INNER JOIN

- Show only rows that matches on both sides

```
SELECT s.grade_num, g.grade_letter
FROM student_work AS s
 INNER JOIN grade_num_letter AS g
  ON (s.grade_num=g.grade_num);
```

CROSS, INNER are semantic

- CROSS JOIN acting as an INNER JOIN:

```
SELECT s.grade_num, g.grade_letter
FROM student_work AS s
 CROSS JOIN grade_num_letter AS g
  ON (s.grade_num=g.grade_num);
```

- INNER JOIN acting as a CROSS JOIN:

```
SELECT name, wname FROM student INNER JOIN
work;
```

In fact you do not need either!

- JOIN acting as an INNER JOIN:

```
SELECT s.grade_num, g.grade_letter
FROM student_work AS s
 JOIN grade_num_letter AS g
  ON (s.grade_num=g.grade_num);
```

- JOIN acting as a CROSS JOIN:

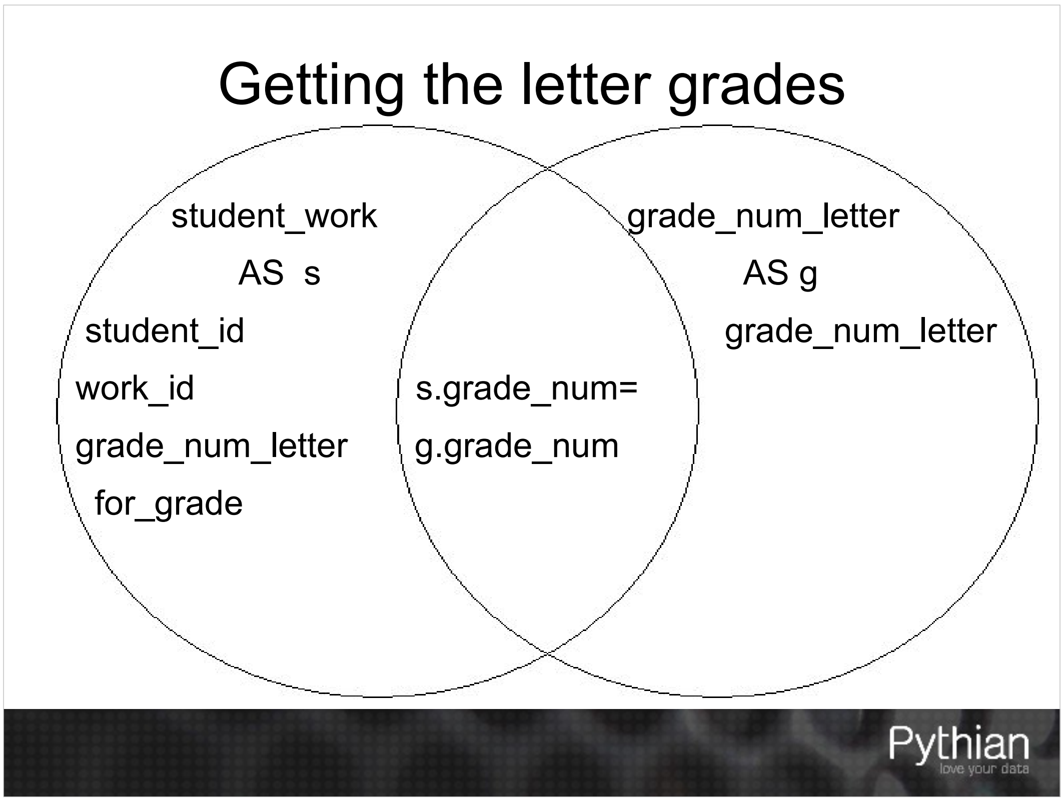
```
SELECT name, wname FROM student JOIN work;
```

