SQL – Part I

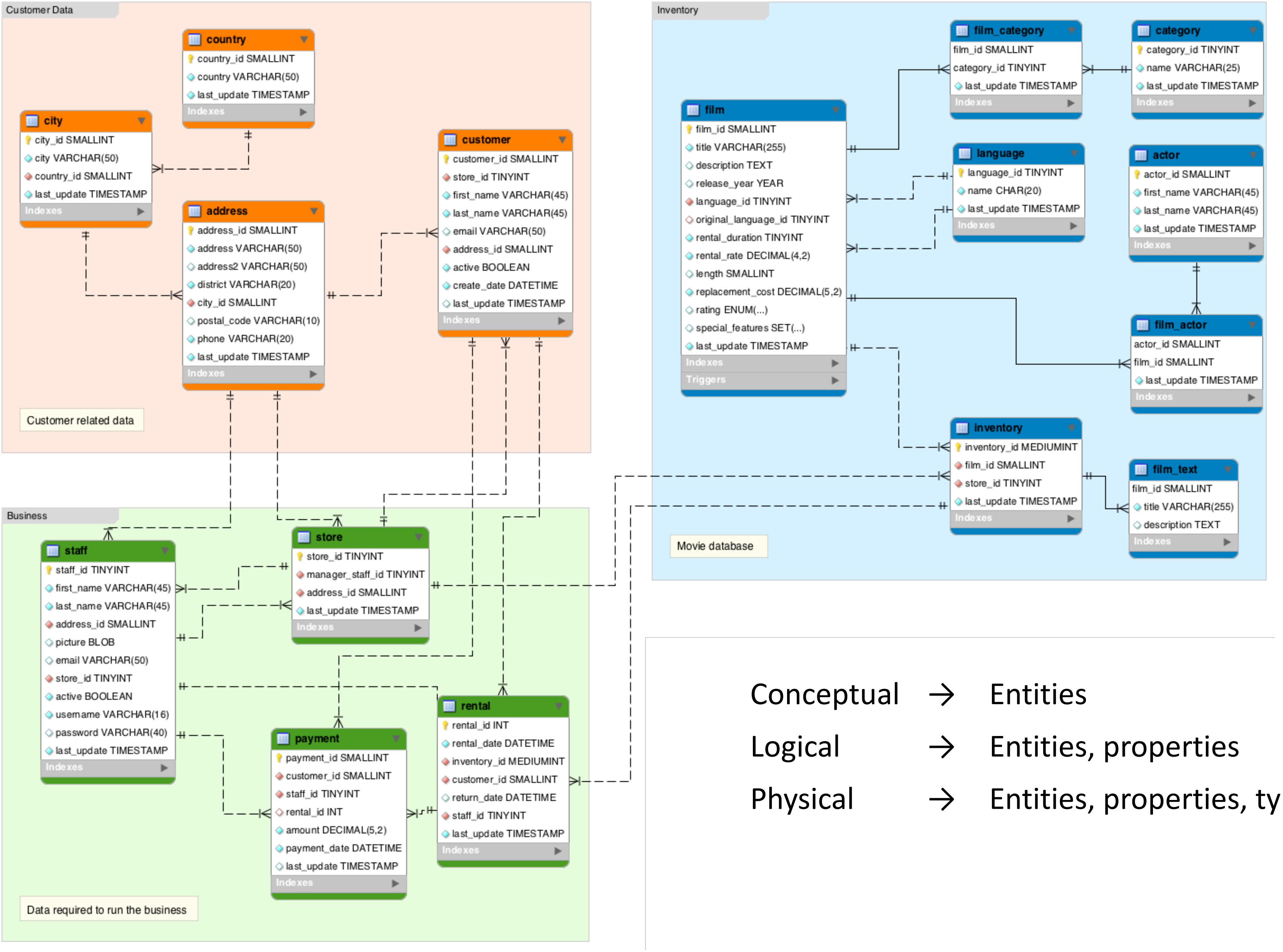
Structured Query Language

Database Design

conceptual, logical, physical

Sakila

Sample Database

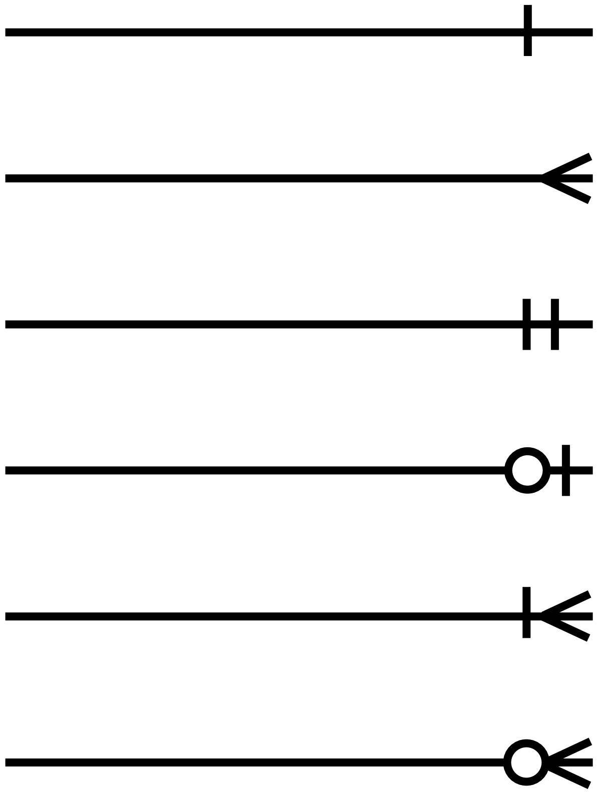


Conceptual → Entities

Logical → Entities, properties

Physical → Entities, properties, types

Cardinality – how one table relates to another



One

Many

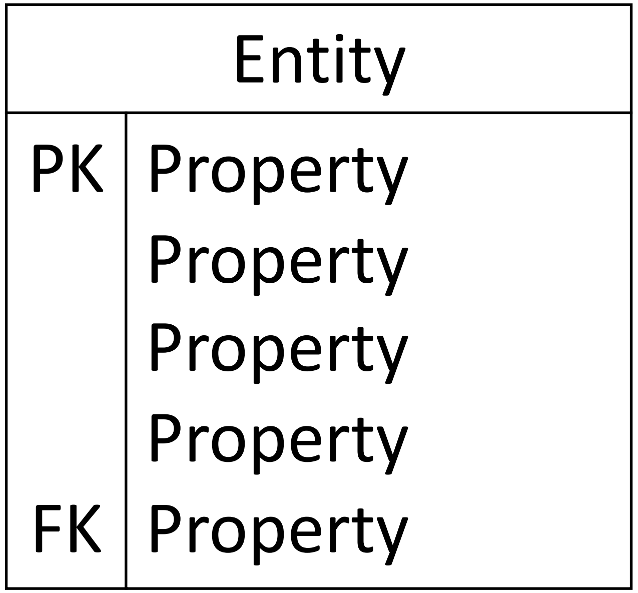
One (and only one)

