INFO20003 Database Systems

Dr Renata Borovica-Gajic

Lecture 9
SQL Summary

Semester 1 2018, Week 5

Homework solutions

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

$$\rho_{sname}(\sigma_{rating>9}(Sailors))$$

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

$$\rho_{sname}(Sailors \bowtie \sigma_{day<'11/1/96'}(Reserves))$$

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

$$\rho_{bname}(Boats \bowtie Reserves)$$

Homework solutions

4. Find all pairs of sailors with the same rating

$$r\ (S1(1 \rightarrow sid1, 2 \rightarrow sname1, 3 \rightarrow rating1, 4 \rightarrow age1), Sailors)$$

$$r\ (S2(1 \rightarrow sid2, 2 \rightarrow sname2, 3 \rightarrow rating2, 4 \rightarrow age2), Sailors)$$

$$\rho_{sname1,\,sname2}(S1 \bowtie_{rating1=rating2 \wedge sid1 \neq sid2} S2)$$

Assignment 1 questions

- Do I need to make assumptions about **all** possible extensions to the requirements in the future?
  - No. Your model should capture the existing description of the problem
- What makes the best solution?
  - A solution that is valid (captures the existing set of requirements) but is *flexible* that for each requirement will allow some *minor* extensions.
  - E.g. Instead of 3 reaction gravities (strong, moderate, weak), you get an additional one
- Do I need to use supertype/subtype to score the highest?
  - No. If your model is valid and flexible – you will get the highest marks in both cases (with or without hierarchies)

Assignment 1: Examples for marks deduction

Examples for marks deduction:

- Entity / Attribute incorrect or missing
- Relationship cardinality incorrect (e.g. one - many)
- Poor naming of object (e.g. T1, T2)
- Wrong data type (e.g. varchar for placing ‘age’)
- Incorrect primary or foreign key
- Not Null is wrong
- Unresolved M-M exists, or associative entity is incorrect
- Business rules can’t be supported (e.g. one medicare card cannot capture the entire family)

NOTE: These are EXAMPLES, not a contract set in stone

Coverage

- 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

- 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
- Table names are Operating System Sensitive
  - If case sensitivity exists in the operating system, then the table names are case sensitive! (i.e. Mac, Linux)
    - Account <> ACCOUNT
- Field names are case insensitive
  - ACCOUNTID == AccountID == AcCoUnTID
- You can do maths in SQL…
  - SELECT 1*1+1/1-1;

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"))

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

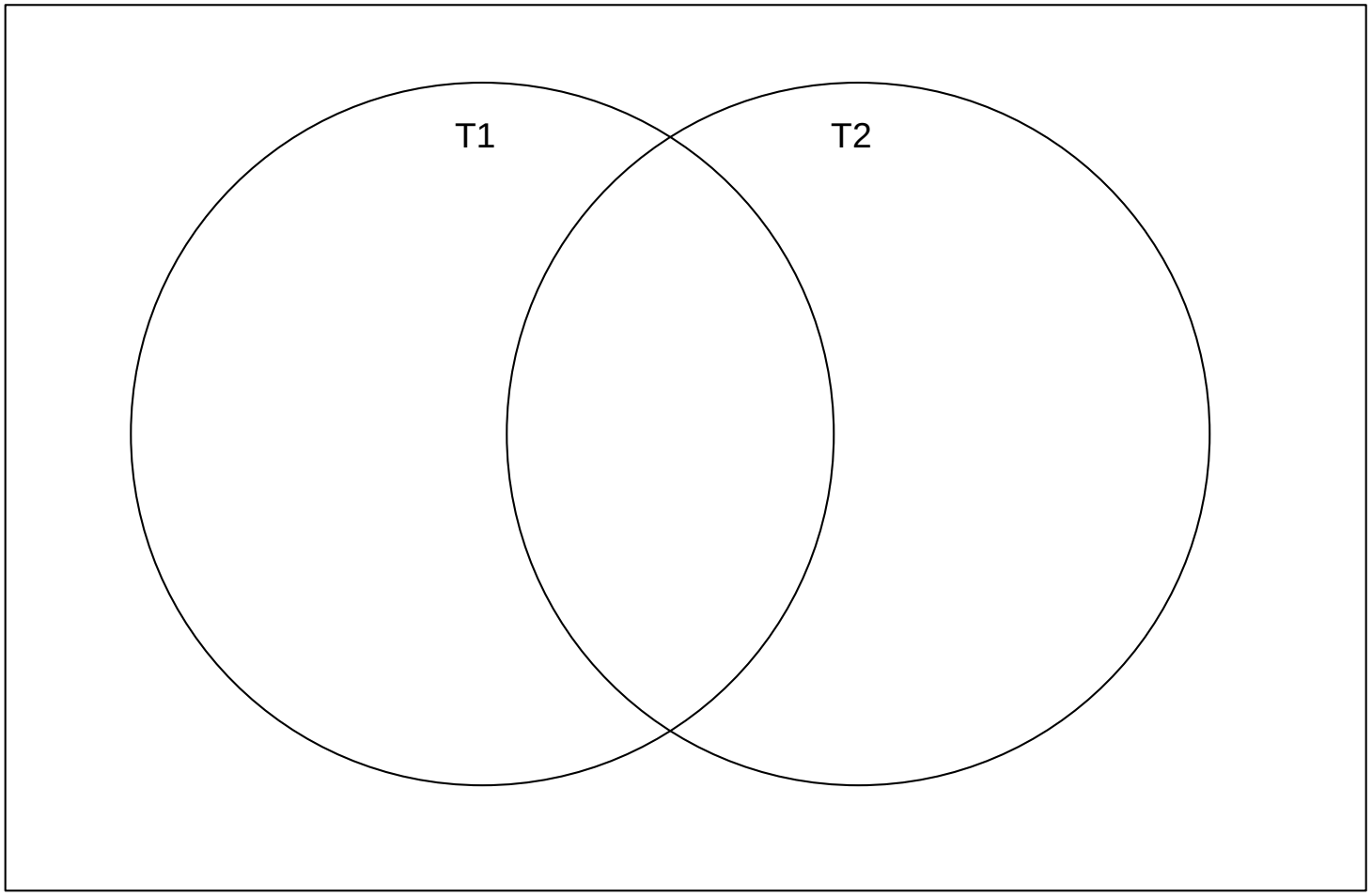
```
SELECT   Employee.Name, EmployeeType
         FROM Employee INNER JOIN Hourly
         ON   Employee.ID = Hourly.ID
UNION
SELECT   Employee.Name, EmployeeType
         FROM Employee INNER JOIN Salaried
         ON   Employee.ID = Salaried.ID;
```

