Dr Renata Borovica-Gajic
David Eccles

INFO90002 Database Systems & Information Modelling

Lecture 10
SQL Part 2
SELECT, HAVING, SET OPERATORS, VENN DIAGRAMS, DML, CASE

Relational Algebra Homework

Optional

RA Homework Example Instances

Reserves (R1)

sid	bid	day
22	101	10/10/96
58	103	11/12/96

Boats

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Sailors 1 (S1)

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Sailors 2 (S2)

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Relational Algebra Homework

1. Find the name of all sailors whose rating is above 9

2. Find all sailors who reserved a boat prior to November 1, 1996

3. Find (the names of) all boats that have been reserved at least once

   = whether the boat has been reserved

4. Find all pairs of sailors with the same rating

   (HINT: not the same sailors in both relations)

Lecture 10. SQL Part 2

SELECT, HAVING, SET OPERATORS, VENN DIAGRAMS, DML, CASE

This Lecture

- Extending your knowledge
  - DML
    - Comparison & Logic Operators
    - Set Operations
    - Subquery
    - Multiple record INSERTs
    - INSERT from a table, UPDATE, DELETE, REPLACE
    - Views
  - DDL
    - ALTER and DROP, TRUNCATE, RENAME
    - CTAS

- How to think about SQL
  - Problem Solving

Things to Remember about SQL

- SQL keywords are case insensitive
  - We try to CAPITALISE them to make them clear
  - Improve readability of your statements
- Table names are Operating System Sensitive
  - If case sensitivity exists in the operating system, then the table names are case sensitive! (i.e. Linux)
    - Account <> ACCOUNT
- Field names are case insensitive
  - ACCOUNTID == AccountID == AcCoUnTID
- You can do maths in SQL…
  - SELECT 1*1+1/1-1;

A Note On SELECT

- The select statement's job is just to return rows of data, it doesn't care about the order of these rows unless you specify the ORDER BY clause
- So what order do rows come out in if you don't specify the ORDER BY clause?
  - Any order
  - Possibly the order the records were created in
  - It is undefined
    - Because SQL may optimise the query which may change the order of results…
- So make sure you get into the habit of using the ORDER BY clause *if* you need a particular order
  - If you don't need order, don't use it – it's going to be slower

HAVING Clause: Revisited

- The HAVING clause was added to SQL because the WHERE keyword **cannot be used** with **aggregate** functions.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

- **Example**:

List the number of customers of each country, but ONLY include countries with more than 5 customers

```
SELECT COUNT(CustomerID), CountryName
FROM Customers
GROUP BY CountryName
HAVING COUNT(CustomerID) > 5;
```

Condition over the aggregate

Comparison and Logic Operators

- Comparison:

Operator	Description
=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<> *OR* !=	Not equal to (depends on DBMS as to which is used)

- Logic:
  - AND, NOT, OR
  - **Example**:

```
SELECT *
FROM Furniture
WHERE ((Type = 'Chair' AND Colour = 'Black')
        OR
        (Type = 'Lamp' AND Colour = 'Black')
       );
```

Some Useful String Functions

- UPPER()
  - Change to upper case
- LOWER()
  - Change to lower case
- LEFT()
  - Take the left X characters from a string
- RIGHT()
  - Take the X right characters from a string

- Many more examples are in the labs!

Set Operations

- UNION
  - Shows all rows returned from the queries (or tables)
- INTERSECT
  - Shows only rows that are common in the queries (or the tables)
- [UNION/INTERSECT] ALL
  - If you want duplicate rows shown in the results you need to use the ALL keyword.. UNION ALL etc.

- In MySQL only UNION and UNION ALL are supported

UNION Example

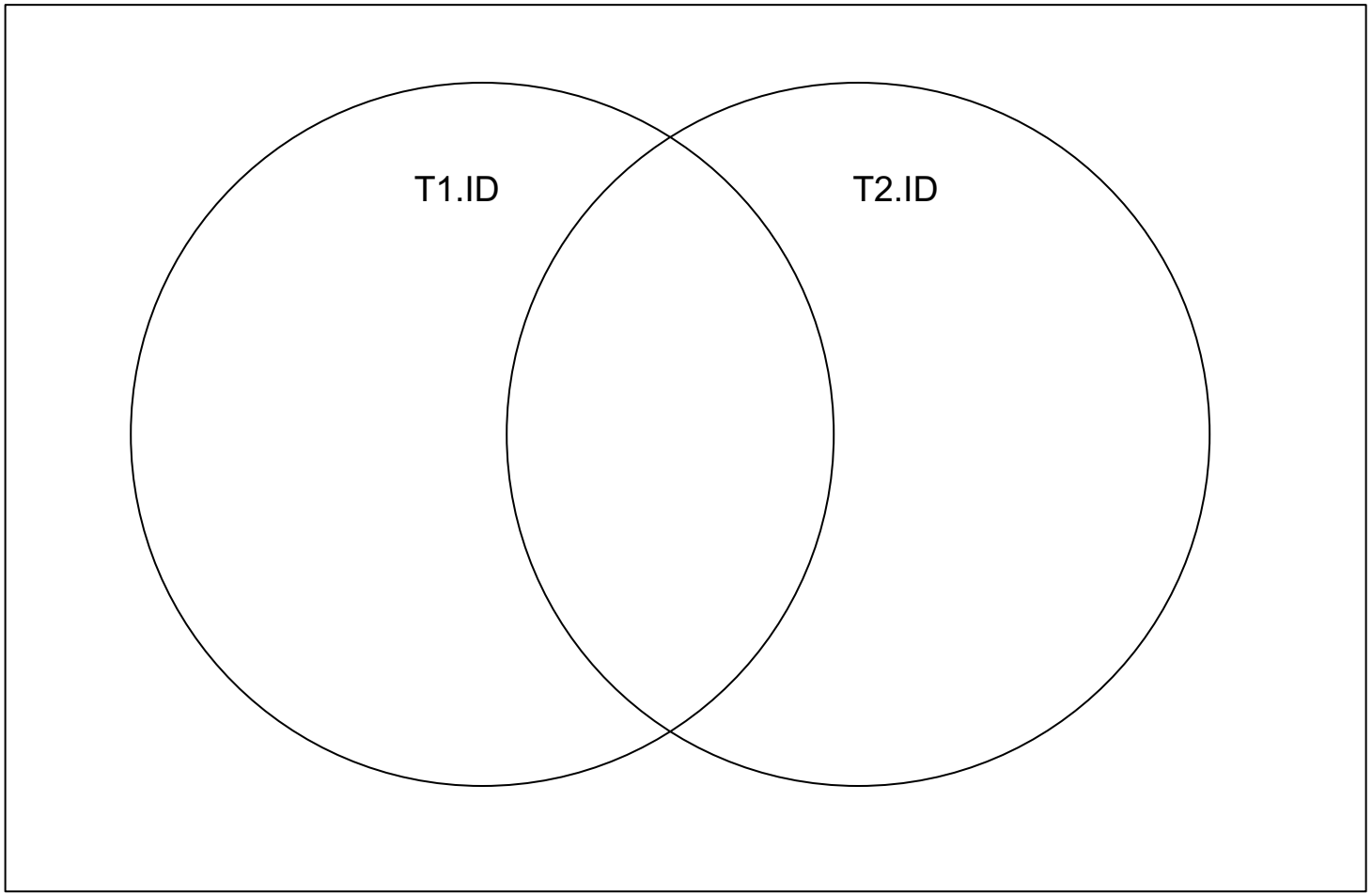
```
SELECT   Employee.Name, EmployeeType
         FROM Employee INNER JOIN Hourly
         ON   Employee.ID = Hourly.ID
UNION
SELECT   Employee.Name, EmployeeType
         FROM Employee INNER JOIN Salaried
         ON   Employee.ID = Salaried.ID;
```

