INFO20003 Database Systems

Dr Renata Borovica-Gajic

Lecture 9
SQL Summary

Week 5

In 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
  - DCL

- 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 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

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

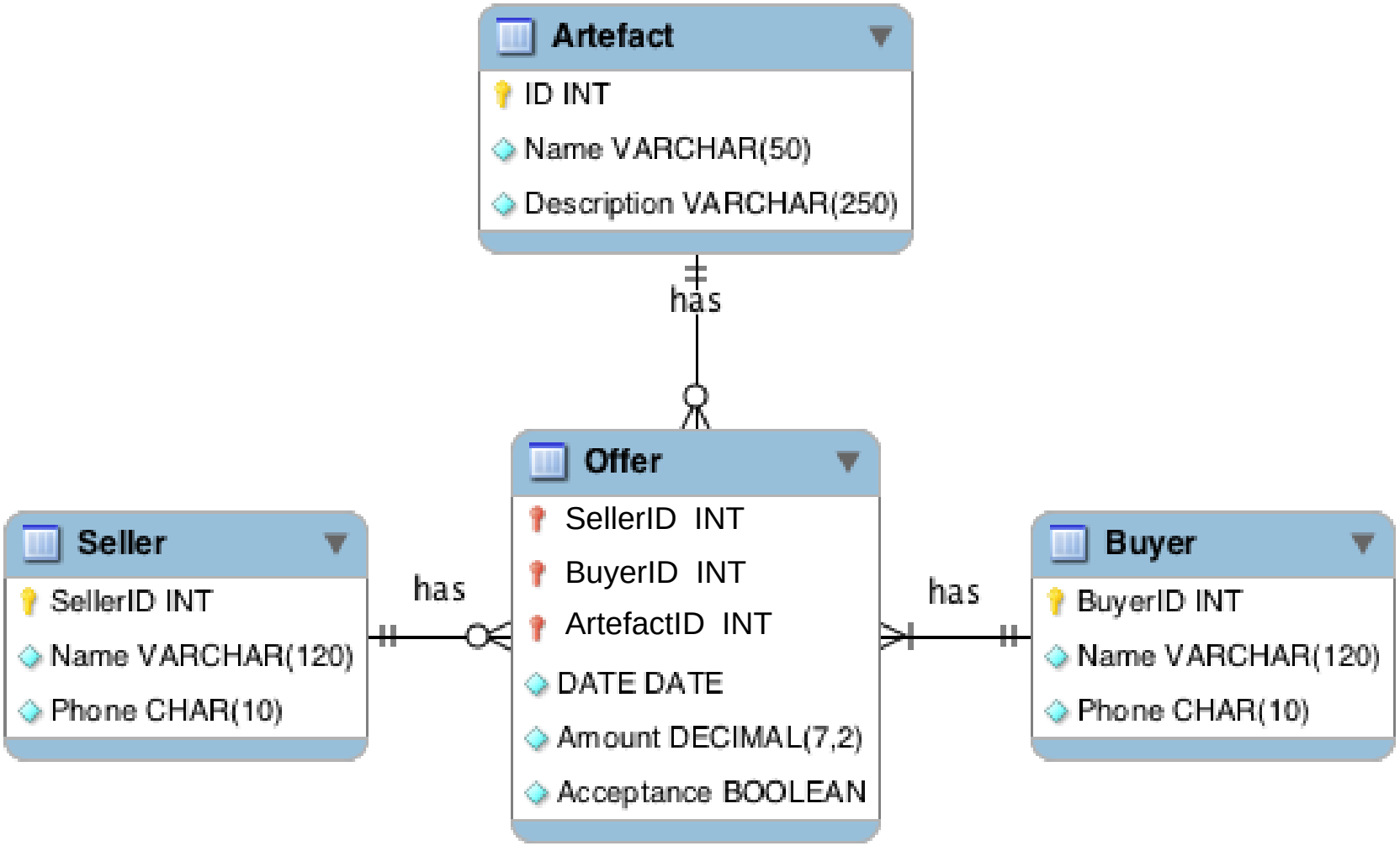
Sub-Query Comparison Operators

- 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
- EXISTS
  - True if the subquery returns one or more records