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

Fetched 4 records. Durati

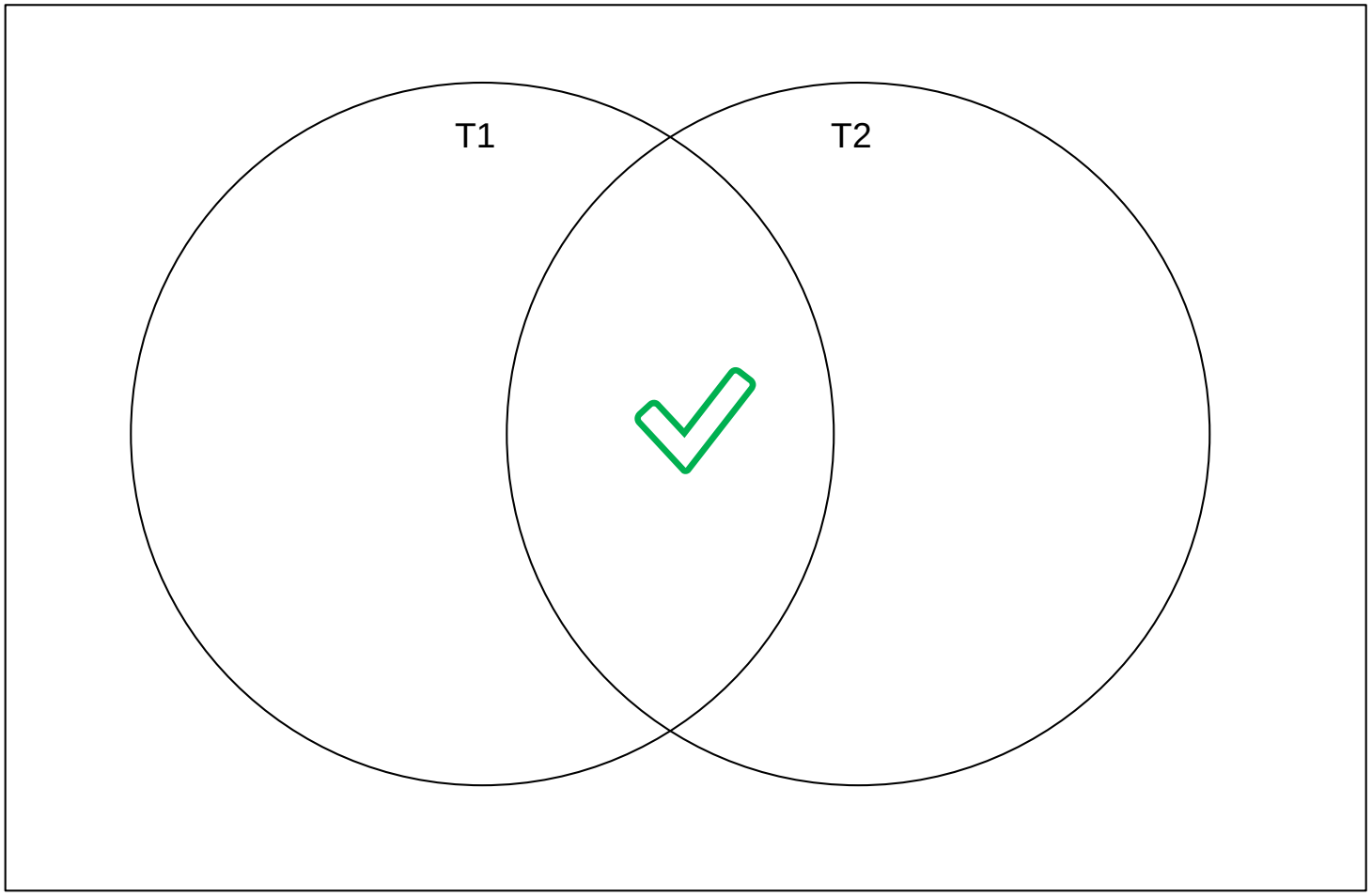
Name	EmployeeType
Alice	H
Alan	H
Sean	S
Linda	S