Output | Snippets | Query 1 Result | Lecture7.sql Result

Fetched 4 records. Durati

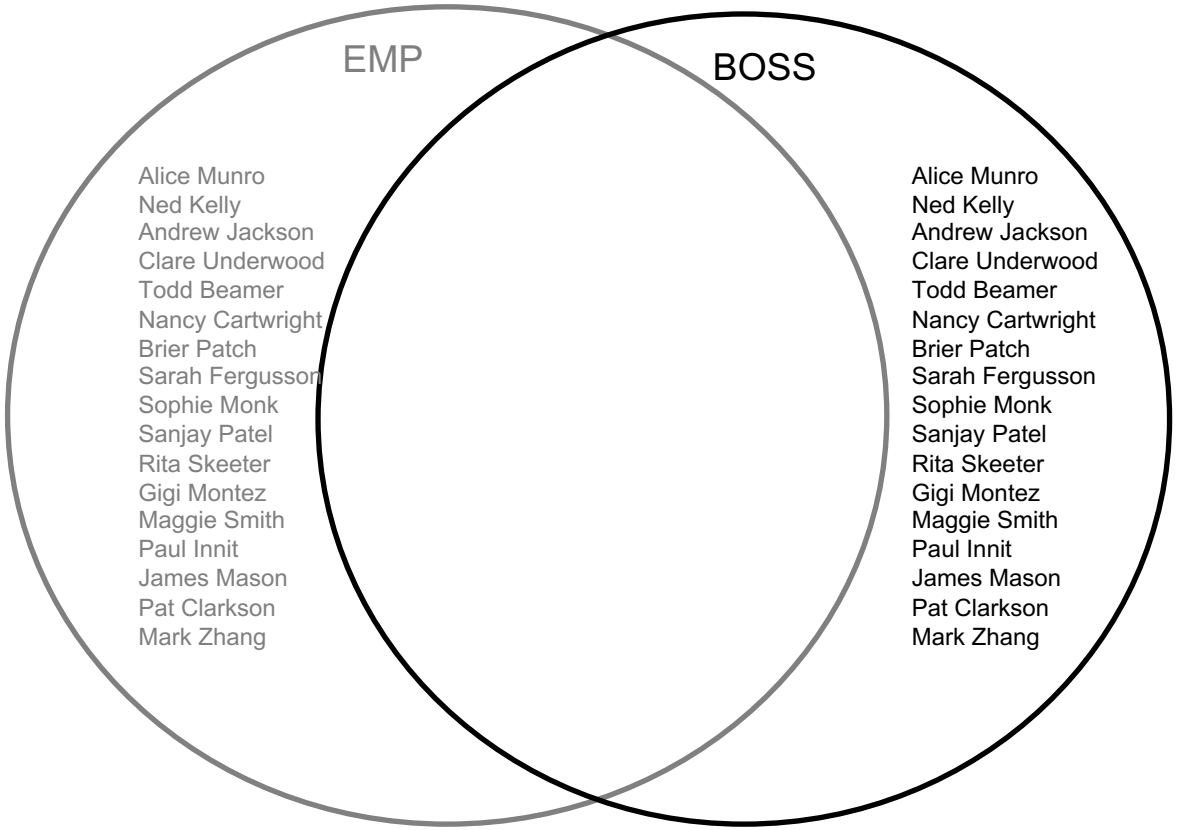
Name	EmployeeType
Alice	H
Alan	H
Sean	S
Linda	S

