Advanced SQL and Connector /J

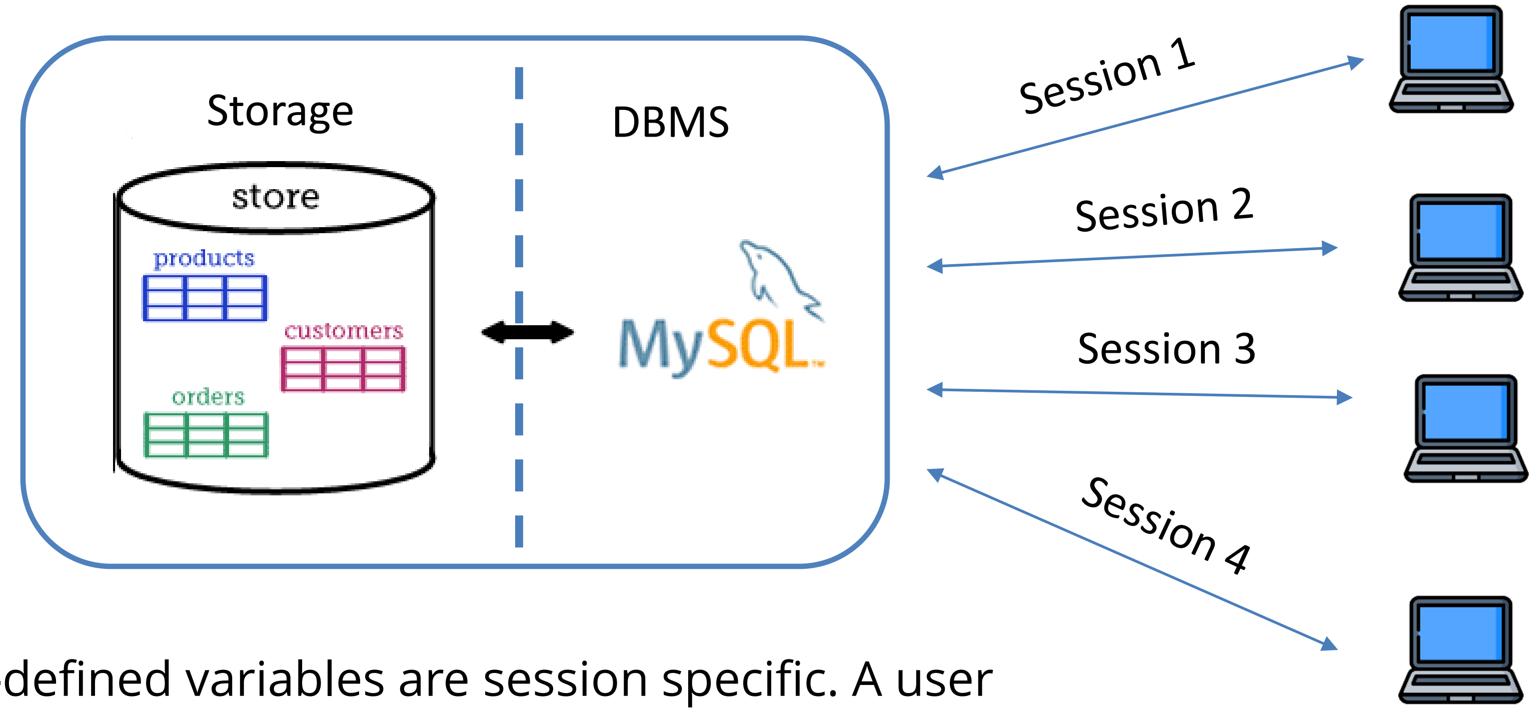
Amos Azaria, Netanel Chkroun

Variables

User-defined variables (@):

You can store a value in a user-defined variable in one statement and refer to it later in another statement. This enables you to pass values from one statement to another

SQL Session



User-defined variables are session specific. A user variable defined by one client cannot be seen or used by other clients.

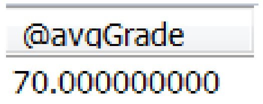
All variables for a given client session are automatically freed when that client exits.

Variables

- **User-defined variables** (@):
  - SET @passGrade = 60
  - SELECT @avgGrade := AVG(grade) FROM grades
- **Local variables in stored procedures** (will learn later)
  - DECLARE passGrade INT

To see a variable's value, you can SELECT it:

- SELECT @avgGrade



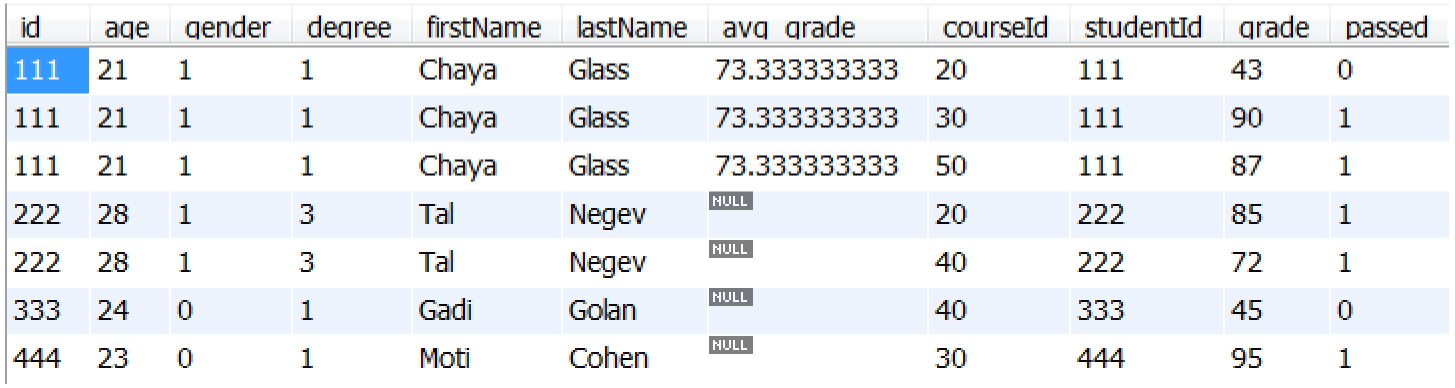
```
@avgGrade
70.000000000
```

TEMPORARY TABLE

- Variables cannot hold tables.
- If you would like to use a table during execution, you can use the TEMPORARY keyword:
  - CREATE TEMPORARY TABLE tempTable (id INT, name VARCHAR(1000));
  - INSERT INTO tempTable (SELECT id, lastName FROM students);
  - (5 rows effected)
- You can also combine create with select:
  - CREATE TEMPORARY TABLE tempTable2 AS (SELECT * FROM students);

Aliases

- SELECT * FROM students AS s INNER JOIN grades AS g ON s.id=g.studentId;
- SELECT * FROM students s INNER JOIN grades g ON s.id=g.studentId;