Zero or one

One or many

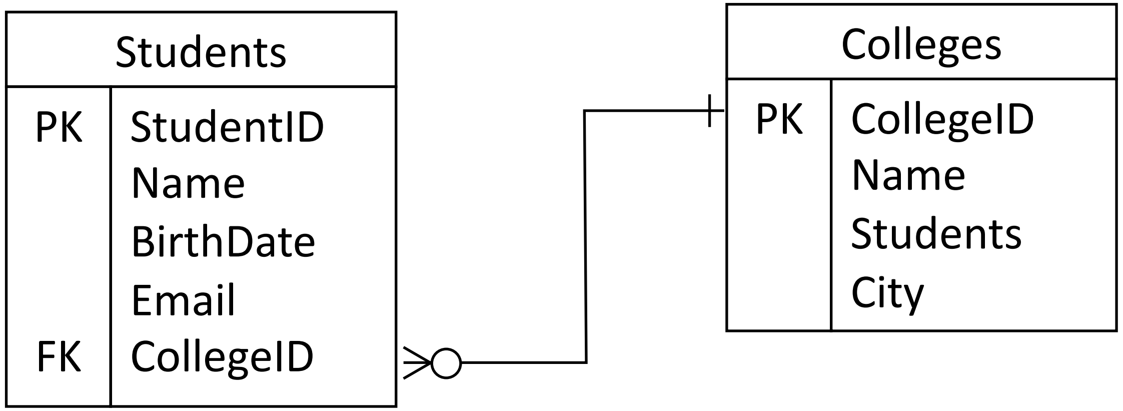
Zero or many

Entities, properties, PK, FK



Entity	
PK	Property
	Property
	Property
	Property
FK	Property

Example



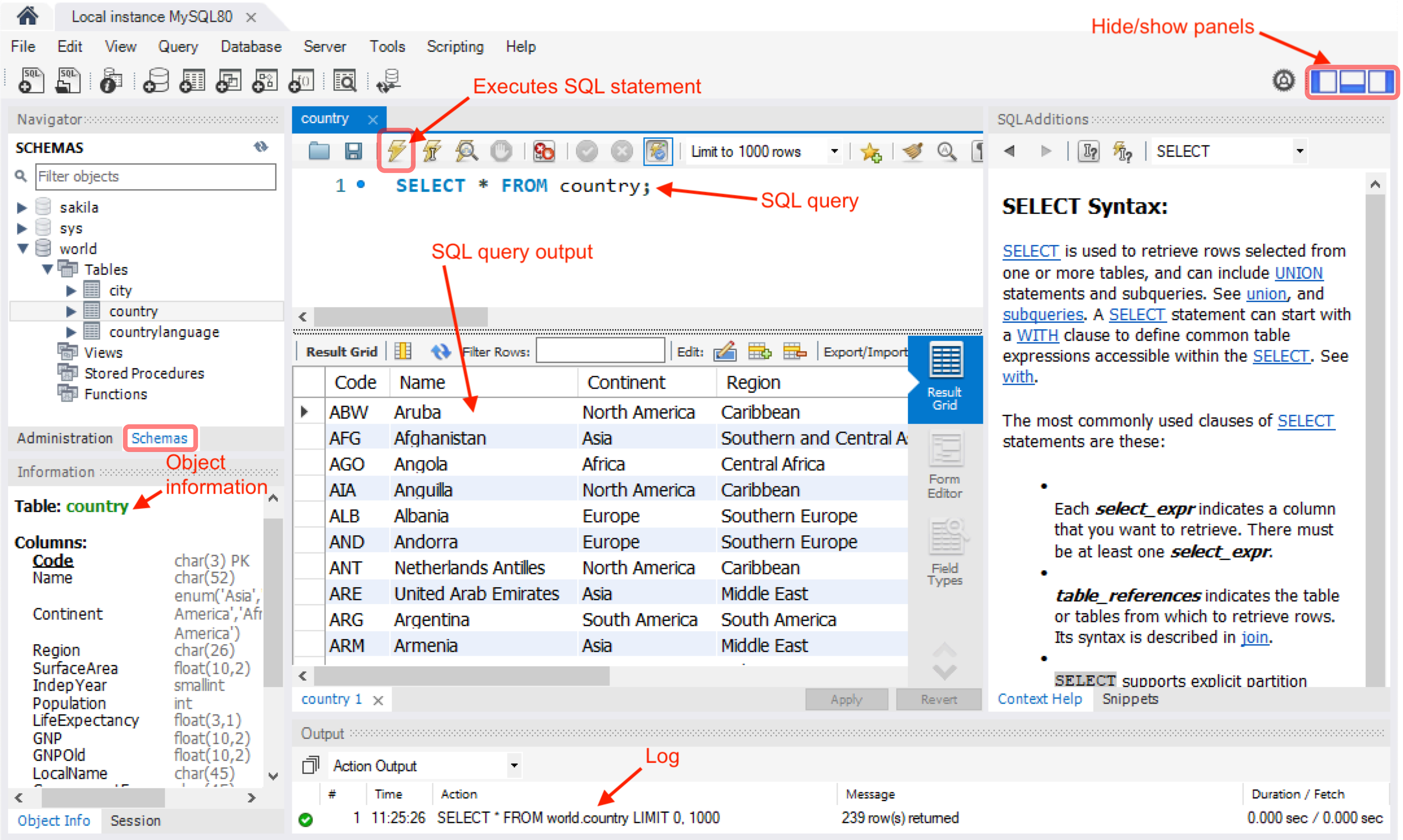
Students	
PK	StudentID
	Name
	BirthDate
	Email
FK	CollegeID

Colleges	
PK	CollegeID
	Name
	Students
	City

Database Platform Installation

MySQL

Workbench



```
1 • SELECT * FROM country;
```

	Code	Name	Continent	Region
▶	ABW	Aruba	North America	Caribbean
	AFG	Afghanistan	Asia	Southern and Central A
	AGO	Angola	Africa	Central Africa
	AIA	Anguilla	North America	Caribbean
	ALB	Albania	Europe	Southern Europe
	AND	Andorra	Europe	Southern Europe
	ANT	Netherlands Antilles	North America	Caribbean
	ARE	United Arab Emirates	Asia	Middle East
	ARG	Argentina	South America	South America
	ARM	Armenia	Asia	Middle East

Table: country

Columns:

Code	char(3) PK
Name	char(52)
Continent	enum('Asia', America','Afr America')
Region	char(26)
SurfaceArea	float(10,2)
IndepYear	smallint
Population	int
LifeExpectancy	float(3,1)
GNP	float(10,2)
GNPOld	float(10,2)
LocalName	char(45)

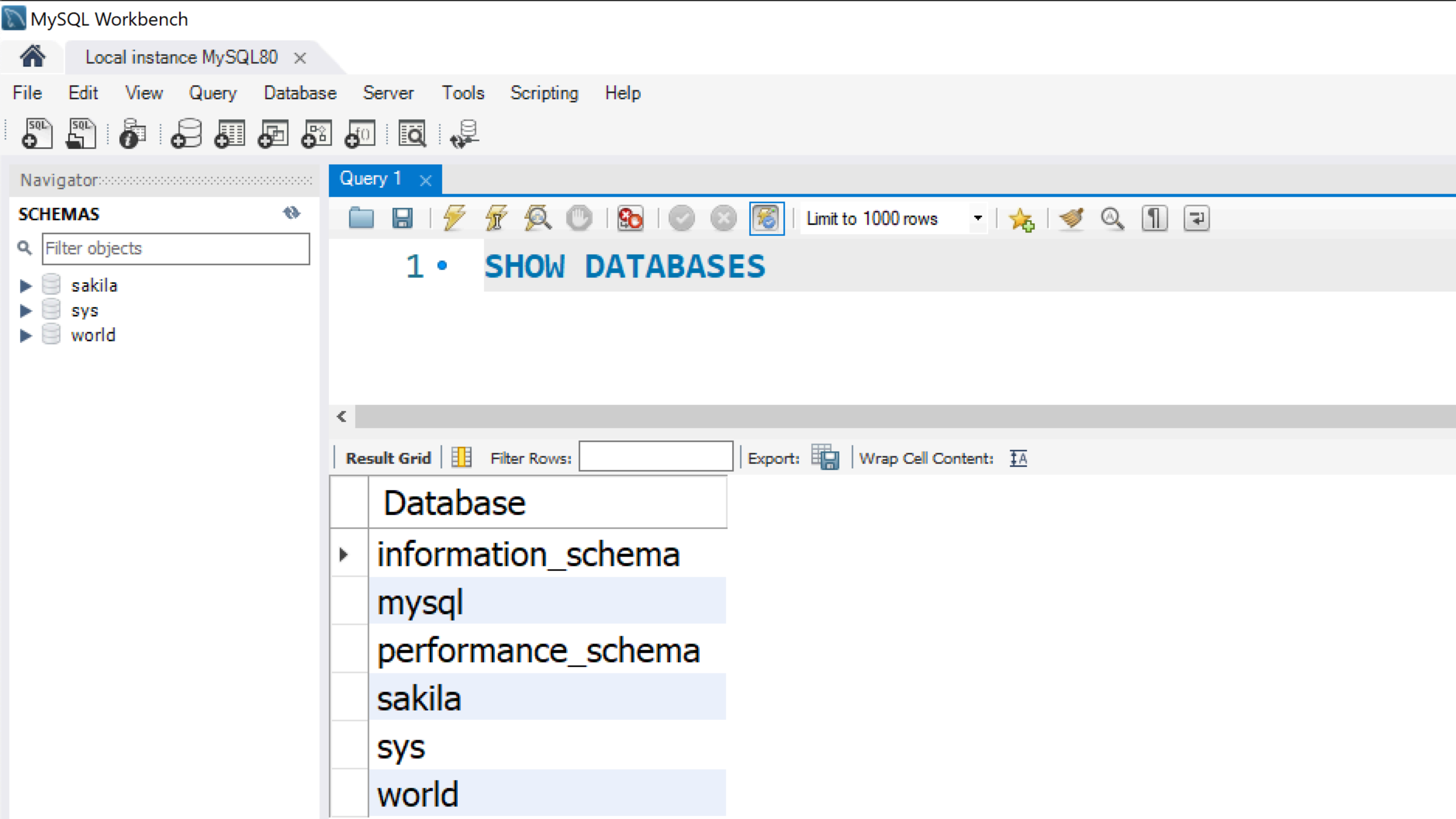
SELECT Syntax:

SELECT is used to retrieve rows selected from one or more tables, and can include UNION statements and subqueries. See union, and subqueries. A SELECT statement can start with a WITH clause to define common table expressions accessible within the SELECT. See with.

The most commonly used clauses of SELECT statements are these:

- Each ***select_expr*** indicates a column that you want to retrieve. There must be at least one ***select_expr***.
- ***table_references*** indicates the table or tables from which to retrieve rows. Its syntax is described in join.
- SELECT supports explicit partition

#	Time	Action	Message	Duration / Fetch
1	11:25:26	SELECT * FROM world.country LIMIT 0, 1000	239 row(s) returned	0.000 sec / 0.000 sec

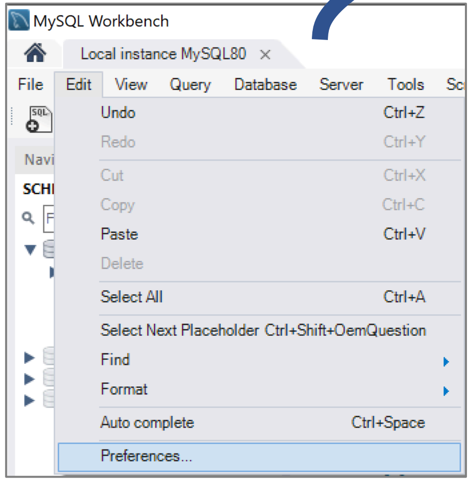


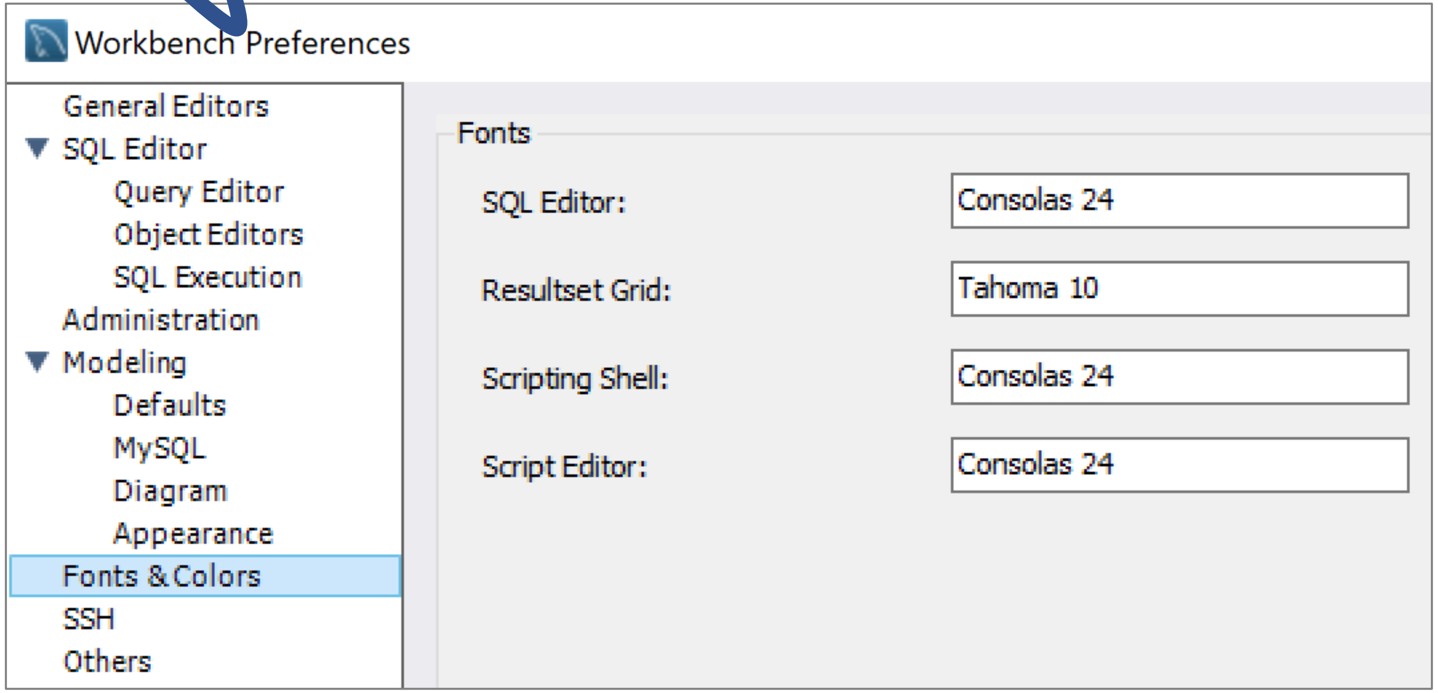
Limit to 1000 rows

```
SHOW DATABASES
```