JOINS depicted as Venn Diagrams



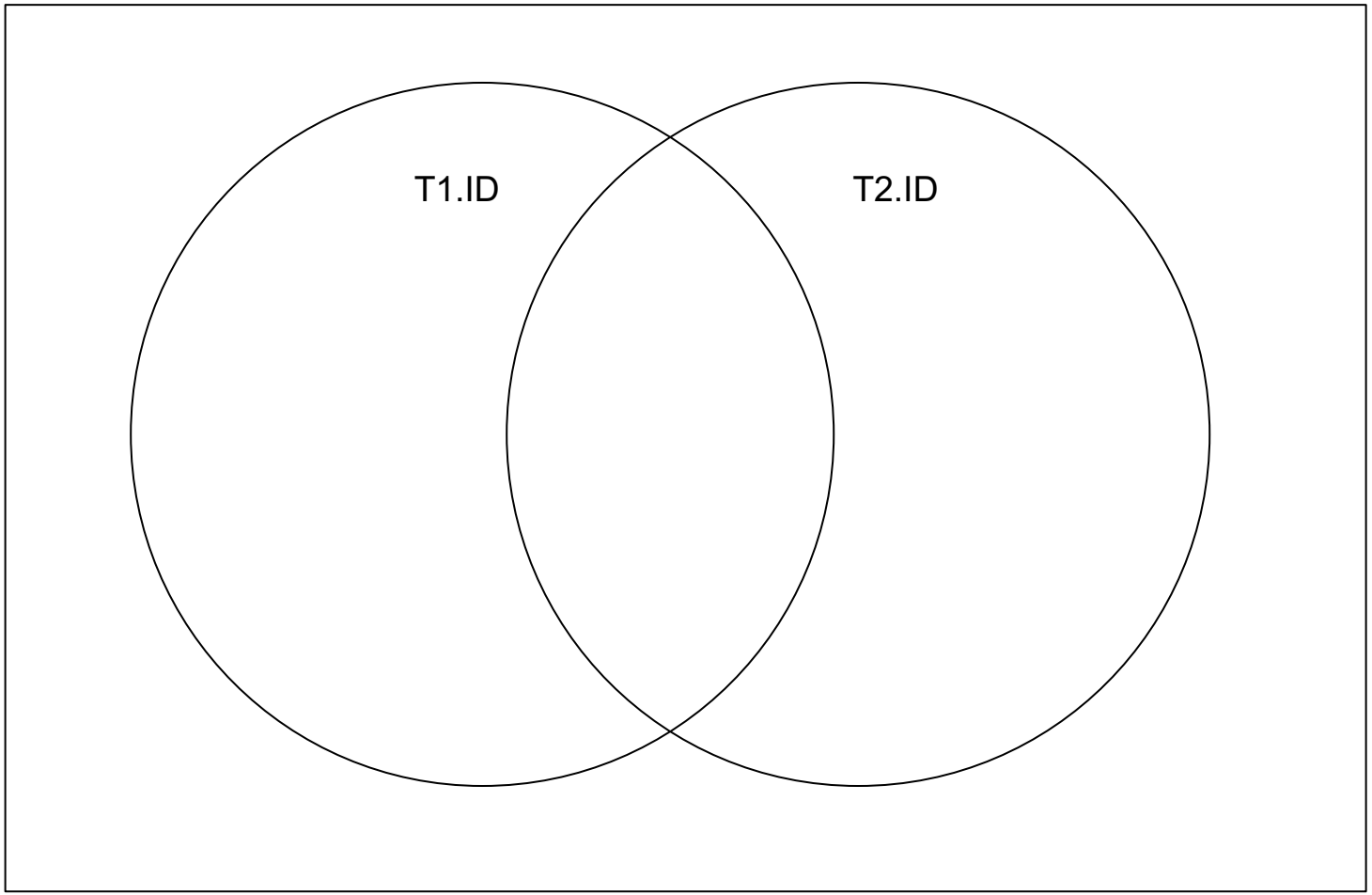
Inner Join – Labs2018 demo

Unary join of the Employee table aliased as EMP, BOSS



JOINS depicted as Venn Diagrams

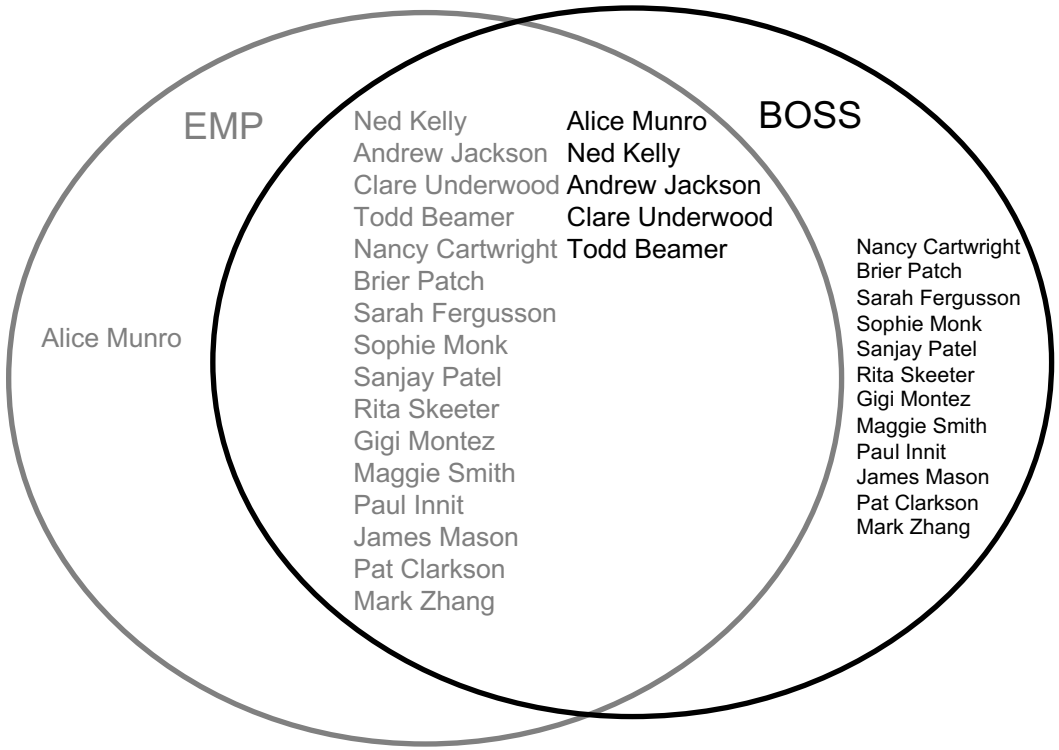
- T1 INNER JOIN T2 ON T1.ID = T2.ID
- T1 NATURAL JOIN T2