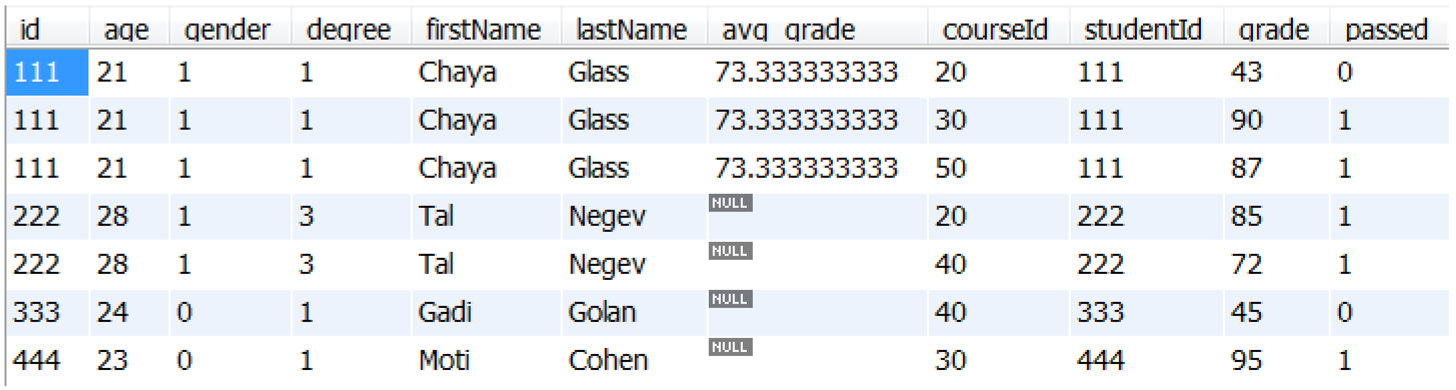
id	age	gender	degree	firstName	lastName	avg_grade	courseId	studentId	grade	passed
111	21	1	1	Chaya	Glass	73.333333333	20	111	43	0
111	21	1	1	Chaya	Glass	73.333333333	30	111	90	1
111	21	1	1	Chaya	Glass	73.333333333	50	111	87	1
222	28	1	3	Tal	Negev	NULL	20	222	85	1
222	28	1	3	Tal	Negev	NULL	40	222	72	1
333	24	0	1	Gadi	Golan	NULL	40	333	45	0
444	23	0	1	Moti	Cohen	NULL	30	444	95	1

Aliases

- SELECT * FROM students INNER JOIN grades ON students.id = grades.studentId

- SELECT * FROM students AS s INNER JOIN grades AS g ON s.id=g.studentId;

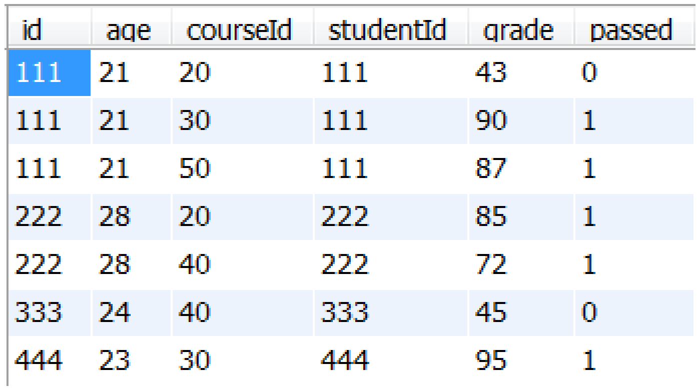
- SELECT * FROM students s INNER JOIN grades g ON s.id=g.studentId;



id	age	gender	degree	firstName	lastName	avg_grade	courseId	studentId	grade	passed
111	21	1	1	Chaya	Glass	73.333333333	20	111	43	0
111	21	1	1	Chaya	Glass	73.333333333	30	111	90	1
111	21	1	1	Chaya	Glass	73.333333333	50	111	87	1
222	28	1	3	Tal	Negev	NULL	20	222	85	1
222	28	1	3	Tal	Negev	NULL	40	222	72	1
333	24	0	1	Gadi	Golan	NULL	40	333	45	0
444	23	0	1	Moti	Cohen	NULL	30	444	95	1

Aliases (cont.)

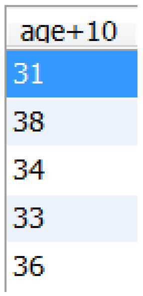
- SELECT * FROM (SELECT id, age FROM students) AS s INNER JOIN grades AS g ON s.id=g.studentId;



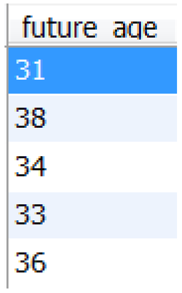
id	age	courseId	studentId	grade	passed
111	21	20	111	43	0
111	21	30	111	90	1
111	21	50	111	87	1
222	28	20	222	85	1
222	28	40	222	72	1
333	24	40	333	45	0
444	23	30	444	95	1

Aliases (cont.)

- SELECT age+10 FROM students;
- SELECT age+10 AS future_age FROM students;



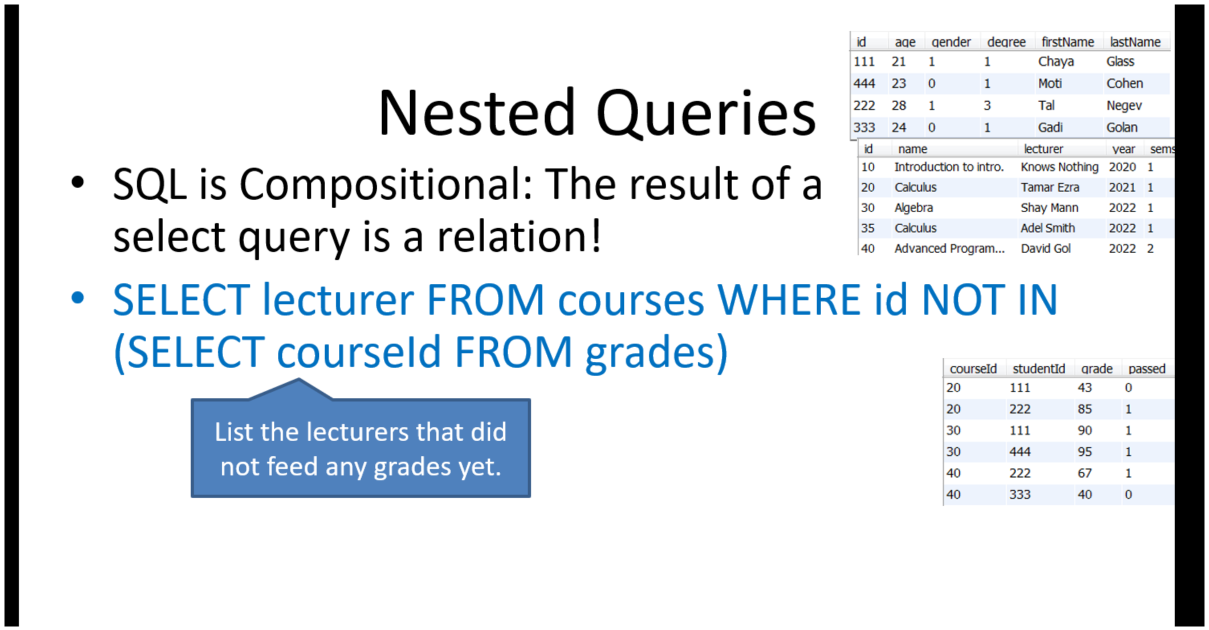
age+10
31
38
34
33
36



future_age
31
38
34
33
36

Aliases in Nested Queries

- We saw nested queries in the WHERE clause



Nested Queries

- SQL is Compositional: The result of a select query is a relation!
- SELECT lecturer FROM courses WHERE id NOT IN (SELECT courseId FROM grades)

List the lecturers that did not feed any grades yet.