JOINS depicted as Venn Diagrams



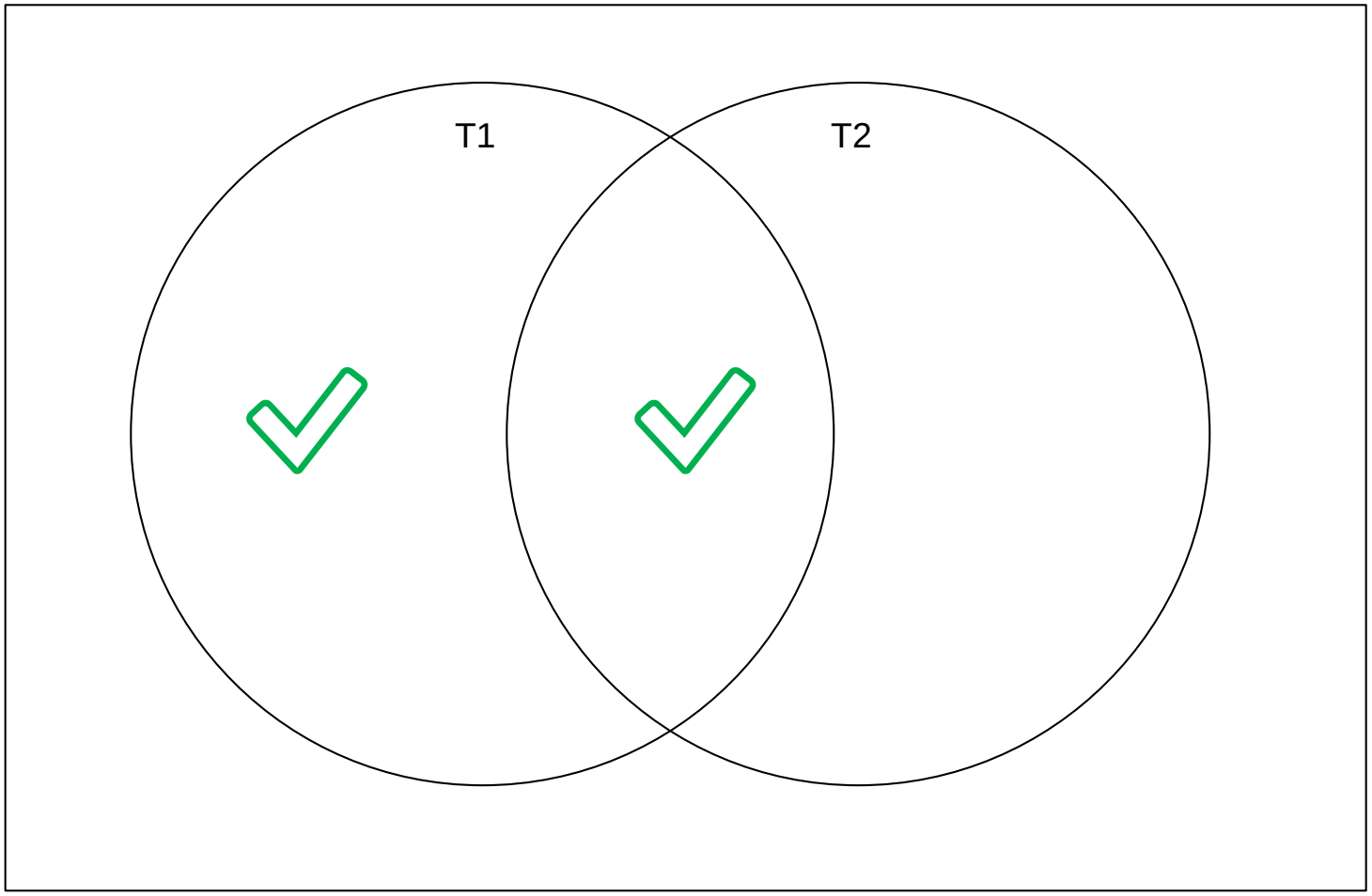
JOINS depicted as Venn Diagrams

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



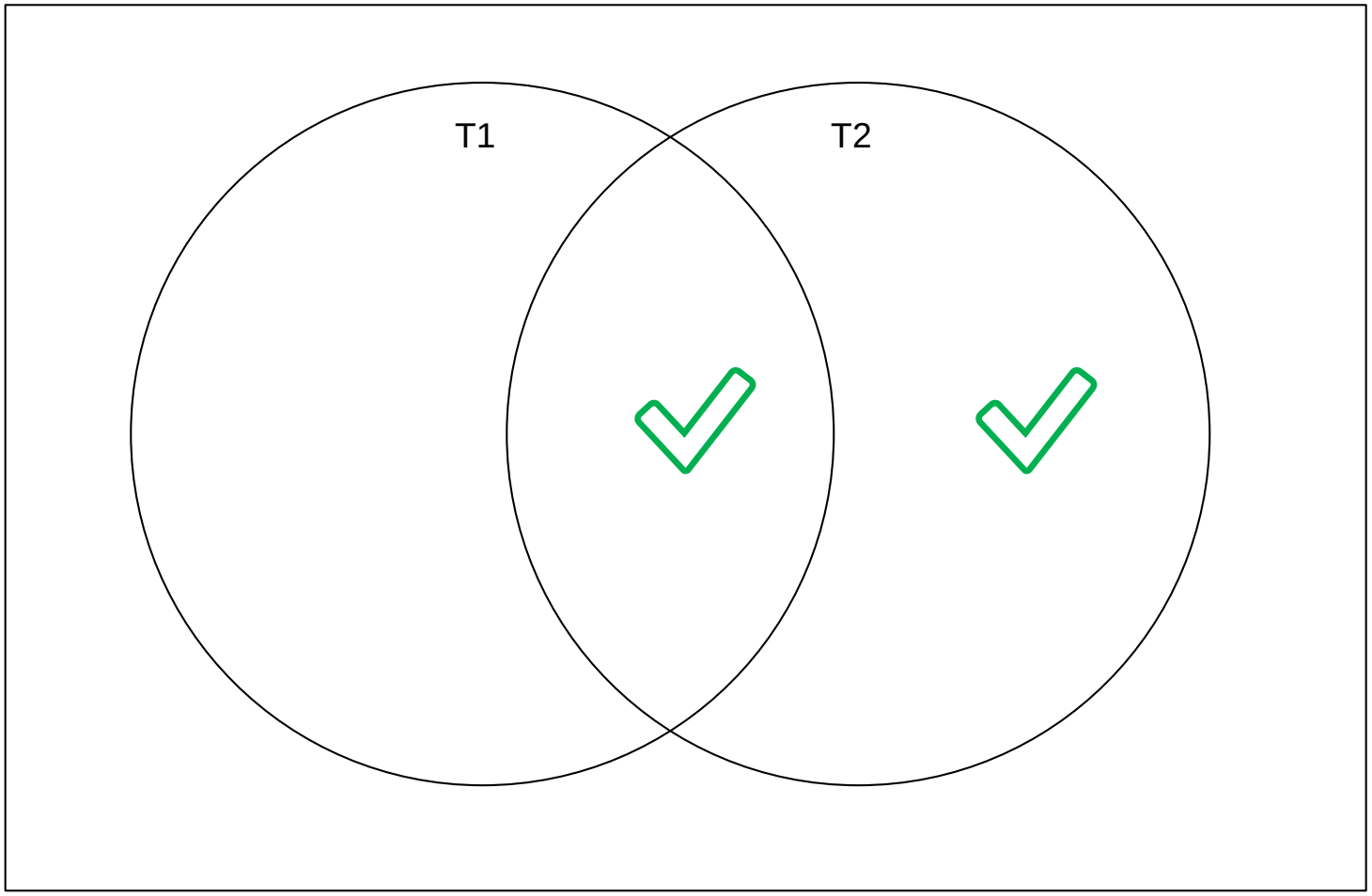
JOINS depicted as Venn Diagrams

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