Database
information_schema
mysql
performance_schema
sakila
sys
world

Customize





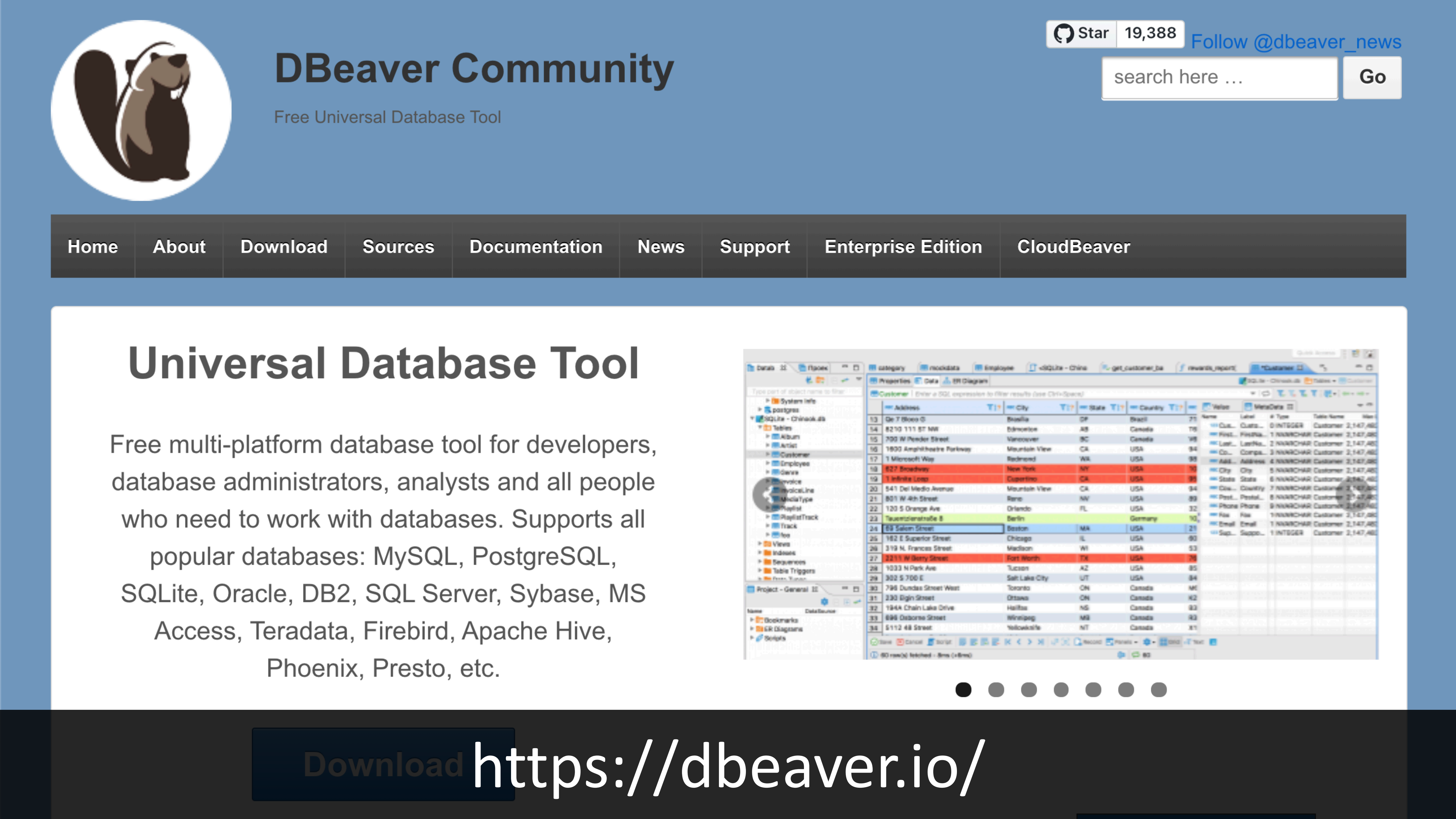
Fonts

SQL Editor: Consolas 24

Resultset Grid: Tahoma 10

Scripting Shell: Consolas 24

Script Editor: Consolas 24



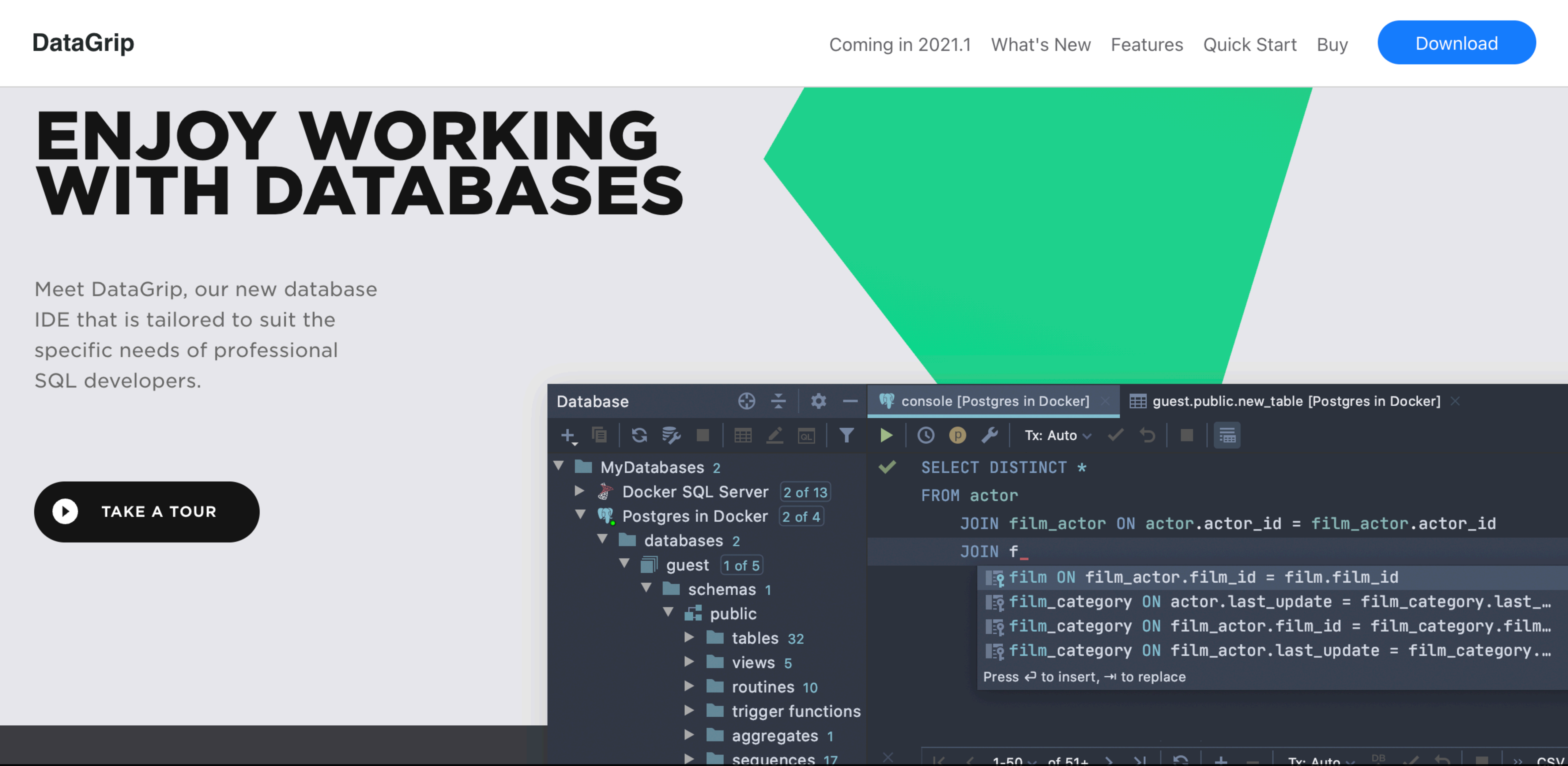
DBeaver Community

Free Universal Database Tool

Universal Database Tool

Free multi-platform database tool for developers, database administrators, analysts and all people who need to work with databases. Supports all popular databases: MySQL, PostgreSQL, SQLite, Oracle, DB2, SQL Server, Sybase, MS Access, Teradata, Firebird, Apache Hive, Phoenix, Presto, etc.

https://dbeaver.io/



ENJOY WORKING WITH DATABASES

Meet DataGrip, our new database IDE that is tailored to suit the specific needs of professional SQL developers.

```
SELECT DISTINCT *
FROM actor
    JOIN film_actor ON actor.actor_id = film_actor.actor_id
    JOIN f_
        film ON film_actor.film_id = film.film_id
        film_category ON actor.last_update = film_category.last_…
        film_category ON film_actor.film_id = film_category.film…
        film_category ON film_actor.last_update = film_category.…
        Press ⏎ to insert, ⇥ to replace
```