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

- General help with SQL: https://www.w3schools.com/sql/ **(great tutorial!)**

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)
```

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

Buyer

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

Result

BuyerID	Name	Phone
1	Maggie	0333333333
2	Nicole	0444444444

More examples using subqueries (NOT IN)

Which Artefacts don't have offers made on them?

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

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

Artefact

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

Result

ID	Name	Description
3	Pot	Old Pot

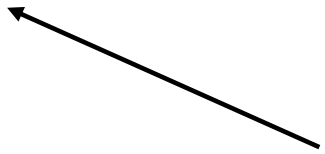
Do we need to use IN? Is there another way…

- 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

Exists example

- Returns true if the subquery returns one or more records
- **Example**: *List the BuyerID, Name and Phone number for all bidders on artefact 1*

```
SELECT * FROM Buyer  WHERE EXISTS
(SELECT * FROM Offer  WHERE Buyer.BuyerID = Offer.BuyerID
                      AND ArtefactID = 1)
```

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

Buyer

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

Result

BuyerID	Name	Phone
1	Maggie	0333333333
2	Nicole	0444444444

ANY/ALL/EXISTS (differences)

- Great tutorial about these:
  - http://www.sqltutorial.org/sql-all/
  - http://www.sqltutorial.org/sql-any/
  - http://www.sqltutorial.org/sql-exists/
- **All**: must satisfy **all** inner conditions

```
SELECT empno, sal
FROM emp
WHERE sal > ALL (200, 300, 400);
```

Equiv.

```
SELECT empno, sal
FROM emp
WHERE sal > 200 AND sal > 300 AND sal> 400;
```

- **Any**: must satisfy **at least one of** the inner conditions (any of)

```
SELECT empno, sal
FROM emp
WHERE sal > ANY (200, 300, 400);
```

Equiv.

```
SELECT empno, sal
FROM emp
WHERE sal > 200 OR sal > 300 OR sal> 400;
```

- **Exists**: the inner query returns **at least one** record