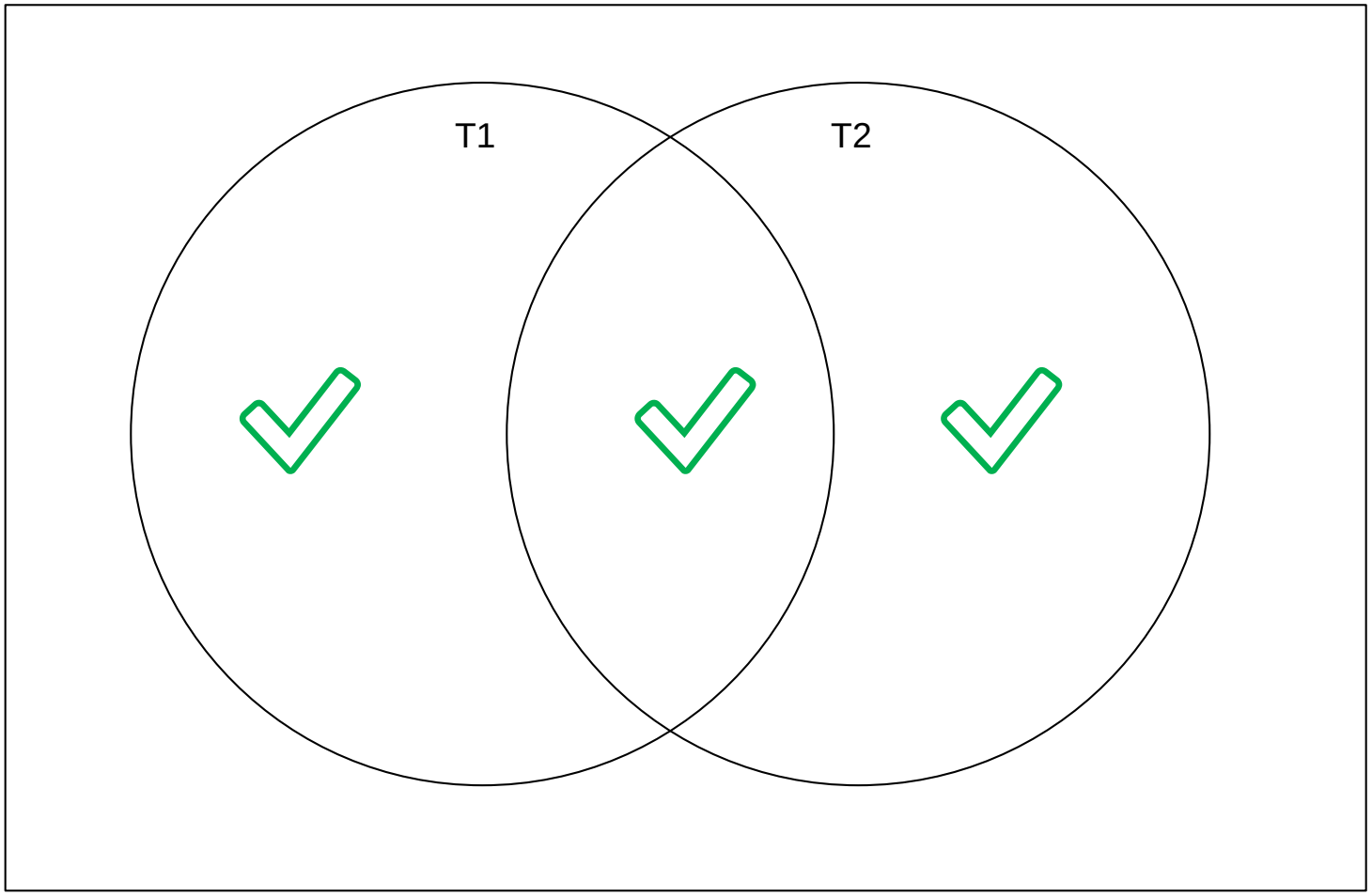
JOINS depicted as Venn Diagrams

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



JOINS depicted as Venn Diagrams

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



Query Nesting

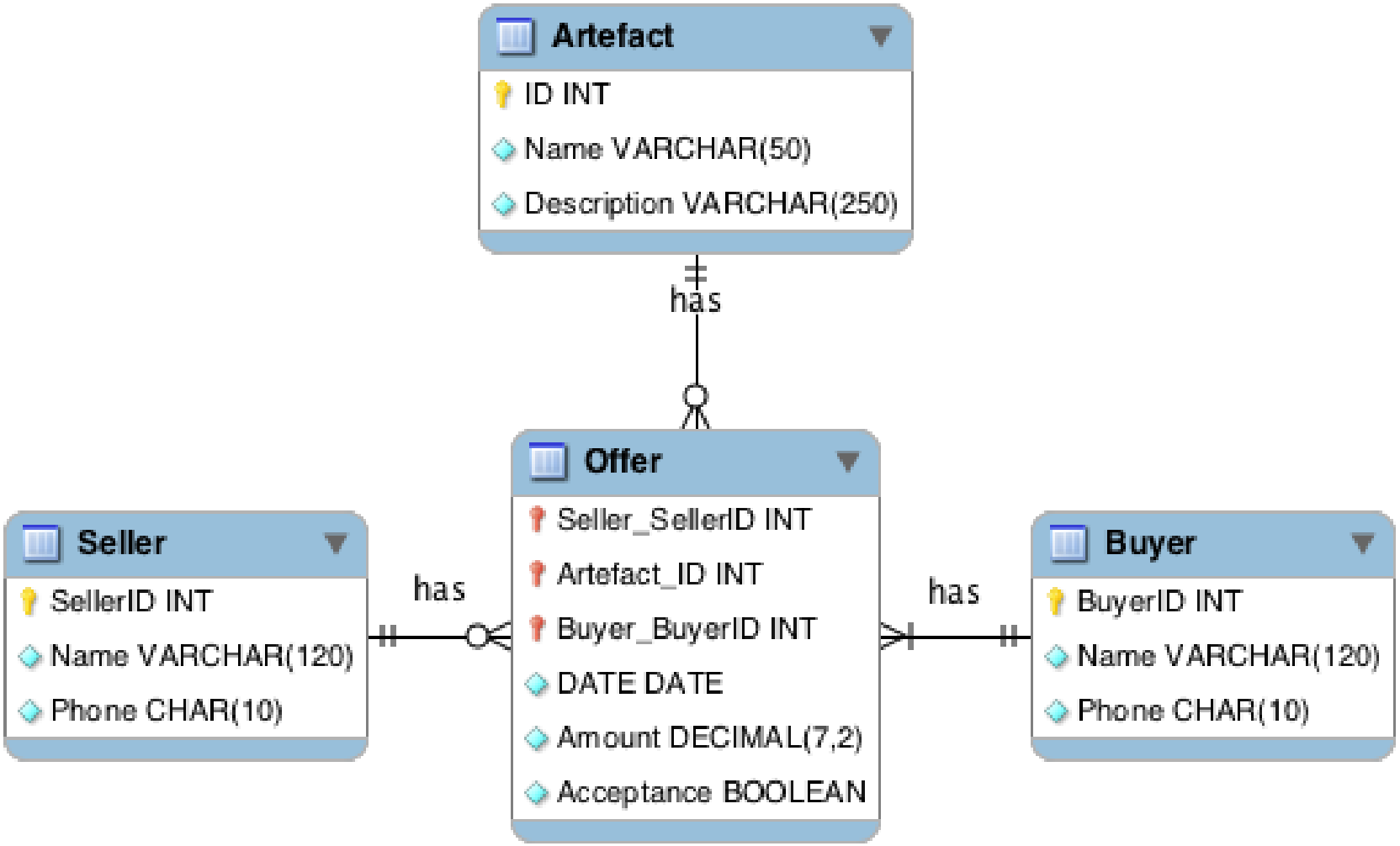
- SQL provides the ability to *nest* subqueries
- A nested query is simply another select query you write to produce a table set
  - Remember that all select queries return a table set of data
- A common use of subqueries is to perform set tests
  - Set membership, set comparisons