https://www.jetbrains.com/datagrip/

SQL Statements

SQL division of tasks

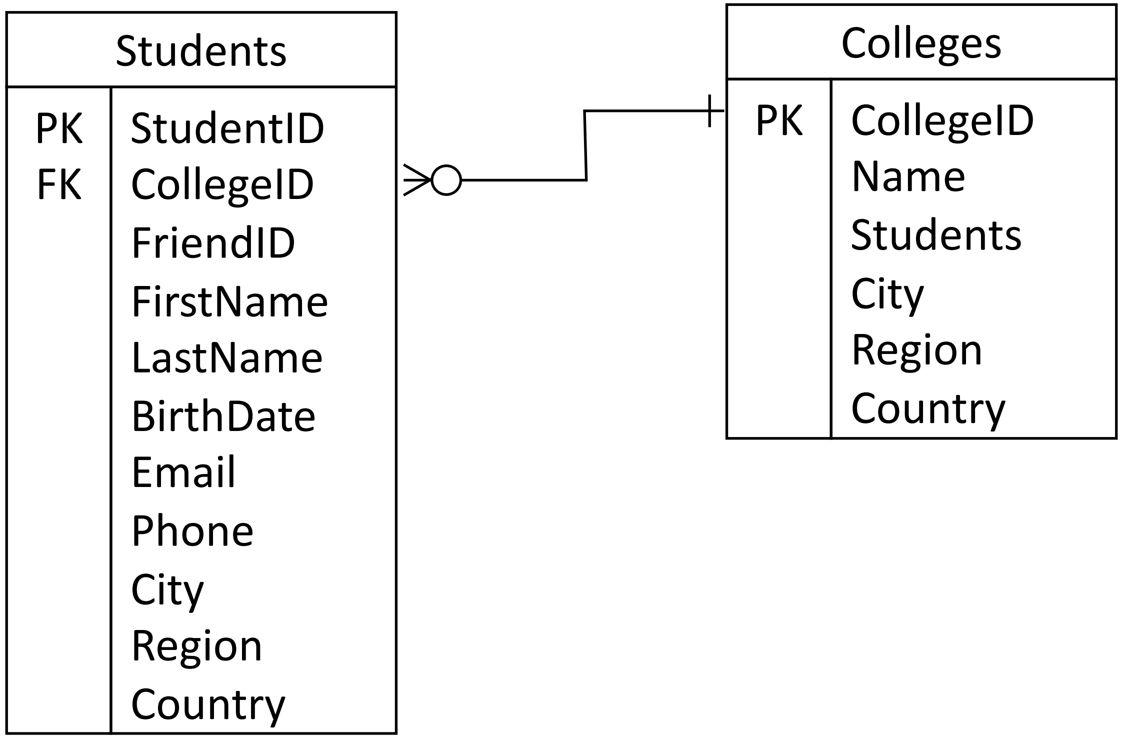
DQL, data query language, is used for querying data - includes like SELECT.

DDL, data definition language, is used to create and modify tables, views, users, other objects in the database. It affects the structure, but not the contents. There are three common commands: CREATE, ALTER, and DROP.

DCL, data control language, is used for access control.

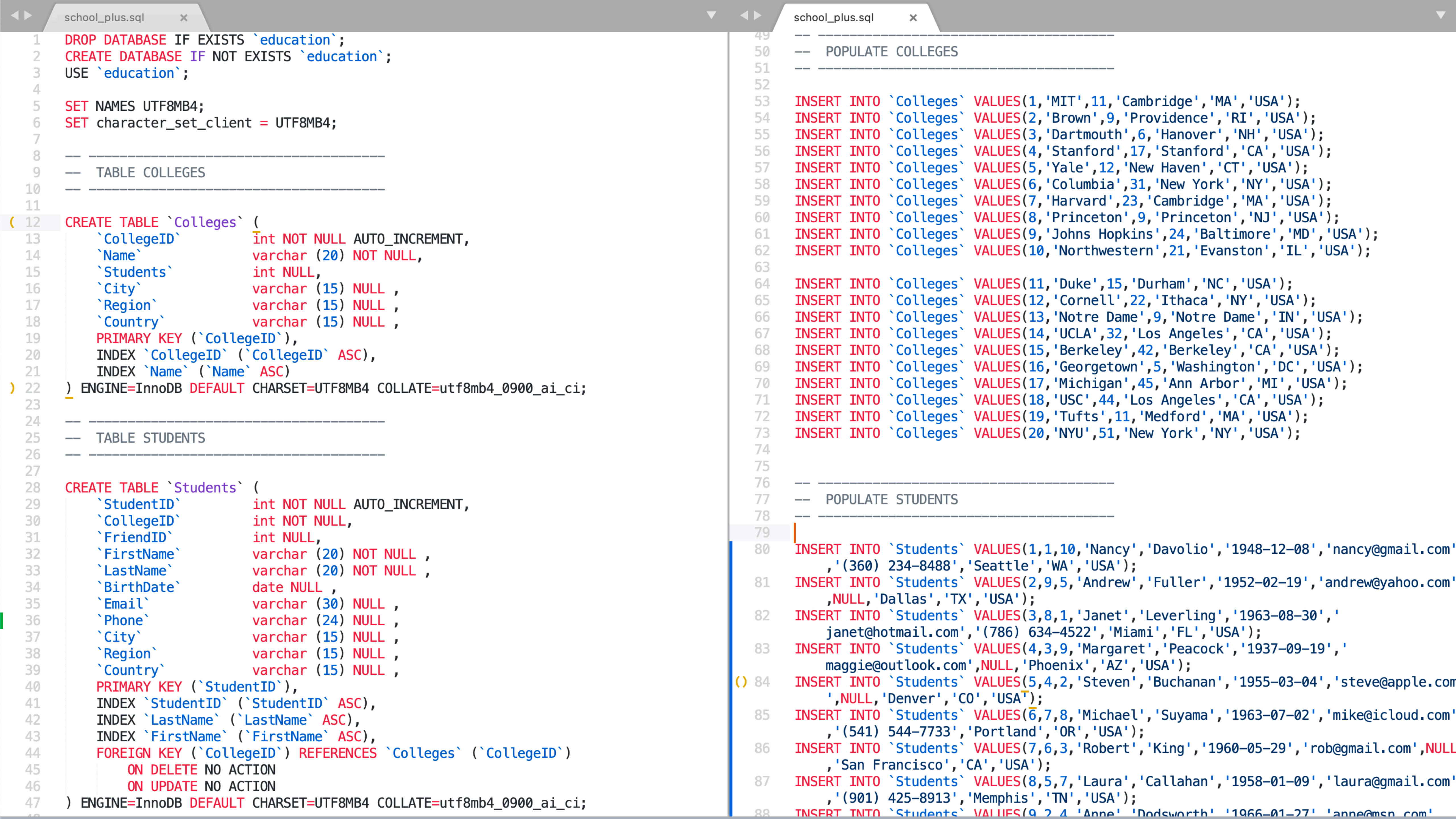
DML, data manipulation language, is used to act on the data itself. The commands are INSERT, UPDATE, and DELETE.

Simple database



Students	
PK	StudentID
FK	CollegeID
	FriendID
	FirstName
	LastName
	BirthDate
	Email
	Phone
	City
	Region
	Country