JOIN clause

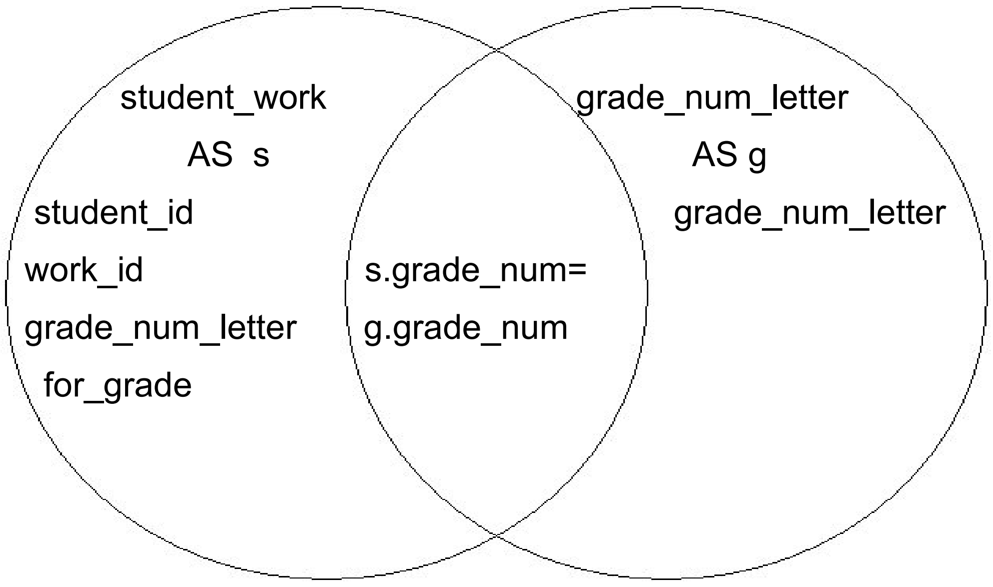
- ON (…) or USING(...)
- Can specify in the WHERE clause
- Same results

My Best Practices

- Don't use a comma to join
  - Unexpected behavior with other JOINs in a query
- Never use JOIN; always use INNER JOIN or CROSS JOIN
  - Whoever debugs will know your intention
- Use a JOIN clause instead of a WHERE clause
  - More clear what is a filter and what is a join



Getting the letter grades



OUTER JOIN

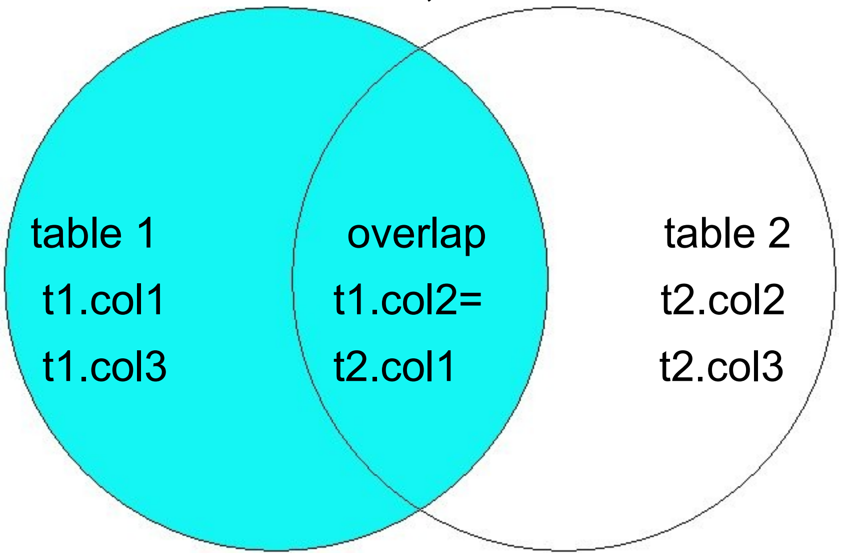
- Show all rows that match on one side
- “get all grades for test3”