Inner Join – Labs2018 demo

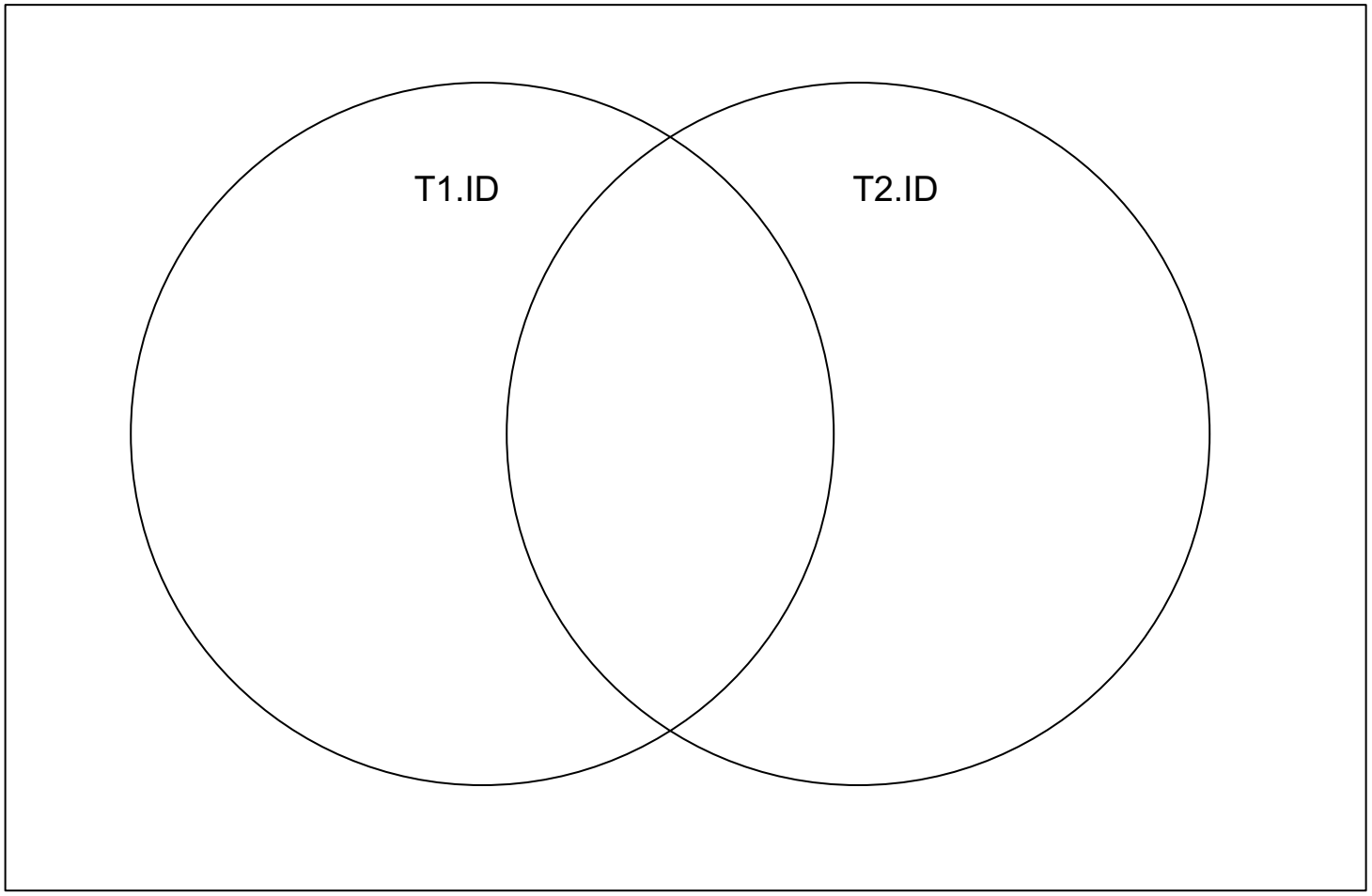
SELECT emp.firstname as efirst, emp.lastname as elast,
boss.firstname as bfirst, boss.lastname as blast
FROM employee emp **INNER JOIN** employee boss
ON emp.bossid = boss.employeeid;

efirst	elast	bfirst	blast
Ned	Kelly	Alice	Munro
Andrew	Jackson	Ned	Kelly
Clare	Underw…	Ned	Kelly
Todd	Beamer	Alice	Munro
Nancy	Cartwright	Todd	Beamer
Brier	Patch	Alice	Munro
Sarah	Fergusson	Brier	Patch
Sophie	Monk	Alice	Munro
Sanjay	Patel	Andrew	Jackson
Rita	Skeeter	Clare	Underw…
Gigi	Montez	Clare	Underw…
Maggie	Smith	Clare	Underw…
Paul	Innit	Andrew	Jackson
James	Mason	Andrew	Jackson
Pat	Clarkson	Andrew	Jackson
Mark	Zhang	Andrew	Jackson



JOINS depicted as Venn Diagrams

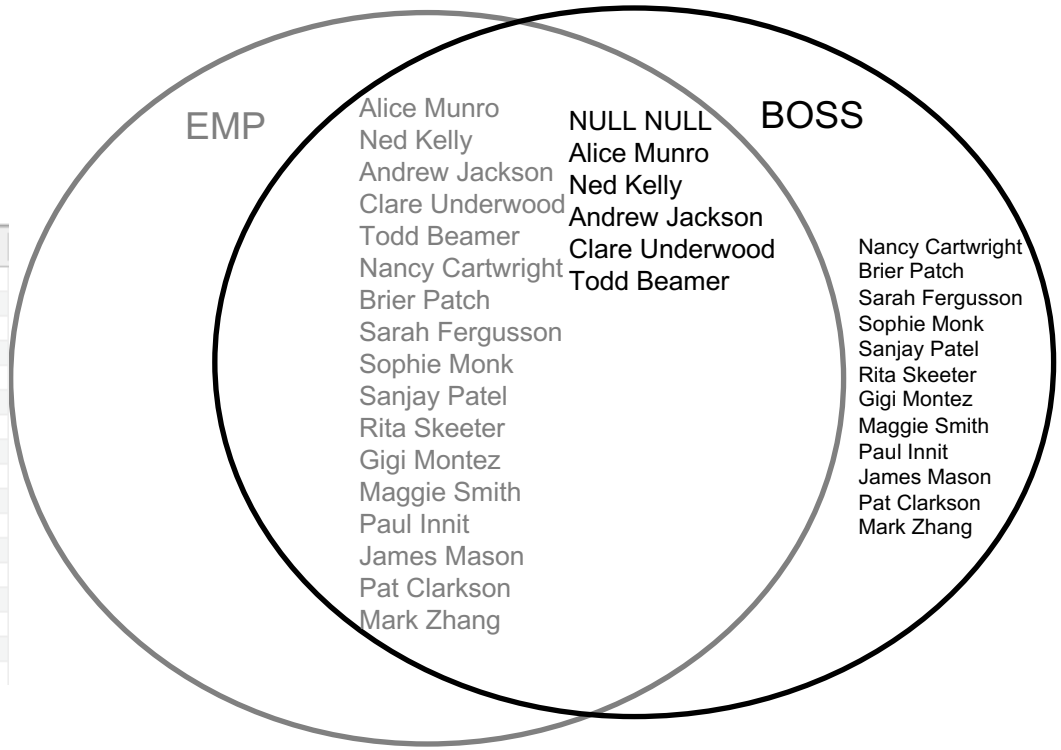
- T1 LEFT OUTER JOIN T2 ON T1.ID = T2.ID