Sub-Query Comparison Operators

- IN / NOT IN
  - Used to test whether the attribute is IN/NOT IN the subquery list
- ANY
  - True if any value returned meets the condition
- ALL
  - True if all values returned meet the condition

- For more info: https://www.w3schools.com/sql/sql_any_all.asp

Auction Bids – Physical Model



Tables

Artefact

ID	Name	Description
1	Vase	Old Vase
2	Knife	Old Knife
3	Pot	Old Pot

Seller

SellerID	Name	Phone
1	Abby	0233232232
2	Ben	0311111111
3	Carl	0333333333

Buyer

BuyerID	Name	Phone
1	Maggie	0333333333
2	Nicole	0444444444
3	Oleg	0555555555

Offer

SellerID	ArtefactID	BuyerID	Date	Amount	Acceptance
1	1	1	2012-06-20	81223.23	N
1	1	2	2012-06-20	82223.23	N
2	2	1	2012-06-20	19.95	N
2	2	2	2012-06-20	23.00	N

Example: Subquery

- List the BuyerID, Name and Phone number for all bidders on artefact 1

```
SELECT * FROM Buyer
    WHERE BuyerID IN
        (SELECT BuyerID FROM Offer WHERE ArtefactID = 1)
```

BuyerID	Name	Phone
1	Maggie	0333333333
2	Nicole	0444444444

More examples using subqueries

- Which Artefacts don't have offers made on them

```
SELECT * FROM Artefact
    WHERE ID NOT IN
        (SELECT ArtefactID FROM Offer);
```

ID	Name	Description
3	Pot	Old Pot

- Which Buyers haven't made a bid for Artefact 3

```
SELECT * FROM Buyer
    WHERE BuyerID NOT IN
        (SELECT BuyerID FROM Offer
            WHERE ArtefactID = 3);
```

BuyerID	Name	Phone
1	Maggie	0333333333
2	Nicole	0444444444
3	Oleg	0555555555

You try …

Which Buyers haven't made a bid for the "Pot" Artefact ?

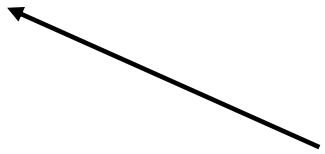
Do we need to use IN? Is there another way…

- List the BuyerID, Name and Phone number for all bidders on artefact 1

```
SELECT * FROM Buyer
WHERE BuyerID IN (SELECT BuyerID FROM Offer
                  WHERE ArtefactID = 1)
```

Equals to

```
SELECT BuyerID, Name and Phone
FROM Buyer NATURAL JOIN Offer
WHERE ArtefactID = 1
```



This is a more efficient way

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)

Views

- Any relation that is not in the physical models, but is made available to the "user" as a virtual relation is called a view.
- Views are good because:
  - They help hide the query complexity from users
  - They help hide data from users
    - Different users use different views
      - Prevents someone from accessing the employee tables to see salaries for instance
    - One way of improving database security
- Create view statement:

  CREATE VIEW nameofview **AS** validsqlstatement

- Once a view is defined
  - Its definition is stored in the database (not the data, but metadata – schema information)
  - Can be used just like any other table

Create View Example

```
CREATE VIEW EmpPay AS
SELECT   Employee.ID, Employee.Name, DateHired,
         EmployeeType, HourlyRate AS Pay
         FROM Employee INNER JOIN Hourly
         ON   Employee.ID = Hourly.ID
UNION
SELECT   Employee.ID, Employee.Name, DateHired,
         EmployeeType, AnnualSalary  AS Pay
         FROM Employee INNER JOIN Salaried
         ON   Employee.ID = Salaried.ID
UNION
SELECT   Employee.ID, Employee.Name, DateHired,
         EmployeeType, BillingRate  AS Pay
         FROM Employee INNER JOIN Consultant
         ON     Employee.ID = Consultant.ID;
```

Using a View

```
SELECT * FROM EmpPay;
```

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

ID	Name	DateHired	EmployeeType	Pay
3	Alice	2012-12-02	H	23.43
4	Alan	2010-01-22	H	29.43
1	Sean	2012-02-02	S	92000.00
2	Linda	2011-06-12	S	92300.00
5	Peter	2010-09-07	C	210.00
6	Rich	2012-05-19	C	420.00

```
SELECT * FROM EmpPay
    WHERE EmployeeType = "H" OR EmployeeType = "C"
```

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

Fetched 4 records. Duration: 0.000 sec, fetc

ID	Name	DateHired	EmployeeType	Pay
3	Alice	2012-12-02	H	23.43
4	Alan	2010-01-22	H	29.43
5	Peter	2010-09-07	C	210.00
6	Rich	2012-05-19	C	420.00

More DDL Commands

- There are more than CREATE!
- ALTER
  - Allows us to add or remove attributes (columns) from a relation (table)
    - ALTER TABLE TableName ADD AttributeName AttributeType
    - ALTER TABLE TableName DROP AttributeName
- RENAME
  - Allows the renaming of tables (relations)
    - RENAME TABLE CurrentTableName TO NewTableName