Colleges	
PK	CollegeID
	Name
	Students
	City
	Region
	Country



```sql
DROP DATABASE IF EXISTS `education`;
CREATE DATABASE IF NOT EXISTS `education`;
USE `education`;

SET NAMES UTF8MB4;
SET character_set_client = UTF8MB4;

-- ----------------------------------------
--  TABLE COLLEGES
-- ----------------------------------------

CREATE TABLE `Colleges` (
    `CollegeID`            int NOT NULL AUTO_INCREMENT,
    `Name`                 varchar (20) NOT NULL,
    `Students`             int NULL,
    `City`                 varchar (15) NULL ,
    `Region`               varchar (15) NULL ,
    `Country`              varchar (15) NULL ,
    PRIMARY KEY (`CollegeID`),
    INDEX `CollegeID` (`CollegeID` ASC),
    INDEX `Name` (`Name` ASC)
) ENGINE=InnoDB DEFAULT CHARSET=UTF8MB4 COLLATE=utf8mb4_0900_ai_ci;

-- ----------------------------------------
--  TABLE STUDENTS
-- ----------------------------------------

CREATE TABLE `Students` (
    `StudentID`            int NOT NULL AUTO_INCREMENT,
    `CollegeID`            int NOT NULL,
    `FriendID`             int NULL,
    `FirstName`            varchar (20) NOT NULL ,
    `LastName`             varchar (20) NOT NULL ,
    `BirthDate`            date NULL ,
    `Email`                varchar (30) NULL ,
    `Phone`                varchar (24) NULL ,
    `City`                 varchar (15) NULL ,
    `Region`               varchar (15) NULL ,
    `Country`              varchar (15) NULL ,
    PRIMARY KEY (`StudentID`),
    INDEX `StudentID` (`StudentID` ASC),
    INDEX `LastName` (`LastName` ASC),
    INDEX `FirstName` (`FirstName` ASC),
    FOREIGN KEY (`CollegeID`) REFERENCES `Colleges` (`CollegeID`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=UTF8MB4 COLLATE=utf8mb4_0900_ai_ci;
```

```sql
-- ----------------------------------------
--  POPULATE COLLEGES
-- ----------------------------------------

INSERT INTO `Colleges` VALUES(1,'MIT',11,'Cambridge','MA','USA');
INSERT INTO `Colleges` VALUES(2,'Brown',9,'Providence','RI','USA');
INSERT INTO `Colleges` VALUES(3,'Dartmouth',6,'Hanover','NH','USA');
INSERT INTO `Colleges` VALUES(4,'Stanford',17,'Stanford','CA','USA');
INSERT INTO `Colleges` VALUES(5,'Yale',12,'New Haven','CT','USA');
INSERT INTO `Colleges` VALUES(6,'Columbia',31,'New York','NY','USA');
INSERT INTO `Colleges` VALUES(7,'Harvard',23,'Cambridge','MA','USA');
INSERT INTO `Colleges` VALUES(8,'Princeton',9,'Princeton','NJ','USA');
INSERT INTO `Colleges` VALUES(9,'Johns Hopkins',24,'Baltimore','MD','USA');
INSERT INTO `Colleges` VALUES(10,'Northwestern',21,'Evanston','IL','USA');

INSERT INTO `Colleges` VALUES(11,'Duke',15,'Durham','NC','USA');
INSERT INTO `Colleges` VALUES(12,'Cornell',22,'Ithaca','NY','USA');
INSERT INTO `Colleges` VALUES(13,'Notre Dame',9,'Notre Dame','IN','USA');
INSERT INTO `Colleges` VALUES(14,'UCLA',32,'Los Angeles','CA','USA');
INSERT INTO `Colleges` VALUES(15,'Berkeley',42,'Berkeley','CA','USA');
INSERT INTO `Colleges` VALUES(16,'Georgetown',5,'Washington','DC','USA');
INSERT INTO `Colleges` VALUES(17,'Michigan',45,'Ann Arbor','MI','USA');
INSERT INTO `Colleges` VALUES(18,'USC',44,'Los Angeles','CA','USA');
INSERT INTO `Colleges` VALUES(19,'Tufts',11,'Medford','MA','USA');
INSERT INTO `Colleges` VALUES(20,'NYU',51,'New York','NY','USA');


-- ----------------------------------------
--  POPULATE STUDENTS
-- ----------------------------------------

INSERT INTO `Students` VALUES(1,1,10,'Nancy','Davolio','1948-12-08','nancy@gmail.com'
    ,'(360) 234-8488','Seattle','WA','USA');
INSERT INTO `Students` VALUES(2,9,5,'Andrew','Fuller','1952-02-19','andrew@yahoo.com'
    ,NULL,'Dallas','TX','USA');
INSERT INTO `Students` VALUES(3,8,1,'Janet','Leverling','1963-08-30','
    janet@hotmail.com','(786) 634-4522','Miami','FL','USA');
INSERT INTO `Students` VALUES(4,3,9,'Margaret','Peacock','1937-09-19','
    maggie@outlook.com',NULL,'Phoenix','AZ','USA');
INSERT INTO `Students` VALUES(5,4,2,'Steven','Buchanan','1955-03-04','steve@apple.com
    ',NULL,'Denver','CO','USA');
INSERT INTO `Students` VALUES(6,7,8,'Michael','Suyama','1963-07-02','mike@icloud.com'
    ,'(541) 544-7733','Portland','OR','USA');
INSERT INTO `Students` VALUES(7,6,3,'Robert','King','1960-05-29','rob@gmail.com',NULL
    ,'San Francisco','CA','USA');
INSERT INTO `Students` VALUES(8,5,7,'Laura','Callahan','1958-01-09','laura@gmail.com'
    ,'(901) 425-8913','Memphis','TN','USA');
INSERT INTO `Students` VALUES(9,2,4,'Anne','Dodsworth','1966-01-27','anne@msn.com'
```

Download installation script

`http://bit.ly/3tiLF3H`

SQL Statements

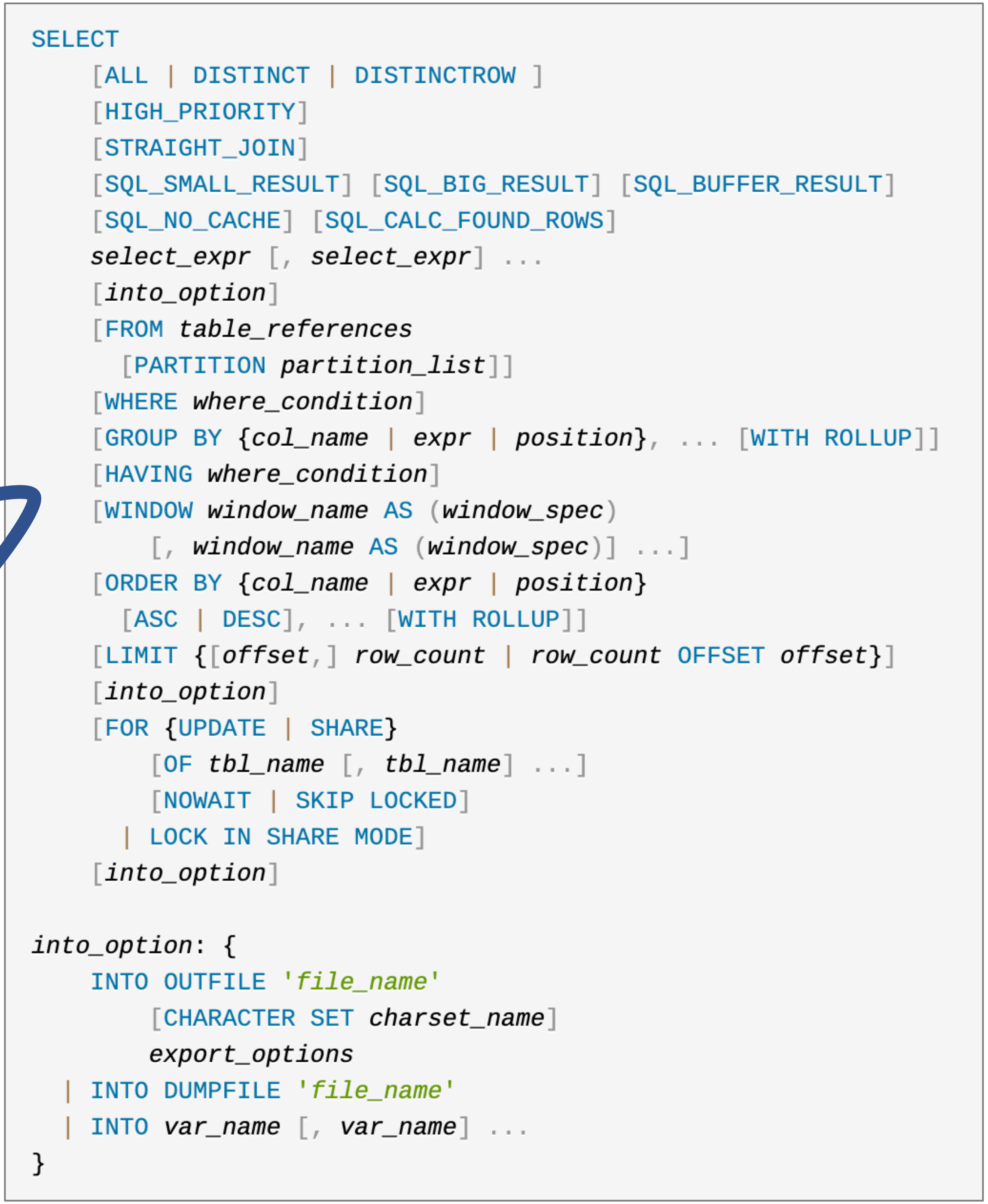
Clauses

CLAUSE	PURPOSE
SELECT	Selects which columns to include
FROM	The tables from which to retrieve data
WHERE	Filters out unwanted data
GROUP BY	Groups rows together
HAVING	Filters groups
ORDER BY	Sorts rows