LEFT OUTER Join – Labs2018 demo

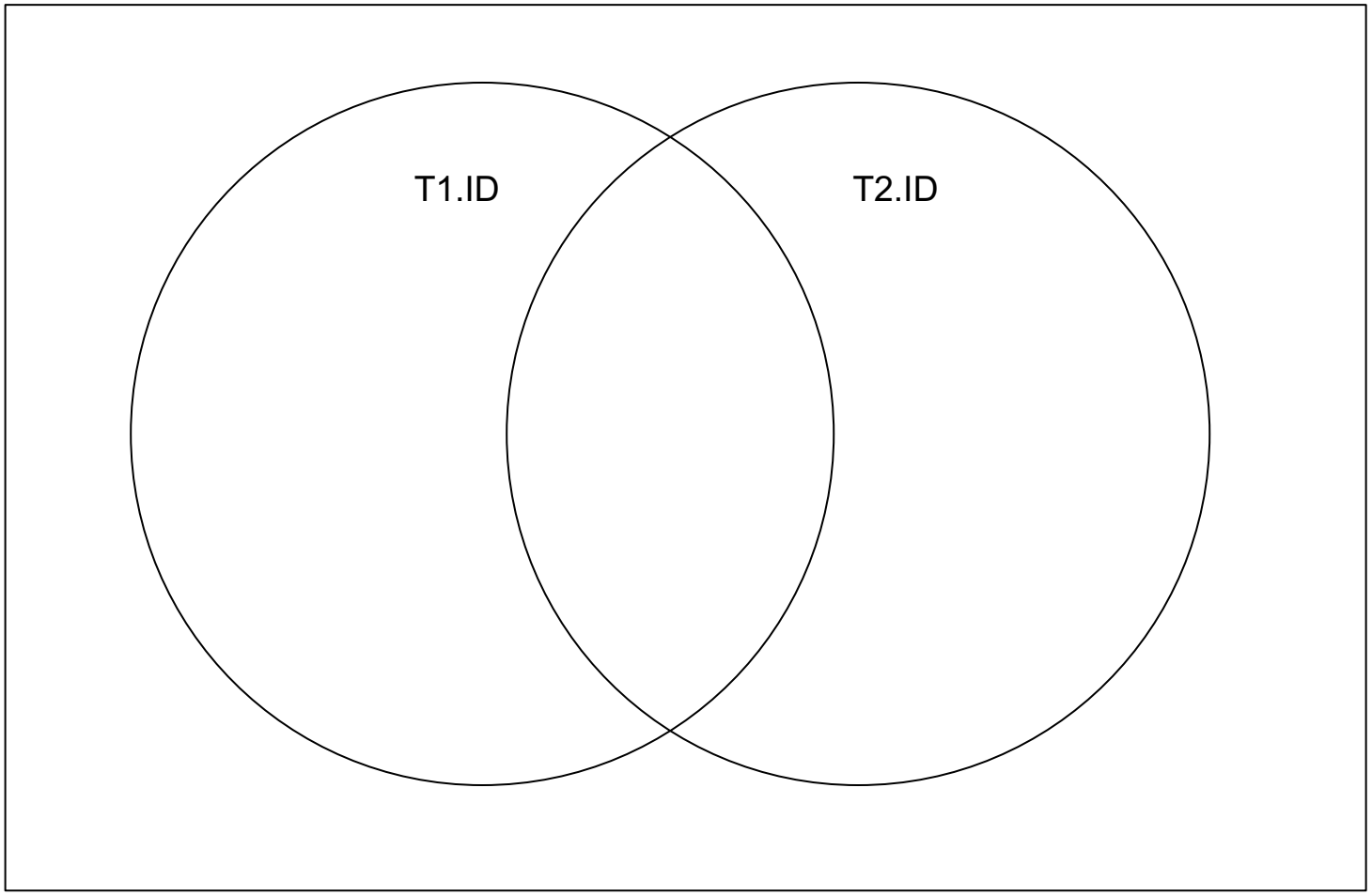
SELECT emp.firstname as efirst, emp.lastname as elast,
boss.firstname as bfirst, boss.lastname as blast
FROM employee emp **LEFT OUTER JOIN** employee boss
ON emp.bossid = boss.employeeid;

efirst	elast	bfirst	blast
Alice	Munro	NULL	NULL
Ned	Kelly	Alice	Munro
Andrew	Jackson	Ned	Kelly
Clare	Underwood	Ned	Kelly
Todd	Beamer	Alice	Munro
Nancy	Cartwright	Todd	Beamer
Brier	Patch	Alice	Munro
Sarah	Fergusson	Brier	Patch
Sophie	Monk	Alice	Munro
Sanjay	Patel	Andrew	Jackson
Rita	Skeeter	Clare	Underwood
Gigi	Montez	Clare	Underwood
Maggie	Smith	Clare	Underwood
Paul	Innit	Andrew	Jackson
James	Mason	Andrew	Jackson
Pat	Clarkson	Andrew	Jackson
Mark	Zhang	Andrew	Jackson