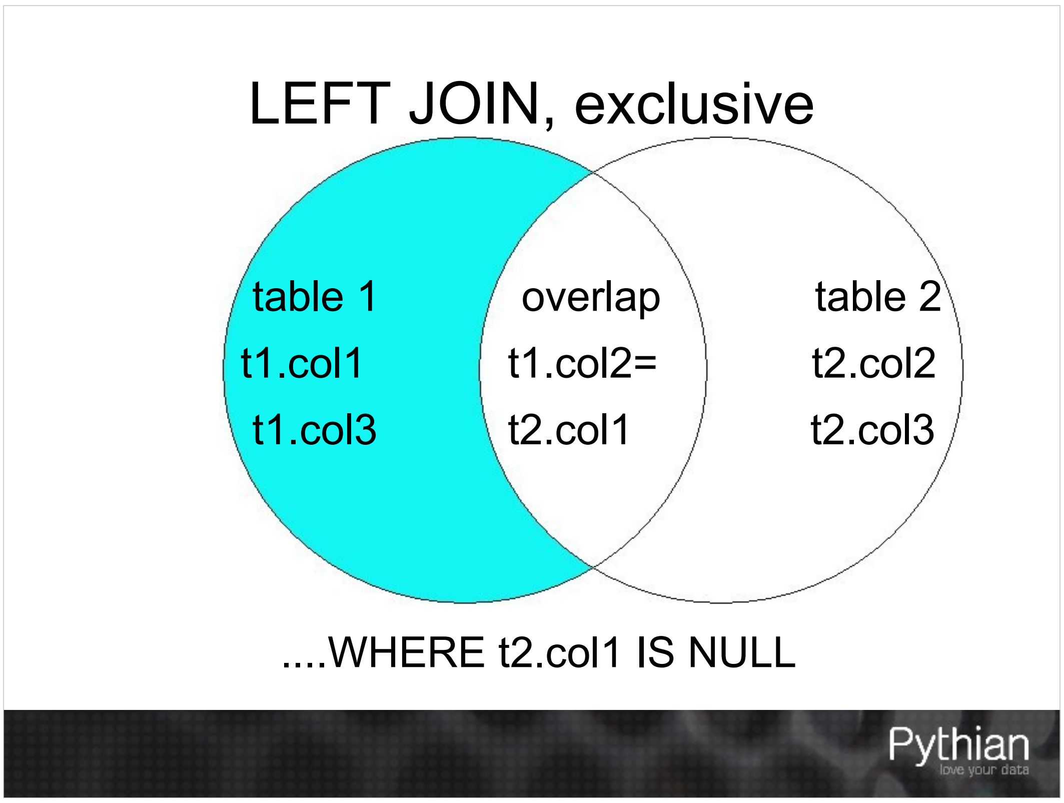
```
SELECT name, wname, grade_num
FROM student CROSS JOIN work
LEFT OUTER JOIN student_work
USING (student_id,work_id)
WHERE wname='test3';
```