SELECT

https://dev.mysql.com/doc/

This block
is a career

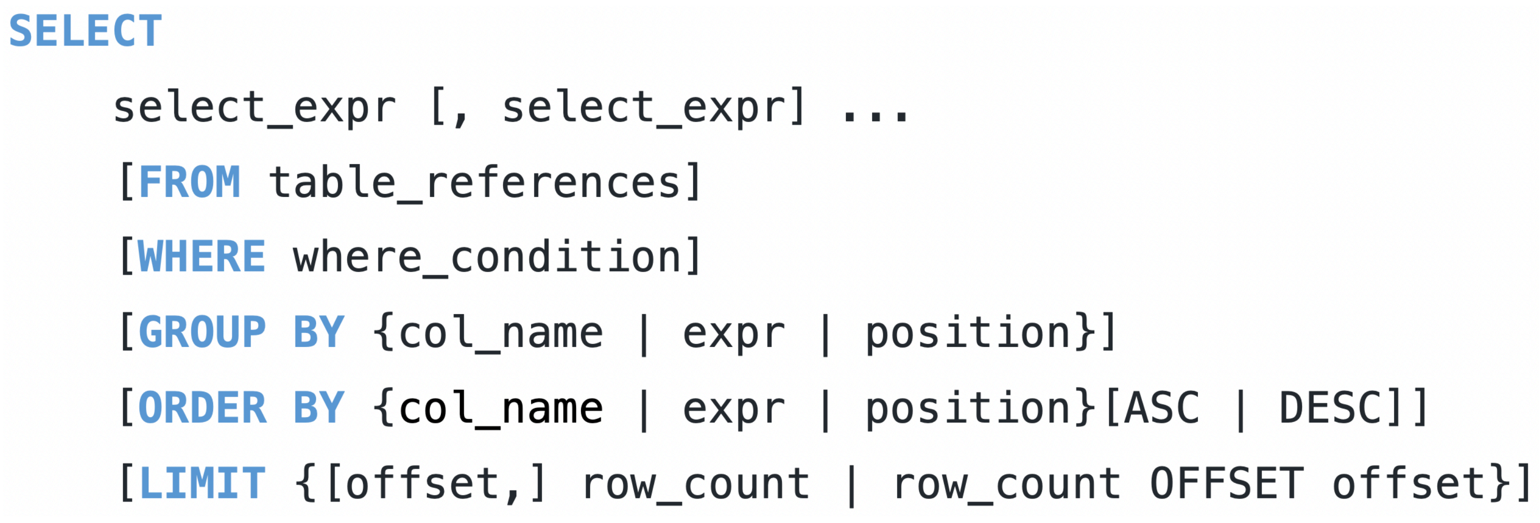


```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
    [HIGH_PRIORITY]
    [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
    [SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr] ...
    [into_option]
    [FROM table_references
      [PARTITION partition_list]]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
    [HAVING where_condition]
    [WINDOW window_name AS (window_spec)
        [, window_name AS (window_spec)] ...]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [into_option]
    [FOR {UPDATE | SHARE}
        [OF tbl_name [, tbl_name] ...]
        [NOWAIT | SKIP LOCKED]
      | LOCK IN SHARE MODE]
    [into_option]

into_option: {
    INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
  | INTO DUMPFILE 'file_name'
  | INTO var_name [, var_name] ...
}
```

Keep clauses in syntactical order

Order of Clauses Matters



```
SELECT
    select_expr [, select_expr] ...
    [FROM table_references]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}]
    [ORDER BY {col_name | expr | position}[ASC | DESC]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

Statements

Several clauses make up a select statement.

You will often use two or three clauses.

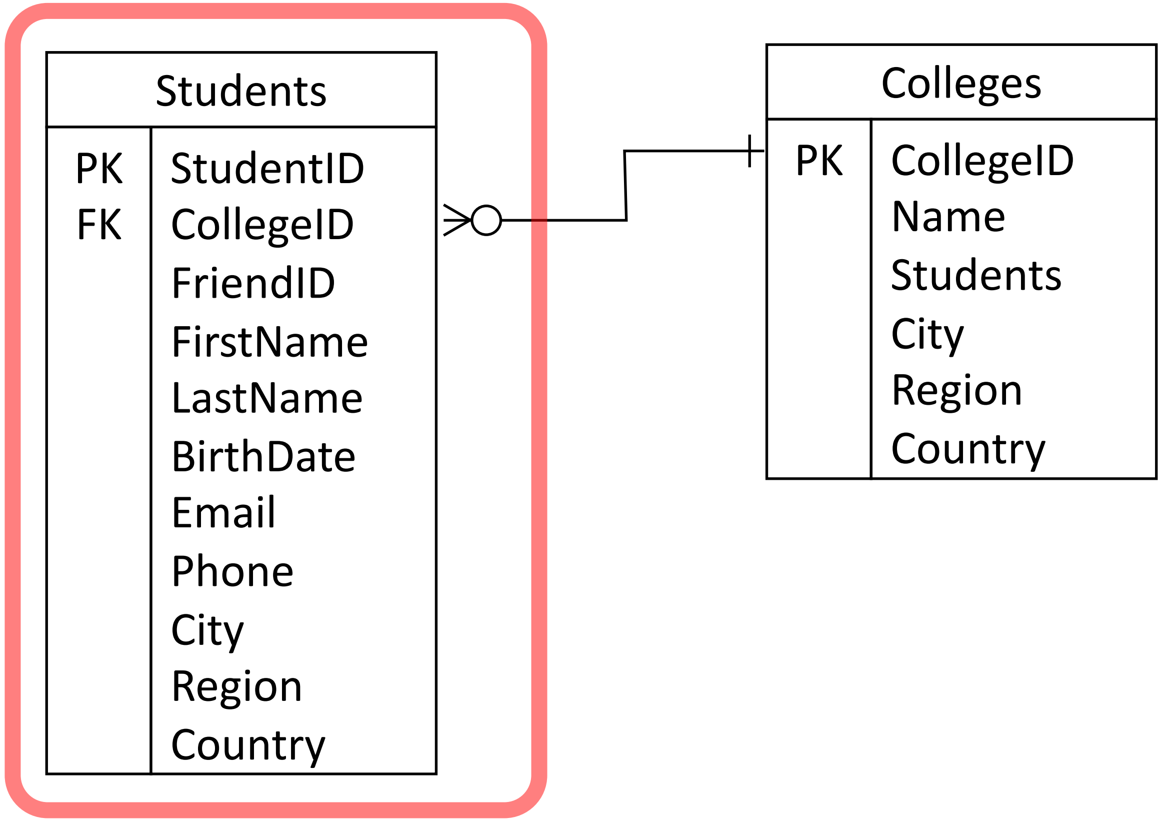
Order of clauses matters.

Only the select clause is mandatory.

QUERY

SQL statement

Simple database



Students	
PK	StudentID
FK	CollegeID
	FriendID
	FirstName
	LastName
	BirthDate
	Email
	Phone
	City
	Region
	Country

Colleges	
PK	CollegeID
	Name
	Students
	City
	Region
	Country

Simple select

SELECT *columns*

FROM *table*

Select