JOINS depicted as Venn Diagrams

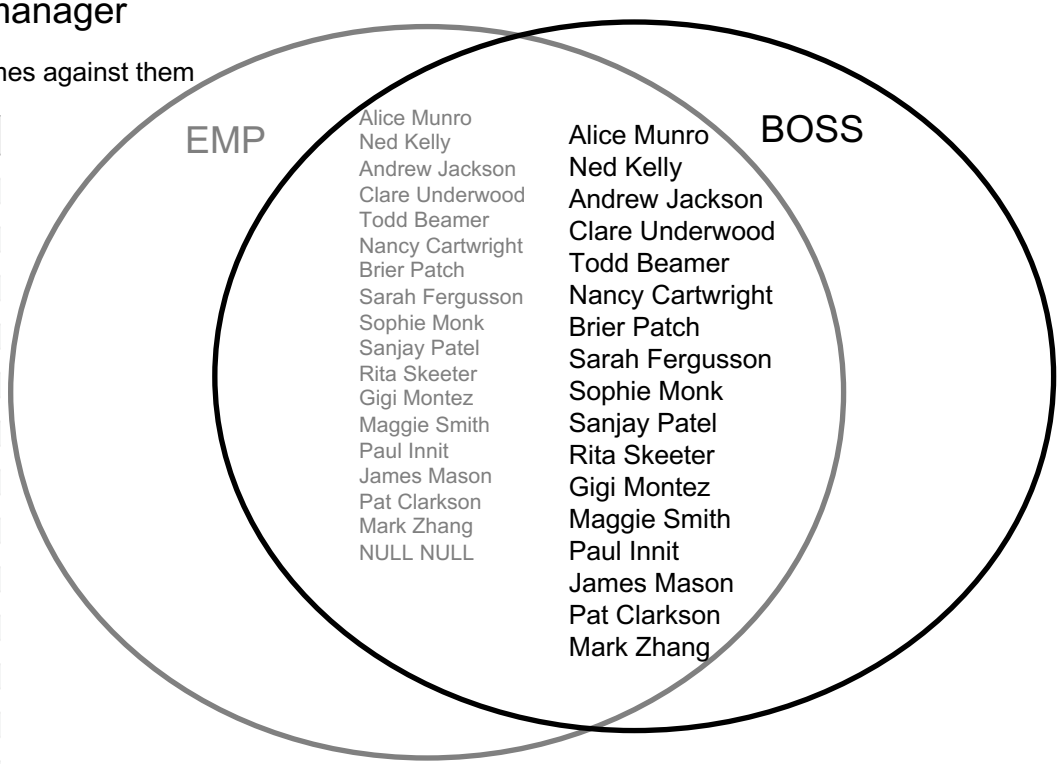
- T1 RIGHT OUTER JOIN T2 ON T1.ID = T2.ID



Right Outer Join – Labs2018 demo

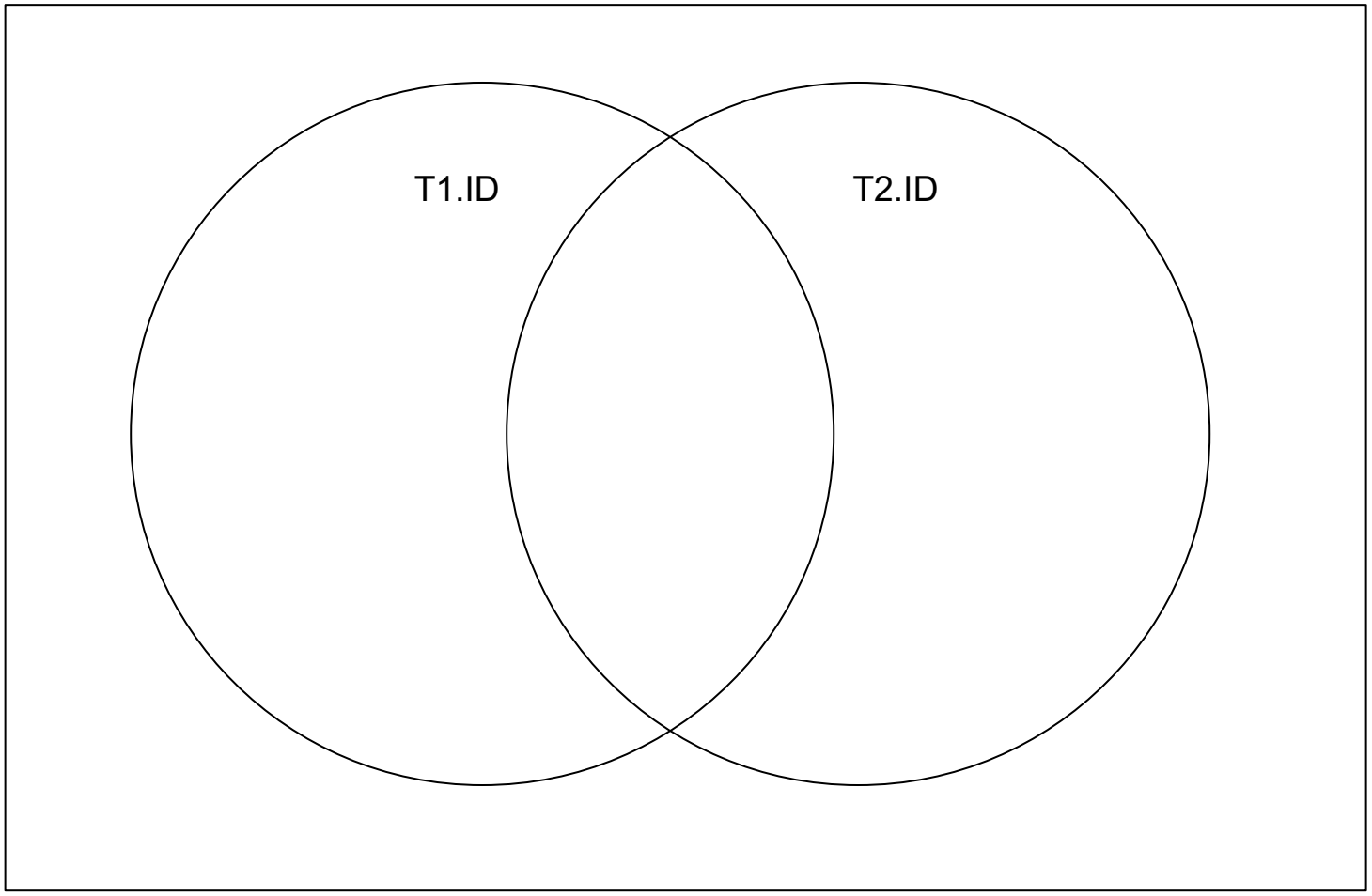
```
SELECT emp.firstname as efirst, emp.lastname as elast,
boss.firstname as bfirst, boss.lastname as blast
FROM employee emp RIGHT OUTER JOIN employee boss
ON emp.bossid = boss.employeeid;
-- Asks list every employee of every manager
-- Hence Sarah, Sophie, Rita, Mark etc have no names against them
```