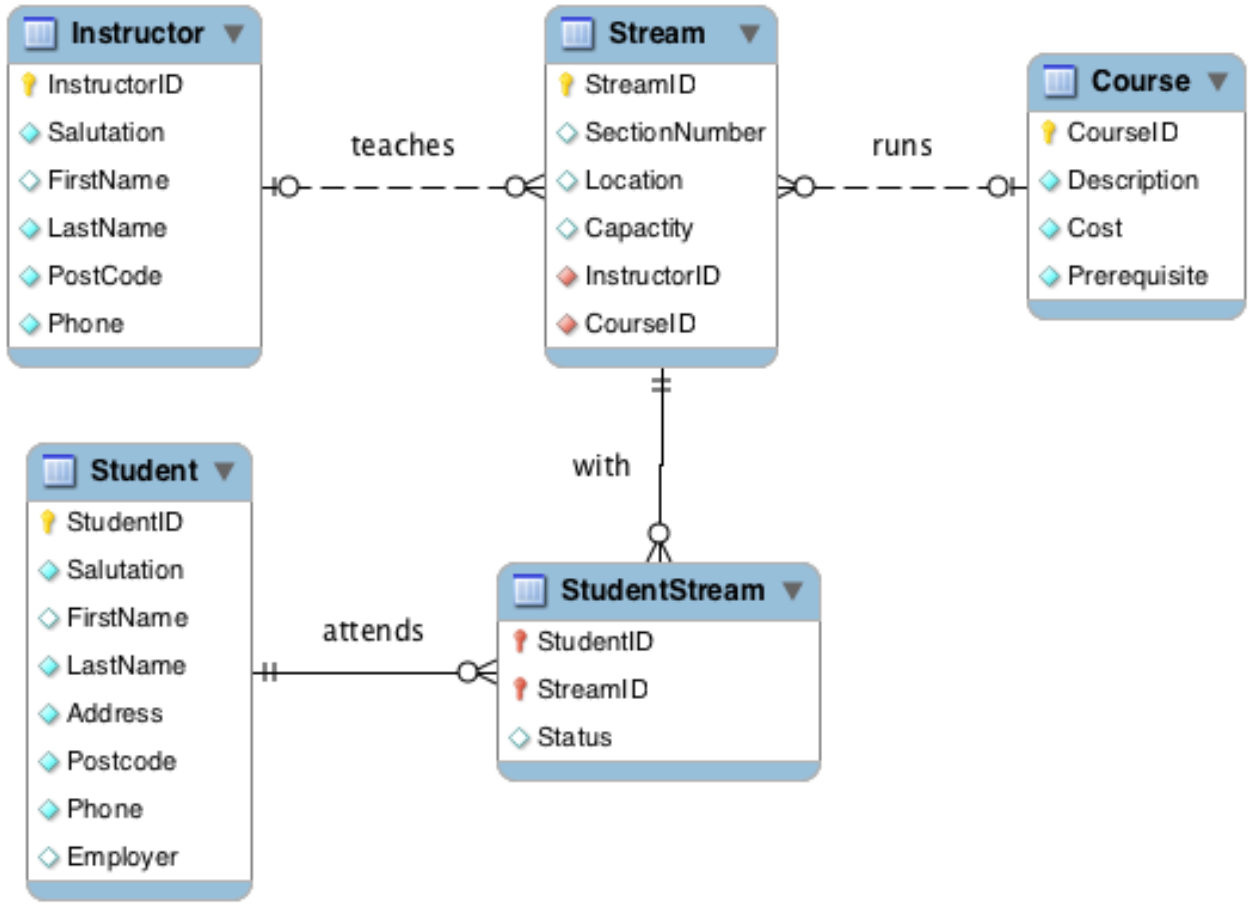
How to Think like SQL

- It's going to be critical for you to think like SQL to handle the queries you will need to write…

- Hopefully the following discussion will help you in this endeavour:
  1. USE the database design as a MAP to help you when you are formulating queries
  2. USE the structure of the SELECT statement as a template
  3. FILL out parts of the SELECT structure and BUILD the query

- Let's try it!

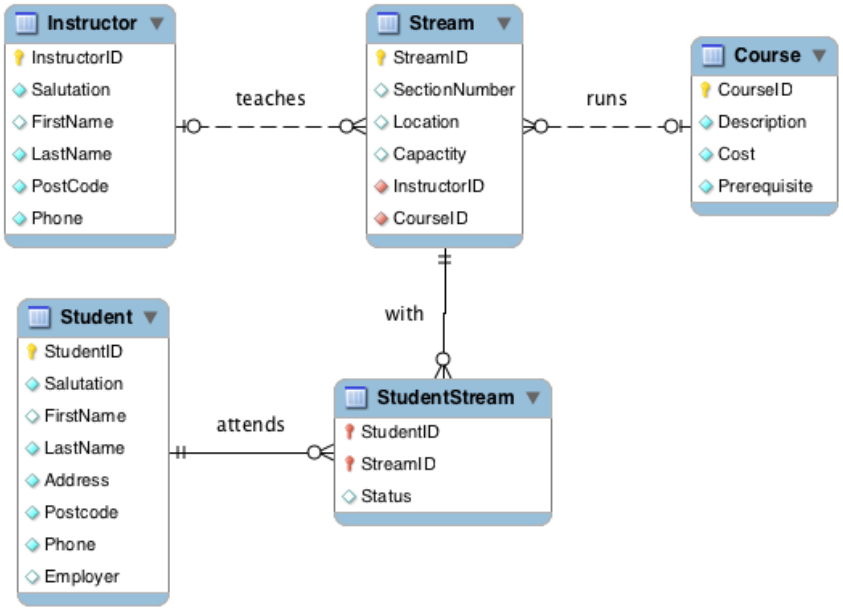
Example



Example: *Which employers employ students who are doing a course in locations where the capacity is greater than 20 persons, and what are those locations?*

How to approach writing queries

Which employers employ students who are doing a course in locations where the capacity is greater than 20 persons, and what are those locations?