id	age	gender	degree	firstName	lastName
111	21	1	1	Chaya	Glass
444	23	0	1	Moti	Cohen
222	28	1	3	Tal	Negev
333	24	0	1	Gadi	Golan

id	name	lecturer	year	sems
10	Introduction to intro.	Knows Nothing	2020	1
20	Calculus	Tamar Ezra	2021	1
30	Algebra	Shay Mann	2022	1
35	Calculus	Adel Smith	2022	1
40	Advanced Program...	David Gol	2022	2

courseId	studentId	grade	passed
20	111	43	0
20	222	85	1
30	111	90	1
30	444	95	1
40	222	67	1
40	333	40	0

but nested queries can be used in the **FROM** clause as well

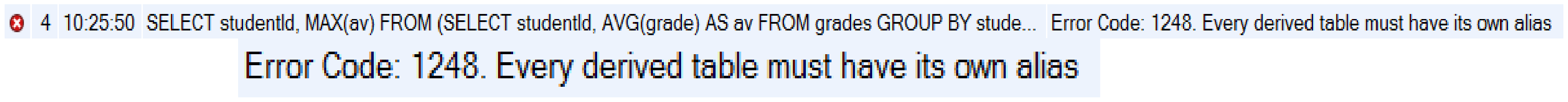
Aliases in Nested Queries

- ~~SELECT MAX(av)~~

  ~~FROM (SELECT studentId, AVG(grade) AS av~~

  ~~FROM grades~~

  ~~GROUP BY studentId) ;~~

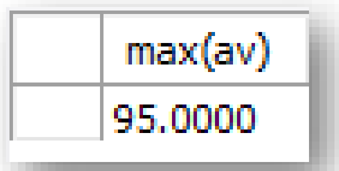


```
4  10:25:50  SELECT studentId, MAX(av) FROM (SELECT studentId, AVG(grade) AS av FROM grades GROUP BY stude...  Error Code: 1248. Every derived table must have its own alias
```

Error Code: 1248. Every derived table must have its own alias

Aliases in Nested Queries

```
• SELECT MAX(av)
  FROM (SELECT studentId, AVG(grade) AS av
            FROM grades
            GROUP BY studentId) AS t ;
```

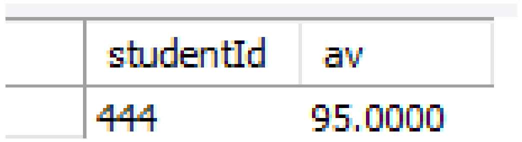


	max(av)
	95.0000

Obtain the student\s with the highest average

- If we want to select the studentId in the outer query-

```
SELECT studentId, AVG(grade) AS av
FROM   grades
GROUP BY studentId
HAVING av = (SELECT MAX(grouped.av)
                    FROM   (SELECT studentId, AVG(grade) AS av
                            FROM grades
                            GROUP BY studentId) AS grouped)
```

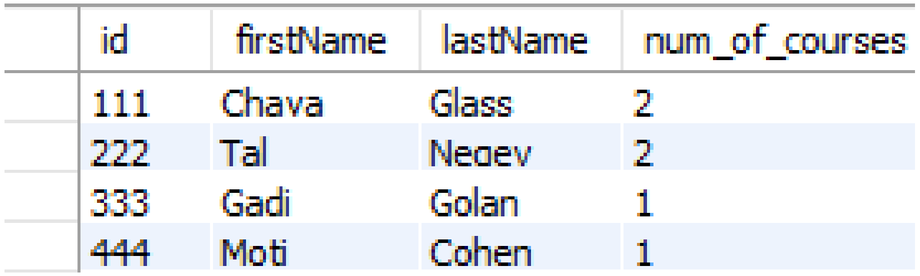


studentId	av
444	95.0000

Aliases in Nested Queries cont.

- Nested queries can be also in the **SELECT** clause-

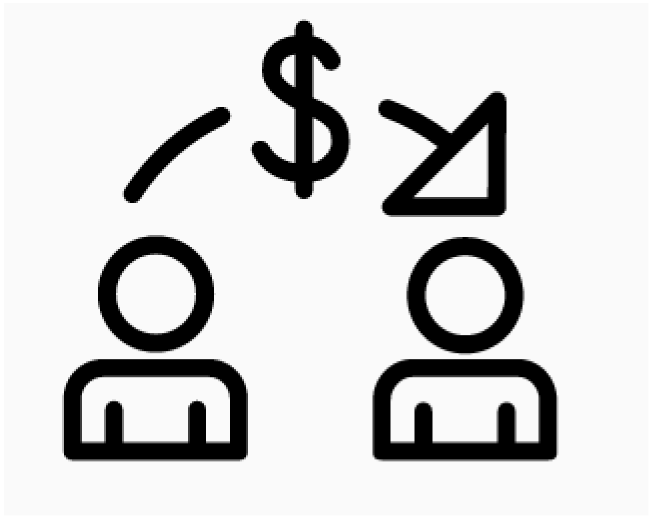
```
SELECT s.id,
        s.firstName,
        s.lastName,
        (SELECT COUNT(grade)
       FROM grades g
       WHERE s.id = g.studentId) AS num_of_courses
FROM students s
```



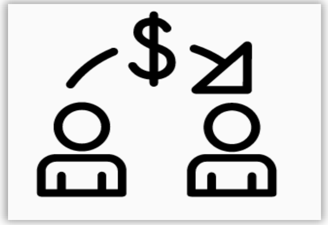
id	firstName	lastName	num_of_courses
111	Chava	Glass	2
222	Tal	Negev	2
333	Gadi	Golan	1
444	Moti	Cohen	1

Transfer 1000$ from Account X to account Y

1. read(X)
2. X = X –1000
3. write(X)
4. read(Y)
5. Y = Y + 1000
6. write(Y)



Transactions



- Suppose we want to transfer money from one bank account to another:
  - SET @transferAmount = 1000;
  - SELECT @firstBalance := amount FROM bankBalances WHERE userId = 777;

    What if we run another instance of this query at this point?
  - UPDATE bankBalances SET amount := @firstBalance - @transferAmount WHERE userId = 777;
  - SELECT @secondBalance := amount FROM bankBalances WHERE userId = 888;

    What if the program crashes here?
  - UPDATE bankBalances SET amount := @secondBalance + @transferAmount WHERE userId = 888;
- What might be the problem with this execution?

Transactions (cont.)

- Transactions are guaranteed to be executed completely or nothing at all (**A**tomicity in ACID). In this example we also rely on the **I**solation attribute (and **D**urability).
- When using a single query, it is treated as a transaction.
- We can combine several queries into a single transaction by using START TRANSACTION and ending with COMMIT.

Transactions (cont.)

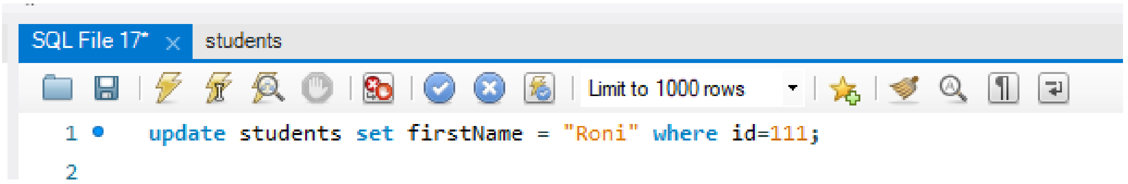
```
SET @transferAmount = 1000;
START TRANSACTION;
SELECT @firstBalance := amount FROM bankBalances
WHERE userId = 777;
UPDATE bankBalances SET amount := @firstBalance -
@transferAmount WHERE userId = 777;
SELECT @secondBalance := amount FROM bankBalances
WHERE userId = 888;
UPDATE bankBalances SET amount := @secondBalance  +
@transferAmount WHERE userId = 888;
COMMIT;
```

Transactions (ROLLBACK)

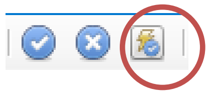
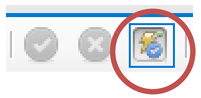
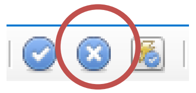
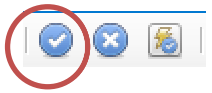
- If you have any error during the transaction, you can call ROLLBACK to undo the current transaction (until previous COMMIT).