efirst	elast	bfirst	blast
Ned	Kelly	Alice	Munro
Todd	Beamer	Alice	Munro
Brier	Patch	Alice	Munro
Sophie	Monk	Alice	Munro
Andrew	Jackson	Ned	Kelly
Clare	Underwood	Ned	Kelly
Sanjay	Patel	Andrew	Jackson
Paul	Innit	Andrew	Jackson
James	Mason	Andrew	Jackson
Pat	Clarkson	Andrew	Jackson
Mark	Zhang	Andrew	Jackson
Rita	Skeeter	Clare	Underwood
Gigi	Montez	Clare	Underwood
Maggie	Smith	Clare	Underwood
Nancy	Cartwright	Todd	Beamer
NULL	NULL	Nancy	Cartwright
Sarah	Fergusson	Brier	Patch
NULL	NULL	Sarah	Fergusson
NULL	NULL	Sophie	Monk
NULL	NULL	Sanjay	Patel
NULL	NULL	Rita	Skeeter
NULL	NULL	Gigi	Montez
NULL	NULL	Maggie	Smith
NULL	NULL	Paul	Innit
NULL	NULL	James	Mason
NULL	NULL	Pat	Clarkson
NULL	NULL	Mark	Zhang



JOINS depicted as Venn Diagrams

- T1 FULL OUTER JOIN T2 ON T1.ID = T2.ID

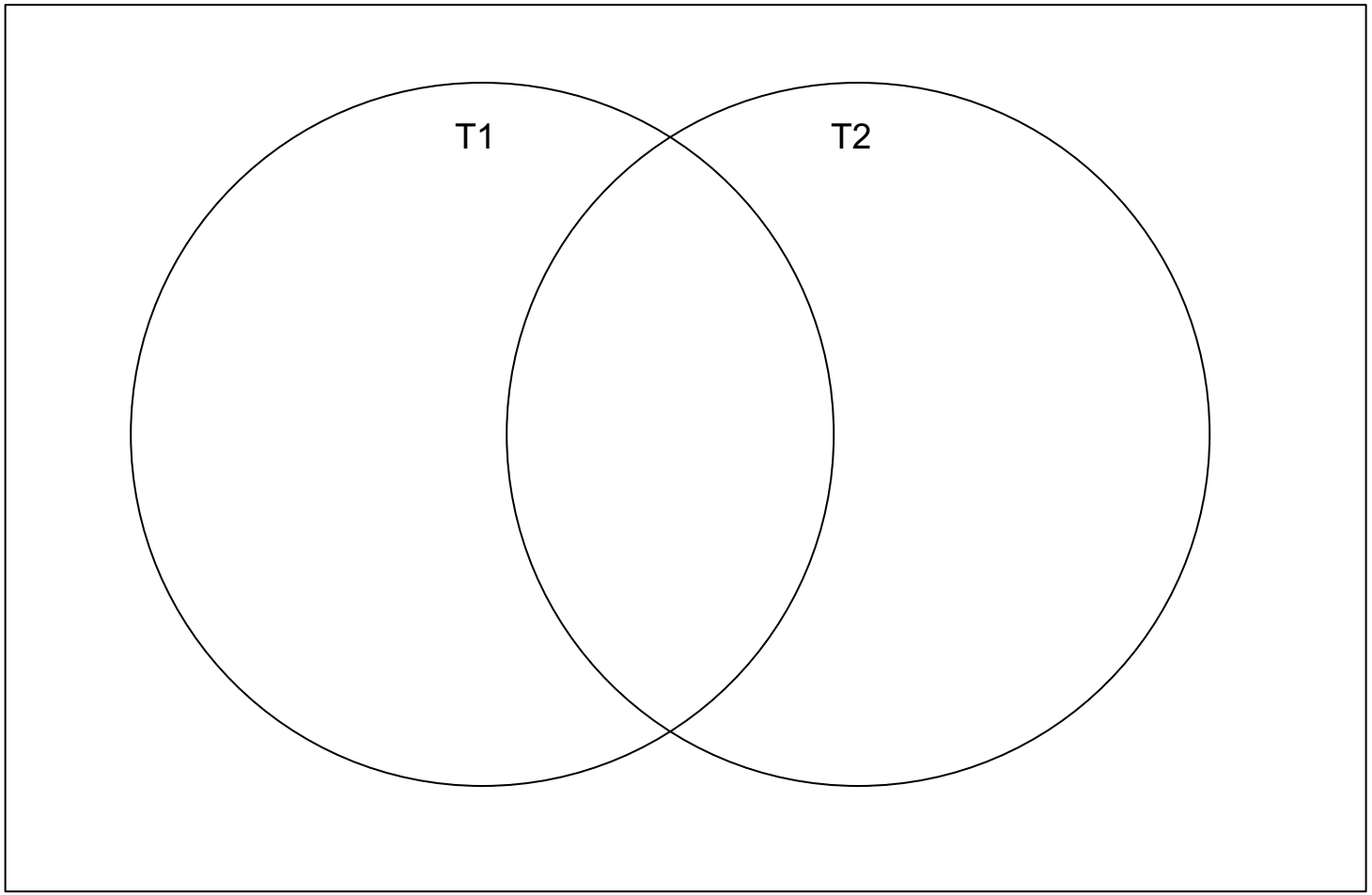


FULL Outer Join – NO Labs2018 demo

- MySQL DBMS server does not support full outer joins

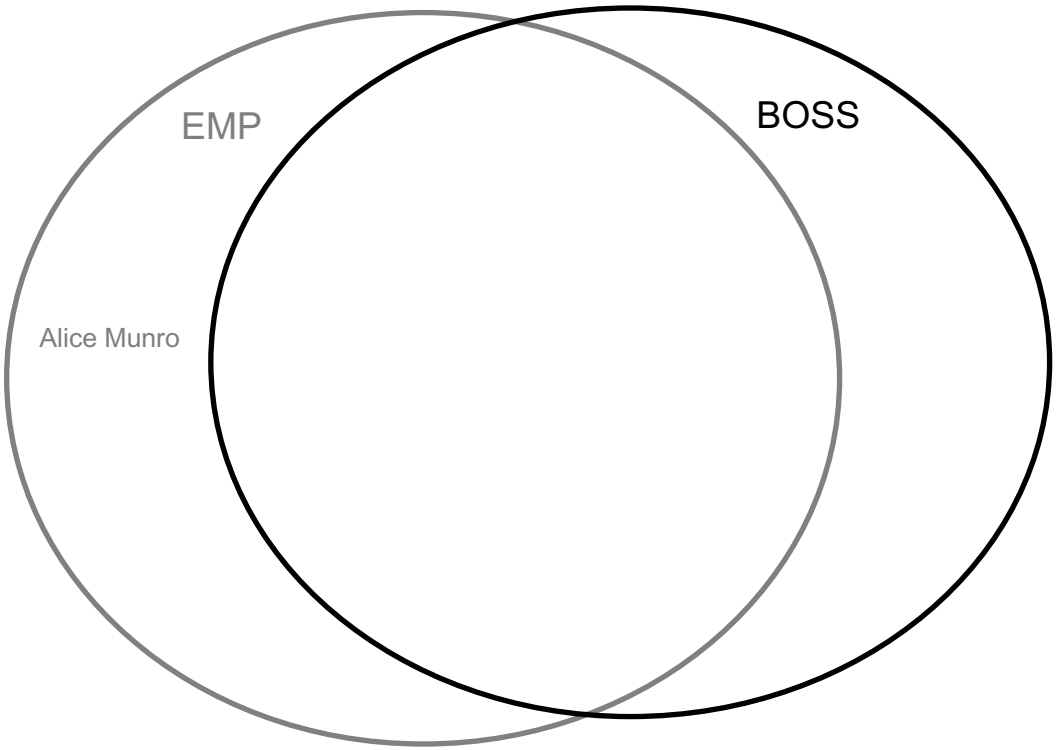
JOINS depicted as Venn Diagrams

- T1 LEFT OUTER JOIN T2 ON T1.ID = T2.ID
  WHERE T2.ID IS NULL



Inner Join – Labs2018 demo

SELECT emp.firstname as efirst, emp.lastname as elast,
boss.firstname as bfirst, boss.lastname as blast
FROM employee emp **LEFT OUTER JOIN** employee boss
ON emp.bossid = boss.employeeid
WHERE boss.employeeid **IS NULL;**



More on INSERT

- Inserting records from a table:
  - Note: table must already exist

```
INSERT INTO NewEmployee
    SELECT * FROM Employee;
```

- Multiple record inserts:

All columns must be inserted

```
INSERT INTO Employee VALUES
    (DEFAULT, "A", "A's Addr", "2012-02-02", NULL, "S"),
    (DEFAULT, "B", "B's Addr", "2012-02-02", NULL, "S"),
    (DEFAULT, "C", "C's Addr", "2012-02-02", NULL, "S");
```

Specific columns will be inserted

```
INSERT INTO Employee