OUTER JOIN

- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- “OUTER” is redundant

LEFT JOIN, inclusive





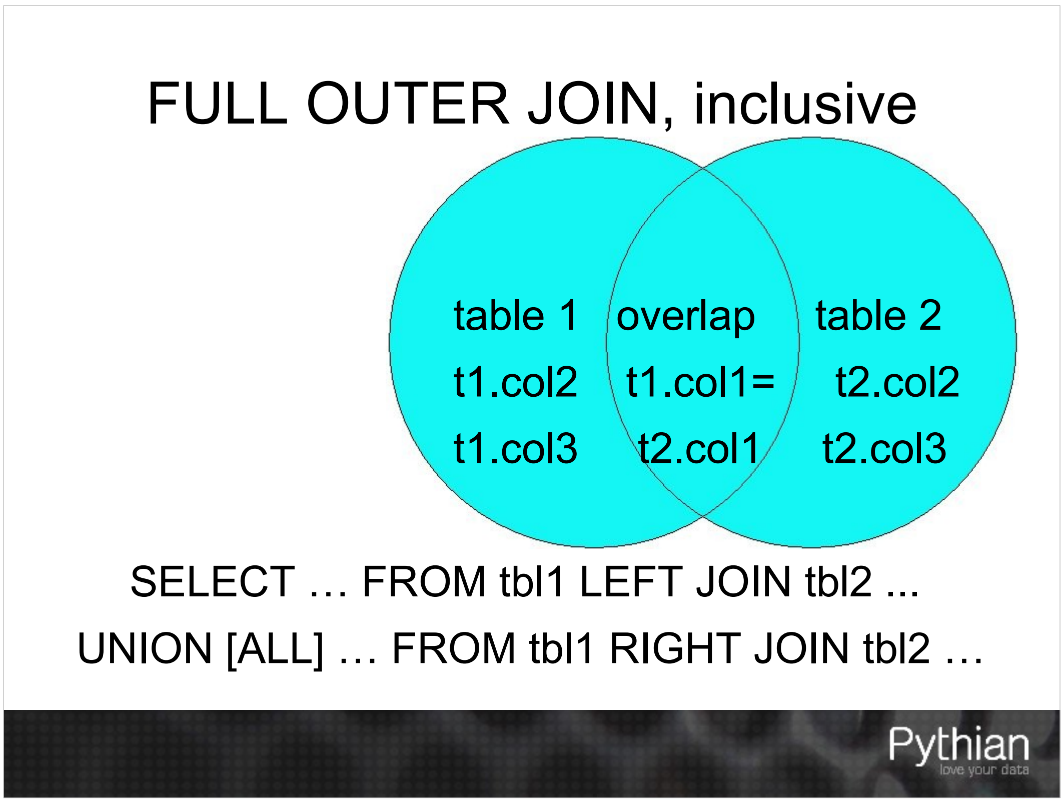
LEFT JOIN, exclusive

....WHERE t2.col1 IS NULL

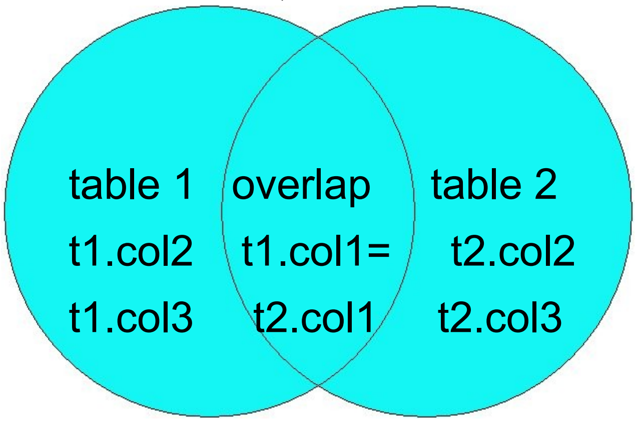
- RIGHT JOIN is just a mirror image

FULL OUTER JOIN

- Does not exist in MySQL
- Can be simulated



FULL OUTER JOIN, inclusive



SELECT … FROM tbl1 LEFT JOIN tbl2 ...

UNION [ALL] … FROM tbl1 RIGHT JOIN tbl2 …

Pythian
love your data

- UNION will eliminate duplicate rows
- UNION ALL just adds togeher the 2 sets

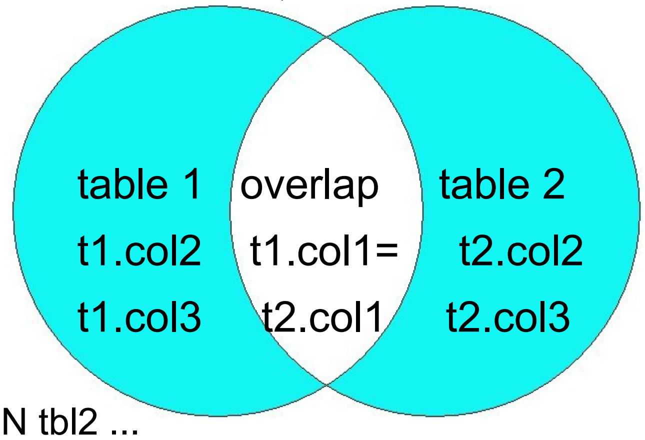
FULL OUTER JOIN

- If it is already in the first result, do not put it in the second result:

```
SELECT … FROM tbl1 LEFT JOIN tbl2 ...
UNION ALL … FROM tbl1 RIGHT JOIN tbl2 …
WHERE tbl1.col IS NULL
```

- In the 2nd query in the union, only rows that have no match in tbl1 are taken.