Transactions in WorkBench

- There are several transaction related buttons on the workbench toolbar:



```
update students set firstName = "Roni" where id=111;
```

  - Autocommit every query  
  - Rollback 
  - Commit 

Stored Procedures

- Procedures that are stored inside the database:
  - Can be accessed from different programming languages.
  - Can save communication time
  - Can be modified 'on the fly'

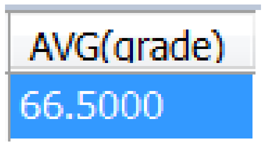
Stored Procedure - Example

```
DELIMITER $$
CREATE PROCEDURE SP_student_avg
(IN stId INT)
BEGIN
  SELECT AVG(grade) FROM grades WHERE
studentId = stId;
END $$
DELIMITER ;
```

We need to change the delimiter, which separates commands, since we use ';' inside the stored procedure. When we finish, we change it back to ';'

Stored Procedure (cont.)

- CALL SP_student_avg(111);

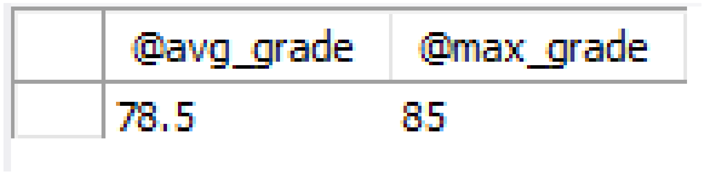


AVG(grade)
66.5000

- DROP PROCEDURE SP_student_avg
- MySQL doesn't (really) support ALTER PROCEDURE, so in order to change a procedure you need to first drop it and then create it again.

Stored procedure with out parameters

```
DELIMITER $$
CREATE PROCEDURE SP_student_avg_2(IN stId INT,
OUT avg_g REAL, OUT max_g INT)
BEGIN
SELECT AVG(grade) INTO avg_g FROM grades
WHERE studentId = stId;
SELECT MAX(grade) INTO max_g FROM grades
WHERE StudentId = stId;
END$$
DELIMITER ;
```

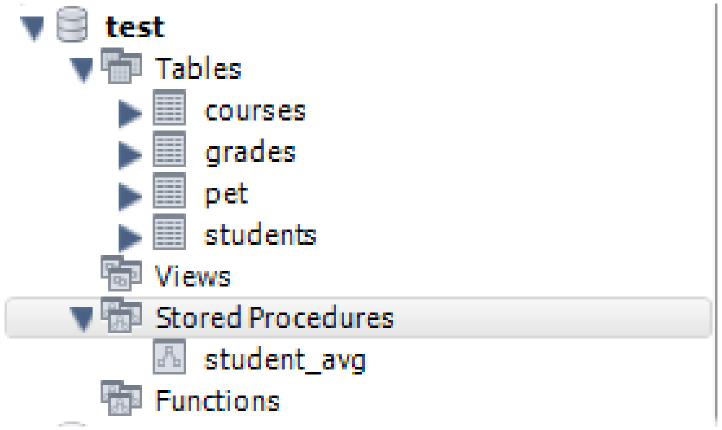


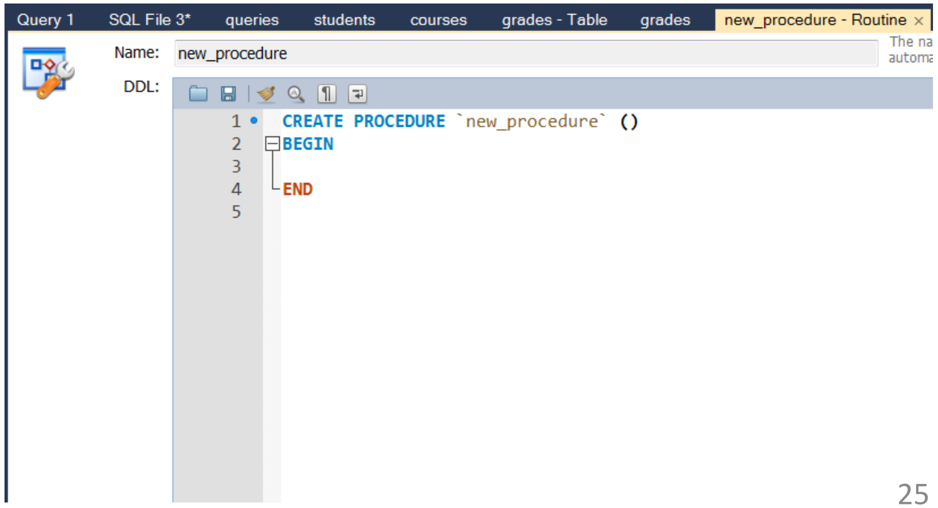
	@avg_grade	@max_grade
	78.5	85

- CALL SP_student_avg_2(222, @avg_grade, @max_grade);
- SELECT @avg_grade, @max_grade;

Stored Procedure Workbench

- In Workbench you can easily create and alter stored procedures using the GUI.
  - Simply right click on the "Stored Procedures" and select "Create" to create a stored procedure.
  - Right click on a stored procedure and select "alter stored procedure"





Name: new_procedure

DDL:

```
CREATE PROCEDURE `new_procedure` ()
BEGIN

END
```

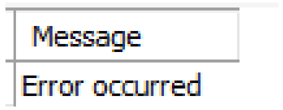
Stored Procedure- Errors handling

```
CREATE PROCEDURE transfer_balance(
     IN sender INT,
     IN receiver INT,
     IN trAmount REAL)
BEGIN

   DECLARE EXIT HANDLER FOR SQLEXCEPTION, SQLWARNING
   BEGIN
     ROLLBACK;
     SELECT 'Error occurred' as Message;
   END;

   START TRANSACTION;
   SELECT @sBl = amount FROM bankBalances WHERE userId = sender;
   UPDATE bankBalances SET amount = @sBl - trAmount WHERE userId = sender;
   SELECT @rBl = amount FROM bankBalances WHERE userId = receiver;
   UPDATE bankBalances SET amount = @rBl  + trAmount WHERE userId = receiver;
   COMMIT;

END;
```



```
Message
Error occurred
```