```
SELECT  empid, first_name, last_name
FROM employees AS E
WHERE
    EXISTS( SELECT  * FROM dependents AS D WHERE D. empid = E. empid);
```

← *"Print all employees who have at least one dependent"*

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

More DDL Commands

- TRUNCATE
  - Same as DELETE * FROM table;
  - Faster but cannot ROLL BACK a TRUNCATE command
    - Have to get data back from backup…
- DROP
  - Potentially DANGEROUS
    - Kills a relation – removes the data, removes the relation
      - There is NO UNDO COMMAND! (have to restore from backup)
    - DROP TABLE TableName

Data Control Language / Other Commands

- DCL
  - Users and permissions
    - **CREATE USER, DROP USER**
    - **GRANT, REVOKE**
    - **SET PASSWORD**
- Other Commands
  - Database administration
    - **BACKUP TABLE, RESTORE TABLE**
    - **ANALYZE TABLE**
  - Miscellaneous
    - **DESCRIBE tablename**
    - **USE db_name**
- They are typically called ‘Database Administration Statements’

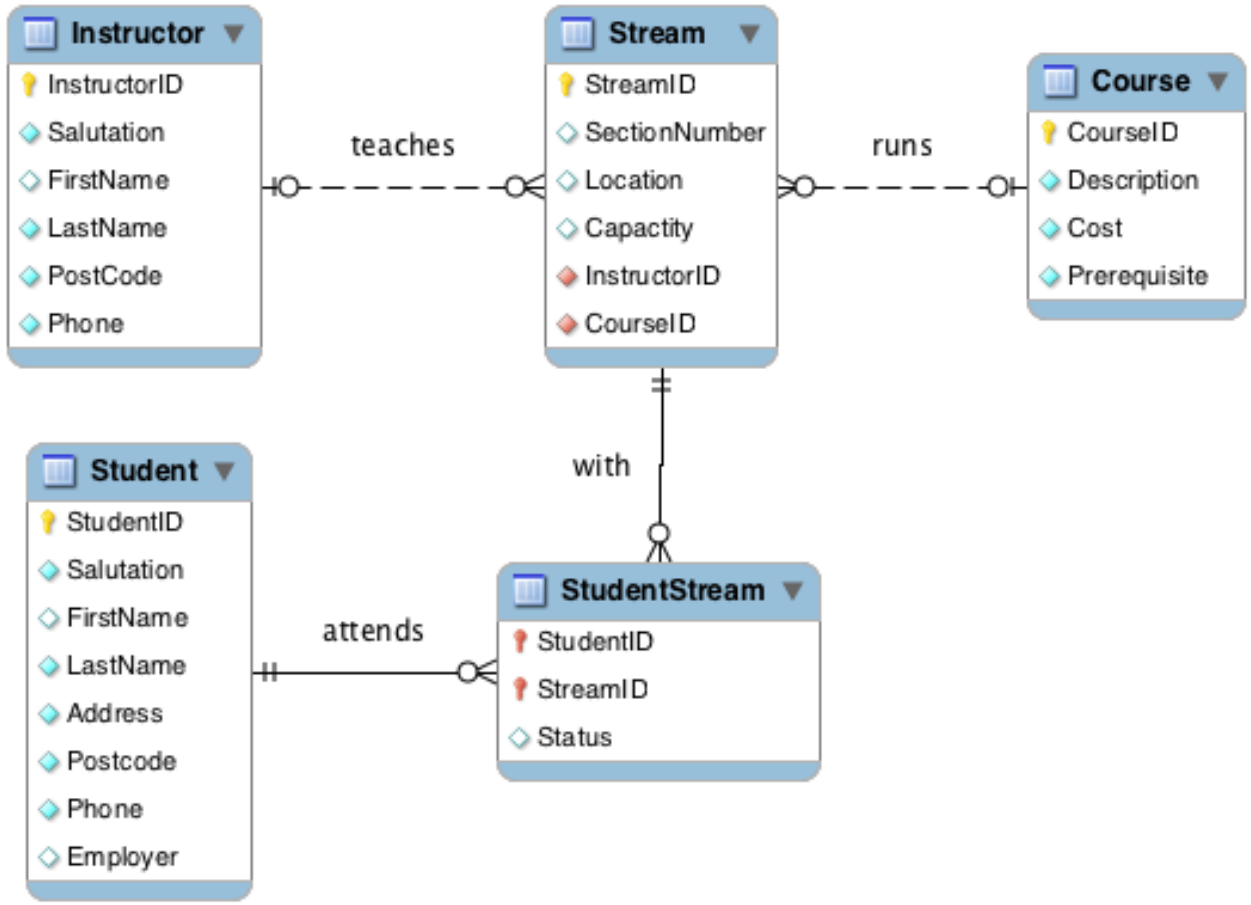
How to think about SQL

- 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!

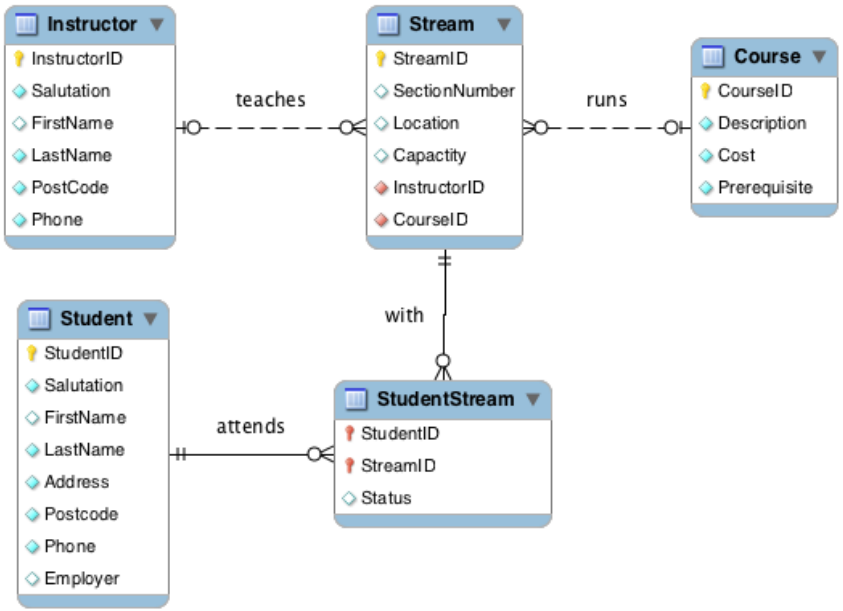
Example



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
    - T: Student, Stream, StudentStream
  - But only if the capacity > 20 (condition)

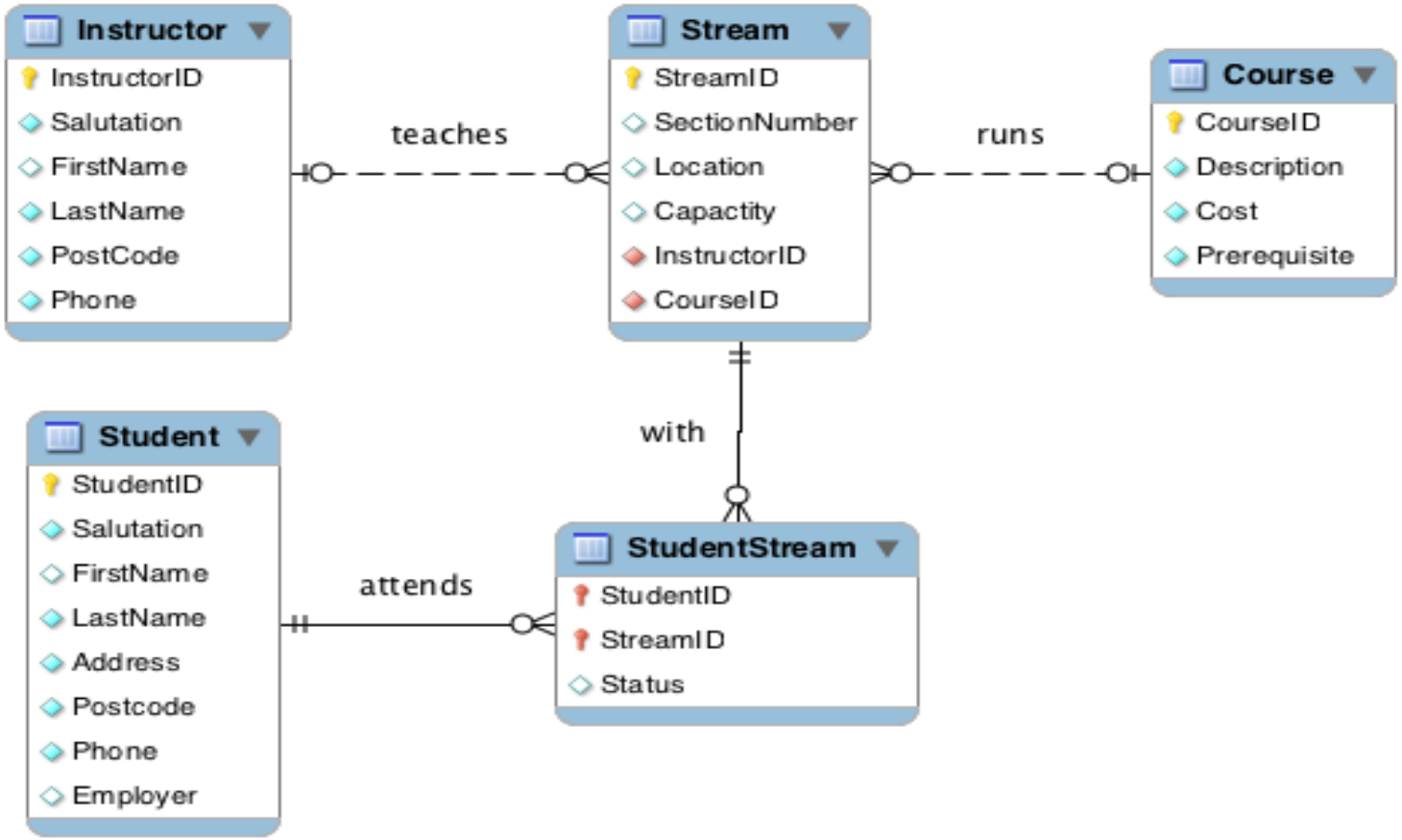
- Lets try to use the structure of the SELECT statement now:

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

```
SELECT Employer, Location
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.



SQL Video

- A very good overview:

https://www.youtube.com/watch?v=uRdIdd-UkTc&index=7&list=PLdQddgMBv5zHcEN9RrhADq3CBColhY2hl

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