FULL OUTER JOIN, exclusive



```
SELECT …
FROM tbl1 LEFT JOIN tbl2 ...
WHERE tbl2.col1 IS NULL
UNION ALL … SELECT … FROM tbl1 RIGHT JOIN tbl2 …
WHERE tbl1.col IS NULL
```

NATURAL modifier

- Does not use a JOIN clause
  - JOIN clause is **all** matching field names
- Works for:
  - NATURAL JOIN
  - NATURAL LEFT JOIN
  - NATURAL RIGHT JOIN

NATURAL JOIN example

- Instead of:

```
SELECT name,grade_num
FROM student INNER JOIN student_work USING
(student_id)
WHERE name='Sheeri Cabral';
```

- Write:

```
SELECT name,grade_num
FROM student NATURAL JOIN student_work
WHERE name='Sheeri Cabral';
```

NATURAL JOIN gone awry

- Having the same field names when the fields are not equal:

```
SELECT sw.grade_num, gnl.grade_letter FROM
student_work AS sw INNER JOIN
grade_num_letter AS gnl USING (grade_num);
```

- Is NOT equivalent to:

```
SELECT sw.grade_num, gnl.grade_letter FROM
student_work AS sw NATURAL JOIN
grade_num_letter AS gnl;
```

This is why the field is called work.wname!

Subqueries

- A subquery is a query within a query
- More “natural” way of thinking for procedural thinkers
- SQL is declarative, and optimized that way

Procedural Thinking

- How to get the names and grades for test1

Procedural Thinking: Get the names and grades for test1

- "Get names and grades"

```
SELECT name, grade_num
```

Procedural Thinking: Get the names and grades for test1

SELECT name, grade_num

- “start with the join table”

FROM student_work

Procedural Thinking: Get the names and grades for test1

```
SELECT name, grade_num
FROM student_work
```

- “get the name”

```
INNER JOIN student USING (student_id)
```

Procedural Thinking: Get the names and grades for test1

SELECT name, grade_num

FROM student_work

INNER JOIN student USING (student_id)

- “But only get test1”

WHERE work_id IN (SELECT work_id FROM work WHERE wname='test1')

Procedural Thinking: Get the names and grades for test1

```
SELECT name, grade_num
FROM student_work
INNER JOIN student USING (student_id)
WHERE work_id IN (SELECT work_id FROM work
WHERE wname='test1');
```

Procedural Thinking: Get the names and grades for test1

```
SELECT name, grade_num
FROM student_work
INNER JOIN student USING (student_id)
WHERE work_id IN (SELECT work_id FROM work
WHERE wname='test1');
```

Declarative Thinking: Get the names and grades for test1

- I have 3 sets of data
- student has the name
- student_work has the grades
- work has the name of the assignment

SELECT name, grade_num FROM .... WHERE wname='test1'

Declarative Thinking: Get the names and grades for test1

- student and student_work relate by student_id

SELECT name, grade_num FROM

student INNER JOIN student_work USING (student_id)

....

WHERE wname='test1';

Pythian
love your data

Declarative Thinking: Get the names and grades for test1

- work and student_work relate by work_id

SELECT name, grade_num FROM

student INNER JOIN student_work USING (student_id)

INNER JOIN work USING (work_id)

WHERE wname='test1';

```
SELECT name, wname, grade_num
FROM student CROSS JOIN work
LEFT OUTER JOIN student_work
USING (student_id,work_id)
WHERE wname='test3';
```

But.....

- That falls apart for test3, because Sheeri did not take test3.
- So now what?

Get all grades for test3

- Start with:

```
SELECT name, grade_num FROM....
WHERE wname='test3';
```

Get all grades for test3

- We want a listing for each row in “work” against each row in “student”

```
SELECT name, grade_num
FROM student CROSS JOIN work
....
WHERE wname='test3';
```

Get all grades for test3

- **We want a listing for each row in “work” against each row in “student”**

- What we want, not how to get it. That's declarative!