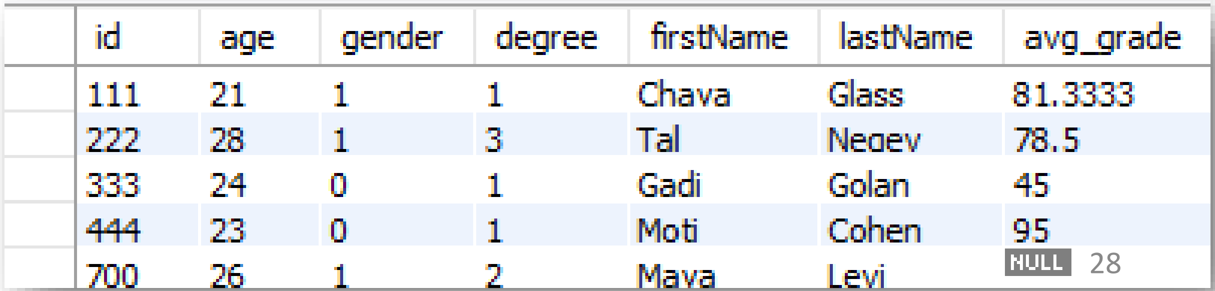
```
CREATE PROCEDURE [dbo].[PgRequestToPlaySP]
               -- Add the parameters for the stored procedure here
               @workerId nchar(20),
    @assignmentId nchar(40),
               @agentPlay int,
               @incriminateMode bit,
               @isIn bit out
AS
BEGIN
     SET NOCOUNT ON;
     --if has entry, update last seen
     declare @hasEntry as int;
     declare @expNum as int;
     set @expNum = 41;
     select @isIn = 1 from PgGames where workerId=@workerId and assignmentId=@assignmentId and status = 1;
     if (@isIn = 1)
     begin
          return @@Error;
     end
     set @isIn = 0;
     select @hasEntry = count(*) from PgWaitingList where workerId=@workerId and assignmentId=@assignmentId and expNum=@expNum;
     if (@hasEntry > 0)
     begin
          update PgWaitingList set lastSeen = CURRENT_TIMESTAMP where workerId=@workerId and assignmentId=@assignmentId and
expNum=@expNum;
     end
     else
     begin
          insert into PgWaitingList (expNum,workerId,assignmentId,insertTime,lastSeen) values
(@expNum,@workerId,@assignmentId,CURRENT_TIMESTAMP,CURRENT_TIMESTAMP);
     end
     --select top 4 desc and create new games
     declare @numOfWaiting as int;
     delete from PgWaitingList where lastSeen < DateADD(mi, -1, Current_TimeStamp); --remove old players
     select @numOfWaiting = count(*) from PgWaitingList where expNum=@expNum;
     if (@numOfWaiting >=4-@agentPlay)
     begin
          --build new games
          declare @maxGameId as int;
          select @maxGameId= max(gameId) from PgGames;
          if (@maxGameId is null)
          begin
               set @maxGameId = 1;
          end
          if (@maxGameId < @expNum * 1000)
          begin
               set @maxGameId = @expNum * 1000;
          end
          --do I need to lock the following two queries?
          insert into PgGames (expNum, workerId, assignmentId, gameId, status, playerId, isPirate, insertTime)
          select top(4-@agentPlay) @expNum, workerId, assignmentId, @maxGameId+1, 1, -1, 0, CURRENT_TIMESTAMP from PgWaitingList
where expNum=@expNum order by insertTime;
          delete from PgWaitingList where assignmentId in (select assignmentId from PgGames where status=1); --remove from waiting

                                        update top (1) PgGames set isPirate=1 where playerId=-1 and assignmentId = (select top 1 assignmentId
```

Stored procedures may be long, and may contain IF clauses, and WHILE clauses

Triggers

- Suppose we would like to have a column that will hold the average for every student.
- Let's add the new column:
  - ALTER TABLE students ADD avg_grade REAL;
- Let's update all rows
  - UPDATE students s join (SELECT studentId, AVG(grade) as av from grades GROUP BY studentId) as v on s.id=v.studentId SET s.avg_grade=v.av;



id	age	gender	degree	firstName	lastName	avg_grade
111	21	1	1	Chava	Glass	81.3333
222	28	1	3	Tal	Negev	78.5
333	24	0	1	Gadi	Golan	45
444	23	0	1	Moti	Cohen	95
700	26	1	2	Maya	Levi	NULL

Triggers (Cont.)

- We still need to update it every time a grade is changed, added or deleted!

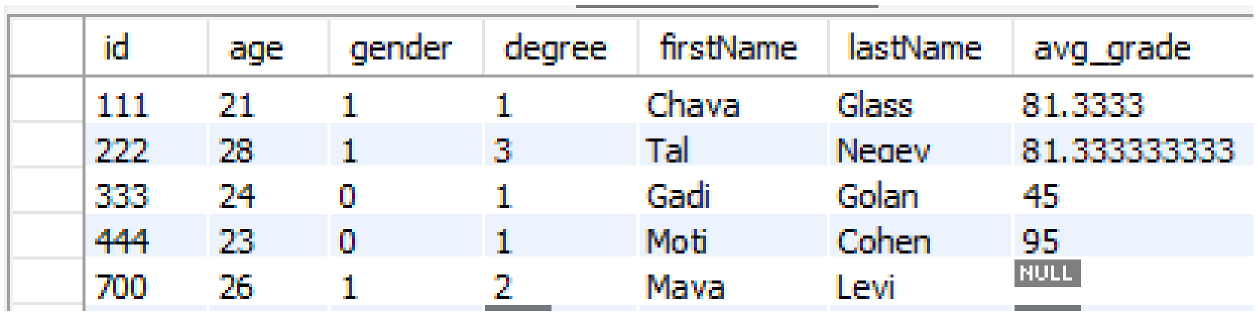
You can only define one event for each trigger (so might need multiple triggers)

You can use BEFORE if you want to access the DB before the change was made

```
DELIMITER $$
CREATE TRIGGER new_grade_received
AFTER INSERT ON grades
FOR EACH ROW
BEGIN
   UPDATE students SET avg_grade = (SELECT AVG(grade) FROM grades
WHERE studentId=NEW.studentId) where id = NEW.studentId;
END$$
DELIMITER ;
```

Triggers execution

- INSERT INTO grades (courseId, studentId, grade, passed) VALUES (30, 222, 87, 1);
- SELECT * FROM students;

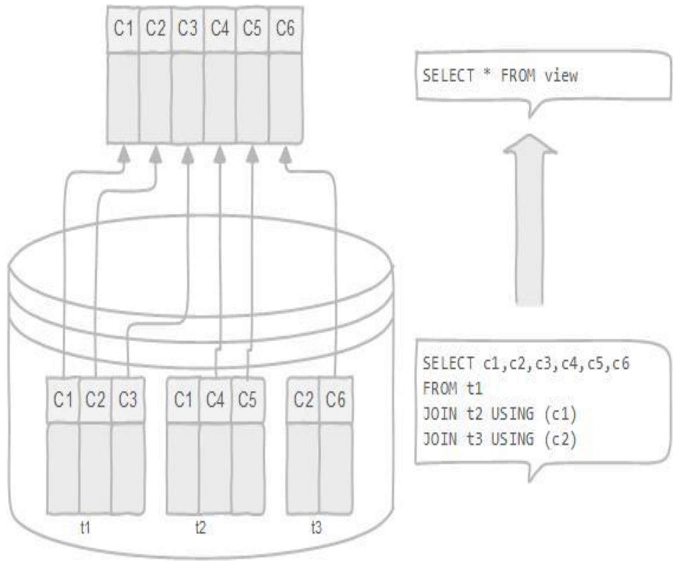


	id	age	gender	degree	firstName	lastName	avg_grade
	111	21	1	1	Chava	Glass	81.3333
	222	28	1	3	Tal	Negev	81.333333333
	333	24	0	1	Gadi	Golan	45
	444	23	0	1	Moti	Cohen	95
	700	26	1	2	Mava	Levi	NULL

- DROP TRIGGER new_grade_received;

VIEWS

- Simplify complex queries
- Limit data access to specific users
- Enable computed columns