    (Name, Address, DateHired, EmployeeType)
    VALUES
        ("D", "D's Addr", "2012-02-02", "C"),
        ("E", "E's Addr", "2012-02-02", "C"),
        ("F", "F's Addr", "2012-02-02", "C");
```

The UPDATE Statement

- Changes *existing* data in tables
  - Order of statements is important
  - Specifying a WHERE clause is important
    - Unless you want it to operate on the whole table

```
UPDATE Hourly
    SET HourlyRate = HourlyRate * 1.10;
```

- **Example**: *Increase all salaries greater than $100000 by 10% and all other salaries by 5%*

```
UPDATE Salaried
    SET AnnualSalary = AnnualSalary * 1.05
    WHERE AnnualSalary <= 100000;
UPDATE Salaried
    SET AnnualSalary = AnnualSalary * 1.10
    WHERE AnnualSalary > 100000;
```

Any problems with this?

The UPDATE Statement: CASE

- A better solution in this case is to use the **CASE** command

```
UPDATE Salaried
    SET AnnualSalary =
        CASE
            WHEN AnnualSalary <= 100000
            THEN AnnualSalary * 1.05
            ELSE AnnualSalary * 1.10
        END;
```

If salary is lower than 100000 increase it by 5%, otherwise increase it by 10%

DELETE, REPLACE

- REPLACE
  - REPLACE works identically as INSERT
    - Except if an old row in a table has a key value the same as the new row then it is overwritten…
- DELETE
  - The DANGEROUS command – deletes *ALL* records

```
DELETE FROM Employee;
```

    - The better version (unless you are really, really sure)

```
DELETE FROM Employee
    WHERE Name = "Grace";
```

  - Be aware of the foreign key constraints
    - ON DELETE CASCADE or ON DELETE RESTRICT (lab practice)

What is examinable

- You need to be able to write SQL
  - SELECT
  - DML (INSERT, UPDATE, DELETE)
  - DDL (CREATE, ALTER, DROP)

Next Time

- More SQL (SQL 3, Lecture Overflow)