- What is the query asking for:
  - Which fields & tables:
    - F: Employer, Location, Capacity
    - T: Student, Stream, StudentStream
  - But only if the capacity > 20 (condition)

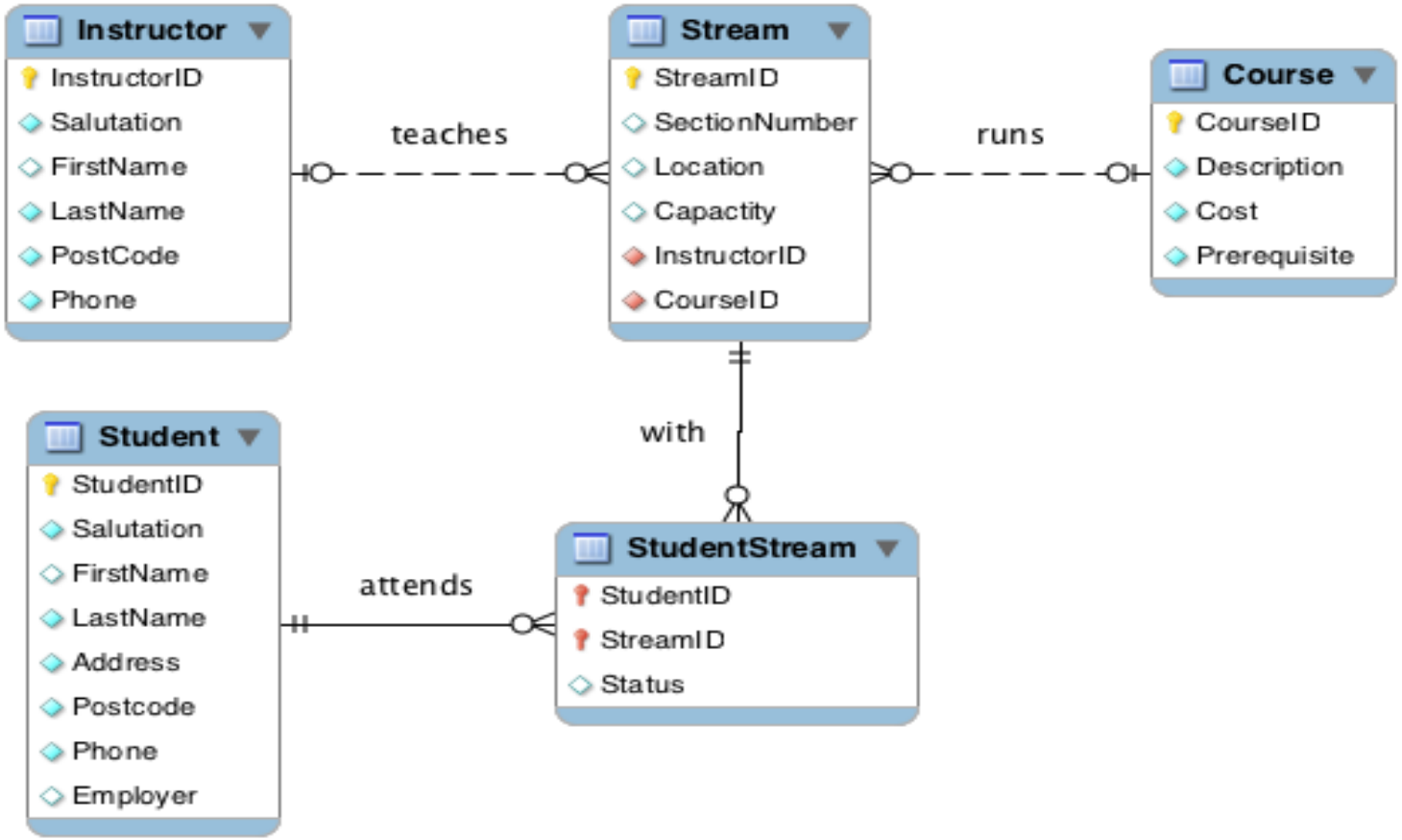
- Lets try to use the structure of the SELECT statement now:

```
SELECT Employer, Location, Capacity
FROM Student INNER JOIN StudentStream
ON Student.StudentID = StudentStream.StudentID
INNER JOIN Stream
ON StudentStream.StreamID = Stream.StreamID
WHERE Capacity > 20;
```

```
SELECT Employer, Location, Capacity
FROM Student
NATURAL JOIN StudentStream
NATURAL JOIN Stream
WHERE Capacity > 20;
```

You try it…

What is the phone number of the instructor who teaches a course that costs over 10000$ attended by studentID 202.



What's Examinable

- You need to know how to write SQL

Next lecture

- Storage and indexing
  - Learn how data is stored and accessed within a DBMS
  - Alternative types of indexes
  - Going “under the hood” of a DBMS