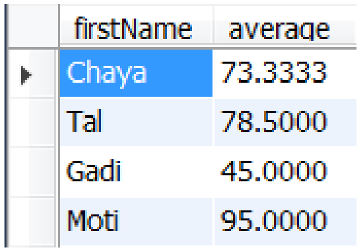

```
SELECT * FROM view
```
```
SELECT c1,c2,c3,c4,c5,c6
FROM t1
JOIN t2 USING (c1)
JOIN t3 USING (c2)
```

http://www.mysqltutorial.org/introduction-sql-views.aspx

View Example

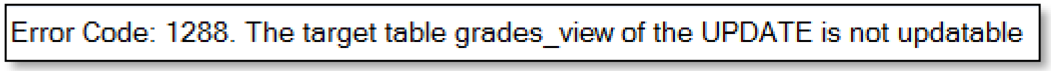
- CREATE VIEW avg_grades_view AS
  SELECT students.firstName, AVG(grade) AS average
  FROM grades
  INNER JOIN students ON grades.studentId = students.id
  GROUP BY studentId;

- SELECT * FROM avg_grades_view



firstName	average
Chaya	73.3333
Tal	78.5000
Gadi	45.0000
Moti	95.0000

- UPDATE avg_grades_view SET average=75 WHERE firstName LIKE 'Chaya';

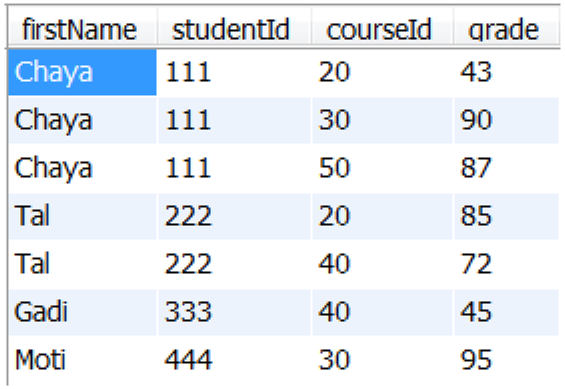


```
Error Code: 1288. The target table grades_view of the UPDATE is not updatable
```

Updatable View

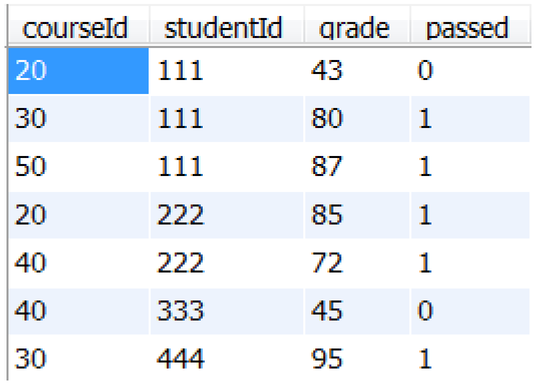
- CREATE VIEW full_grades_view AS

  SELECT students.firstName, studentId, courseId, grade FROM grades INNER JOIN students ON grades.studentId = students.id;
- SELECT * FROM full_grades_view;
- UPDATE full_grades_view SET grade = 80 WHERE firstName LIKE 'Chaya' AND courseId=30;



firstName	studentId	courseId	grade
Chaya	111	20	43
Chaya	111	30	90
Chaya	111	50	87
Tal	222	20	85
Tal	222	40	72
Gadi	333	40	45
Moti	444	30	95

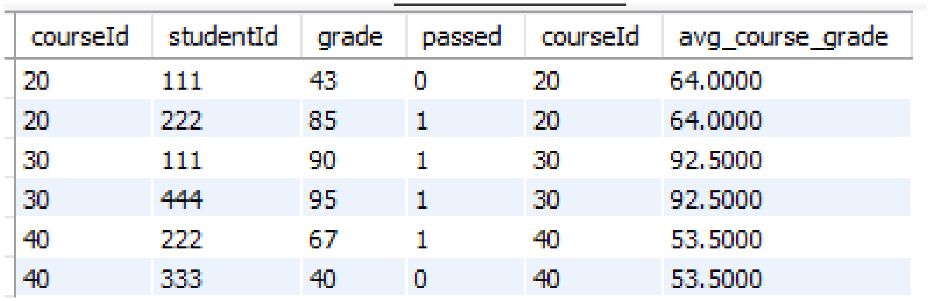
- SELECT * FROM grades;



courseId	studentId	grade	passed
20	111	43	0
30	111	80	1
50	111	87	1
20	222	85	1
40	222	72	1
40	333	45	0
30	444	95	1

Window Functions

- Suppose we want to get the grades of all the students, but also compare them to the average grade in each course.
- SELECT * FROM grades JOIN (SELECT courseId, avg(grade) as avg_course_grade FROM grades GROUP BY courseId) AS with_avg ON grades.courseId = with_avg.courseId;



courseId	studentId	grade	passed	courseId	avg_course_grade
20	111	43	0	20	64.0000
20	222	85	1	20	64.0000
30	111	90	1	30	92.5000
30	444	95	1	30	92.5000
40	222	67	1	40	53.5000
40	333	40	0	40	53.5000

Window Functions (cont.)

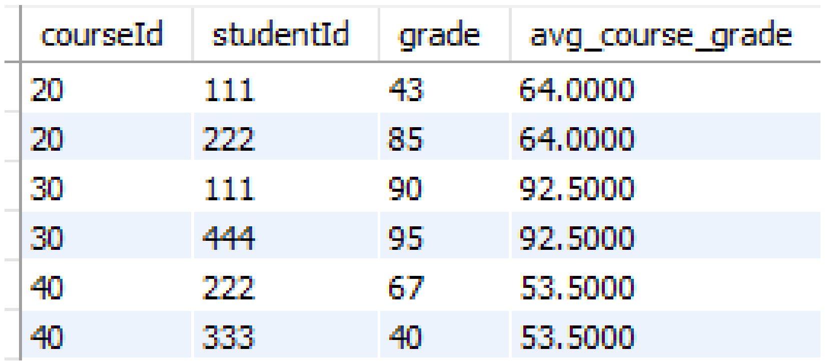
- Window functions act on the aggregating functions, but do not reduce the number of rows (to match the number of groups).
- This is very useful when we want to obtain all the original input and join it with new information.

Window Functions (cont.)

- SELECT courseId, studentId, grade,

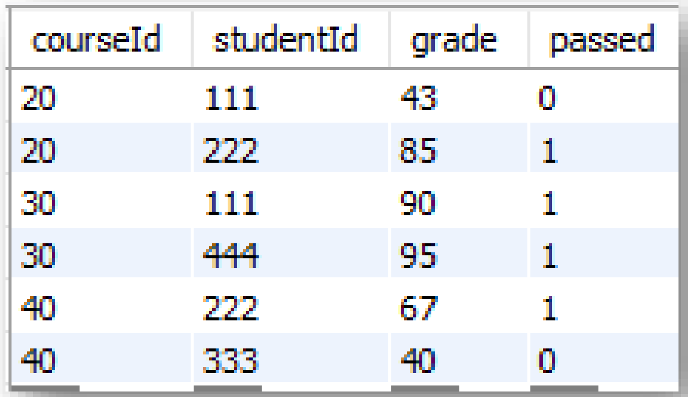
  avg(grade) OVER (PARTITION BY courseId) AS avg_course_grade

  FROM grades;



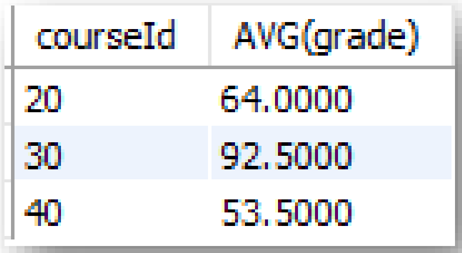
courseId	studentId	grade	avg_course_grade
20	111	43	64.0000
20	222	85	64.0000
30	111	90	92.5000
30	444	95	92.5000
40	222	67	53.5000
40	333	40	53.5000