Get all grades for test3

- Grades might not exist for all the rows
- ...so we'll need an outer join
- Fill in the values from student_work for the grades that do exist, joining on student_id and work_id:

```
 SELECT name, grade_num
 FROM student CROSS JOIN work
 LEFT JOIN student_work USING (student_id,
work_id) WHERE wname='test3';
```

Drop the lowest test score

How?

Drop the lowest test score

For our purposes, NULL = 0.

So we'll need to keep the CROSS JOIN as in the previous query:

```
SELECT name, grade_num
FROM student CROSS JOIN work
LEFT JOIN student_work USING (student_id,
work_id)
```

Drop the lowest test score

Get all tests:

```
SELECT name, grade_num
FROM student CROSS JOIN work
LEFT JOIN student_work USING (student_id,
work_id) WHERE wname like 'test_';
```

We expect 24 rows returned....

Drop the lowest test score

Get minimum grade from all tests **per** person:

SELECT name, min(grade_num)

FROM student CROSS JOIN work

LEFT JOIN student_work USING (student_id, work_id) WHERE wname like 'test_'

GROUP BY student_id;

We expect 4 rows

Drop the lowest test score

Convert to an UPDATE statement.....

SELECT name, min(grade_num)

FROM student CROSS JOIN work

LEFT JOIN student_work USING (student_id, work_id) WHERE wname like 'test_'

GROUP BY student_id;

This is hard!

Drop the lowest test score

Now add in the rest...

```
UPDATE student_work as upd
INNER JOIN student_work as sel USING
(student_id, work_id)
RIGHT JOIN student USING (student_id, work_id)
CROSS JOIN work
SET upd.for_grade='n' WHERE wname like 'test_'
AND upd.grade_num=min(sel.grade_num)
GROUP BY sel.student_id;
```

That doesn't work....

Sometimes you need a subquery.....

```
UPDATE student_work
SET for_grade='n' WHERE
CONCAT(student_id,work_id) IN (SELECT
CONCAT(student_id,work_id) FROM
student CROSS JOIN work
LEFT JOIN student_work USING (student_id,
work_id)  WHERE wname like 'test_'
GROUP BY student_id);
```

Pythian
love your data

That doesn't work either....

- Sometimes you need to do it in >1 query!
- Sometimes it's not necessary, but more optimal
- Problem is the min(grade_num)....GROUP BY
- So use a temporary table:
- CREATE TEMPORARY TABLE grade_to_drop SELECT min(coalesce(grade_num,0)) FROM student CROSS JOIN work LEFT JOIN student_work USING (student_id,work_id) WHERE wname like 'test_' group by student_id;

Temporary table

```
CREATE TEMPORARY TABLE grade_to_drop (
student_id tinyint unsigned not null,
work_id tinyint unsigned default null,
grade_num tinyint unsigned default null
);
```

Temporary table

```
INSERT INTO grade_to_drop (student_id,
grade_num)
SELECT student_id,min(coalesce(grade_num,0))
FROM student CROSS JOIN work
LEFT JOIN student_work USING
(student_id,work_id)
WHERE wname like 'test_'GROUP BY student_id;
```

Temporary table

```
UPDATE grade_to_drop AS gtd INNER JOIN
student_work AS sw USING
(student_id,grade_num)
SET gtd.work_id = sw.work_id ;

UPDATE student_work AS sw INNER JOIN
grade_to_drop AS gtd USING
(student_id,work_id)
SET sw.for_grade='n' ;
```

Temporary table

Repeat for dropping the lowest homework

```
DROP TABLE IF EXISTS grade_to_drop;
```

More best practices

- EXPLAIN all your queries, and get the best “type” possible
- Avoid JOIN hints (index hints, STRAIGHT_JOIN)
- Try to optimize subqueries into JOINs if possible

That's it!

- Questions?
- Comments?