Window function VS group By

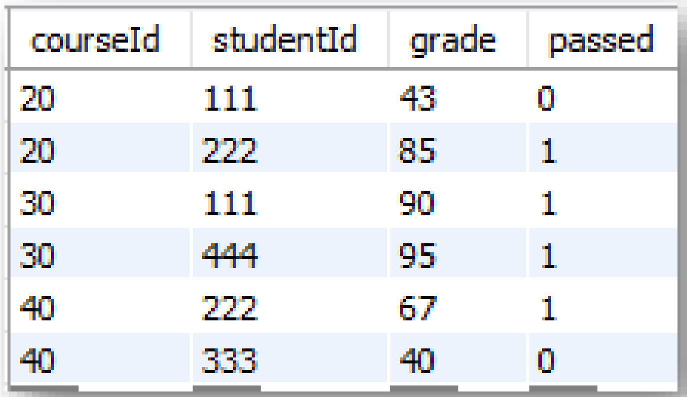


courseId	studentId	grade	passed
20	111	43	0
20	222	85	1
30	111	90	1
30	444	95	1
40	222	67	1
40	333	40	0

Group By

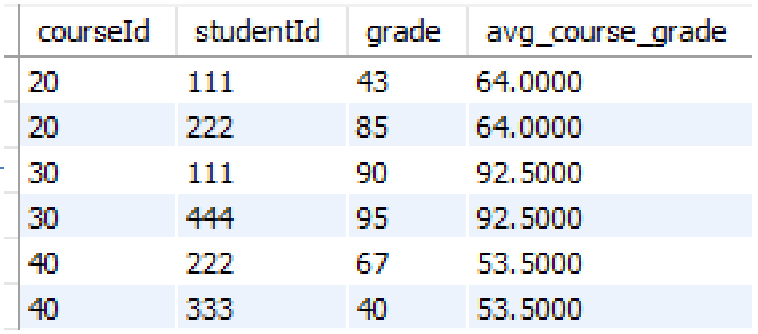


courseId	AVG(grade)
20	64.0000
30	92.5000
40	53.5000



courseId	studentId	grade	passed
20	111	43	0
20	222	85	1
30	111	90	1
30	444	95	1
40	222	67	1
40	333	40	0

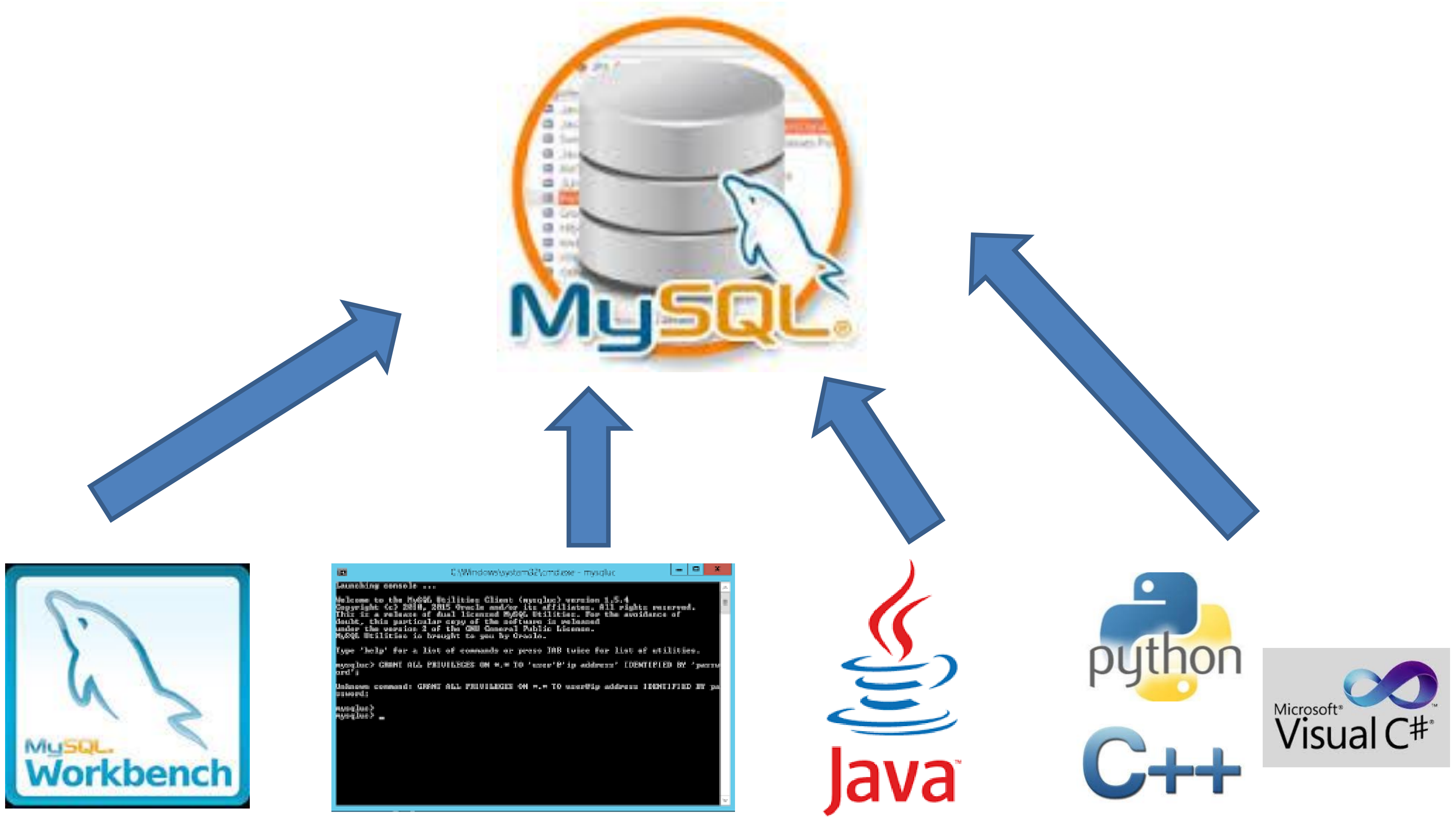
Window function



courseId	studentId	grade	avg_course_grade
20	111	43	64.0000
20	222	85	64.0000
30	111	90	92.5000
30	444	95	92.5000
40	222	67	53.5000
40	333	40	53.5000

Connecting to MySQL from Java (Connector /J)

Methods to connect to MySQL Server



```
Launching console ...

Welcome to the MySQL Utilities Client (mysqluc) version 1.5.4
Copyright (c) 2010, 2015 Oracle and/or its affiliates. All rights reserved.
This is a release of dual licensed MySQL Utilities. For the avoidance of
doubt, this particular copy of the software is released
under the version 2 of the GNU General Public License.
MySQL Utilities is brought to you by Oracle.

Type 'help' for a list of commands or press TAB twice for list of utilities.

mysqluc> GRANT ALL PRIVILEGES ON *.* TO 'user'@'ip address' IDENTIFIED BY 'passw
ord';

Unknown command: GRANT ALL PRIVILEGES ON *.* TO user@ip address IDENTIFIED BY pa
ssword;

mysqluc>
mysqluc> _
```

SELECT * FROM students (in JAVA)

```
import java.sql.*;
public class Main{
  public static void main(String[] args){
    try{
      Class.forName("com.mysql.jdbc.Driver");
      try(Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/myDbName", "user",
"pwd")){
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM students");
        int numOfColumns = rs.getMetaData().getColumnCount();
        while (rs.next()){
          for (int col = 1; col <= numOfColumns; col++){
            System.out.print(rs.getString(col) + " ");
          }
          System.out.println();
        }
      }} catch (Exception ex){ex.printStackTrace();}
  }
}
```

Reflection

Try with resources (java 7). No need to call con.close()

rs is initially located before the first row

We use the getXXX method of the appropriate type to retrieve the value in each column .

To get column names we can use: rs.getMetaData().getColumnLabel(col)

```
111 21 1 1 Chaya Glass 73.33
222 28 1 3 Tal Negev null
333 24 0 1 Gadi Golan null
444 23 0 1 Moti Cohen null
700 26 1 2 Maya Levi null
```

Jar file is missing

```
java.lang.ClassNotFoundException: com.mysql.jdbc.Driver
	at java.net.URLClassLoader.findClass(URLClassLoader.java:381)
	at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
	at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:331)
	at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
	at java.lang.Class.forName0(Native Method)
	at java.lang.Class.forName(Class.java:264)
	at ariel.databases.Main.main(Main.java:19)
```

- Solution 1: use Gradel.
- Solution 2:
  - Goto https://dev.mysql.com/downloads/connector/j/ download jar file.
  - Create a bin folder: copy jar file into the folder
  - Add bin to libraries (in IntelliJ: Project Structure -> Libraries -> + -> JAVA -> find bin directory)

Select with parameters

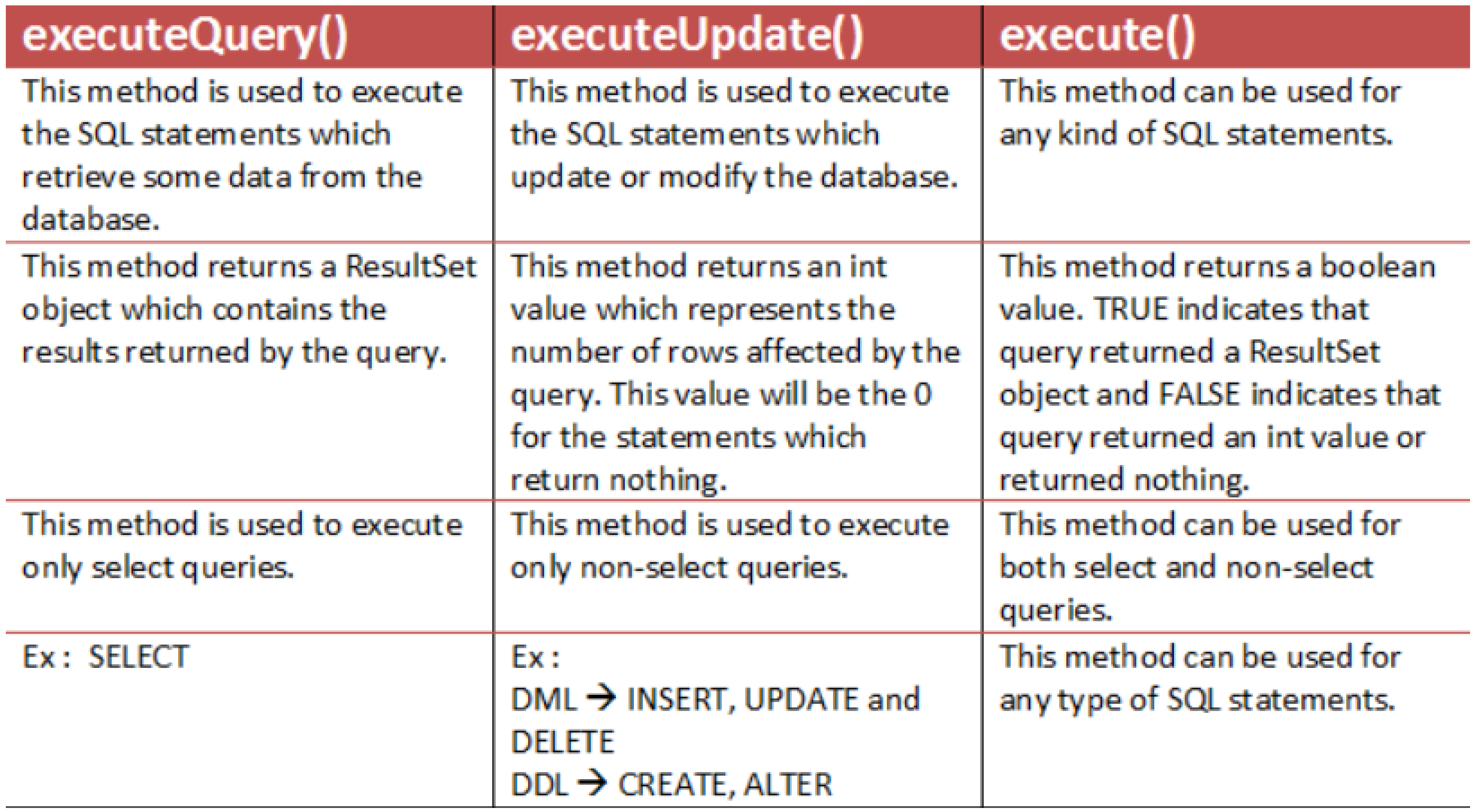
- prepareStatement allows the creating of a statement with missing parameters and filling them up later.
- May be faster and can provide some level of security (especially when part of the query are obtained from user input)

```
userId=111;
String query = “SELECT * FROM students WHERE studentId=?"
try (PreparedStatement pstmt = con.prepareStatement(query))
{
   pstmt.setString(1, userId);
   pstmt.executeQuery();
}
```

Update, Delete & Insert

```
userId = 222;
String query = "DELETE FROM students WHERE studentId=?"
try (PreparedStatement pstmt = con.prepareStatement(query))
{
   pstmt.setString(1, userId);
   pstmt.executeUpdate();
}
```

executeQuery(), executeUpdate(), execute()



executeQuery()	executeUpdate()	execute()
This method is used to execute the SQL statements which retrieve some data from the database.	This method is used to execute the SQL statements which update or modify the database.	This method can be used for any kind of SQL statements.
This method returns a ResultSet object which contains the results returned by the query.	This method returns an int value which represents the number of rows affected by the query. This value will be the 0 for the statements which return nothing.	This method returns a boolean value. TRUE indicates that query returned a ResultSet object and FALSE indicates that query returned an int value or returned nothing.
This method is used to execute only select queries.	This method is used to execute only non-select queries.	This method can be used for both select and non-select queries.
Ex : SELECT	Ex : DML → INSERT, UPDATE and DELETE DDL → CREATE, ALTER	This method can be used for any type of SQL statements.

Credit: http://javaconceptoftheday.com/difference-between-executequery-executeupdate-execute-in-jdbc/

Executing Stored Procedure

```
String query = "{CALL SP_student_avg(?)}";
CallableStatement stmt = con.prepareCall(query);
int studentId = 222;
stmt.setInt(1, studentId);
ResultSet rs = stmt.executeQuery();
```

Stored Procedure Using Out Params

```
Int n = 111;
CallableStatement stmt = con.prepareCall("CALL SP_student_avg_2(?,?,?)");
stmt.setInt(1, n);
stmt.registerOutParameter(2, Types.DOUBLE);
stmt.registerOutParameter(3, Types.INTEGER);
stmt.execute();

Double avgGrade = stmt.getDouble(2);
Integer maxGrade = stmt.getInt(3);

System.out.println("the average grade is: "+avgGrade+".");
System.out.println("the maximum grade is: "+maxGrade+".");
```