```
SELECT columns
FROM table
WHERE condition
ORDER BY columns
```

SELECT

Retrieving records

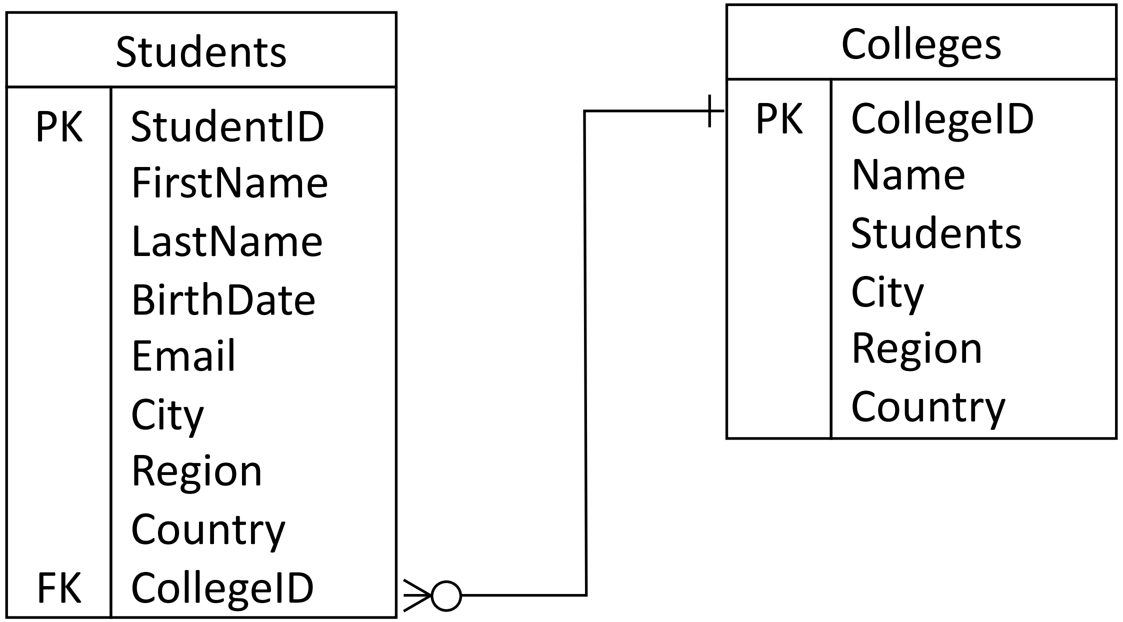
Case sensitive – some times

- SQL Keywords – case insensitive but usually written in caps
- Tables and columns are case sensitive depending on platform/OS
- Example, MySQL case sensitive on Linux, insensitive on Windows

Select basics

SELECT *list_of_columns*
FROM *table[s]*
[**WHERE** *search_conditions*]

SELECT *
FROM Colleges
WHERE City=‘Cambridge’



Students
PK StudentID
FirstName
LastName
BirthDate
Email
City
Region
Country
FK CollegeID

Colleges
PK CollegeID
Name
Students
City
Region
Country

Specify columns

```
SELECT FirstName, LastName
FROM Students
```

Renaming columns and naming expressions

```
-- rename column
SELECT  Name AS University
FROM Colleges

-- expression plus rename
SELECT Name AS University, Students*1000 AS ‘number of students’
FROM Colleges
```

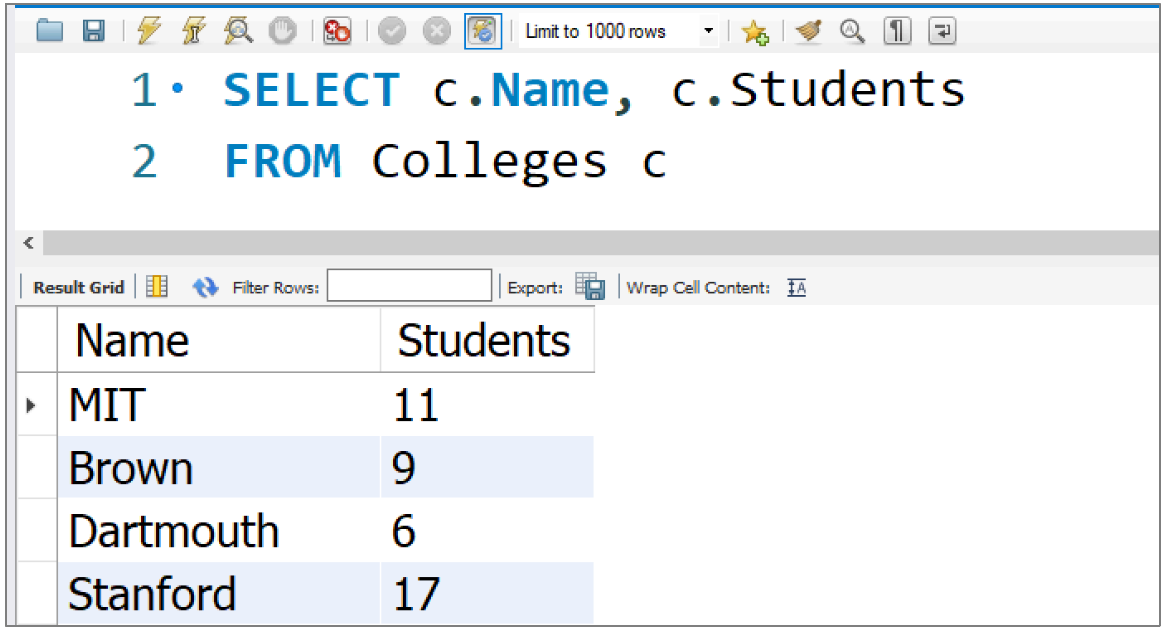
Pretty name for column

```
SELECT FirstName,
       LastName,
       CONCAT(FirstName, " ", LastName) AS FullName
FROM students
```

A *select_expr* can be given an alias using AS *alias_name*. The alias is used as the expression's column name and can be used in GROUP BY, ORDER BY, or HAVING clauses.

Table aliases

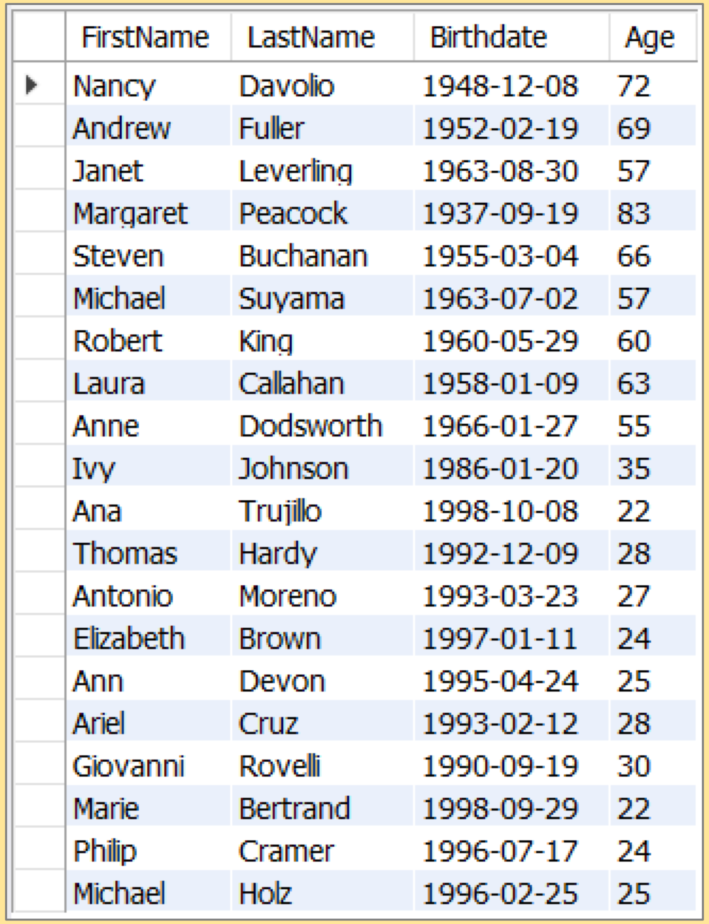
```
SELECT c.Name, c.Students
FROM Colleges c
```



```
SELECT c.Name, c.Students
FROM Colleges c
```

Name	Students
MIT	11
Brown	9
Dartmouth	6
Stanford	17

Active learning: write the query for the following output



	FirstName	LastName	Birthdate	Age
▸	Nancy	Davolio	1948-12-08	72
	Andrew	Fuller	1952-02-19	69
	Janet	Leverling	1963-08-30	57
	Margaret	Peacock	1937-09-19	83
	Steven	Buchanan	1955-03-04	66
	Michael	Suyama	1963-07-02	57
	Robert	King	1960-05-29	60
	Laura	Callahan	1958-01-09	63
	Anne	Dodsworth	1966-01-27	55
	Ivy	Johnson	1986-01-20	35
	Ana	Trujillo	1998-10-08	22
	Thomas	Hardy	1992-12-09	28
	Antonio	Moreno	1993-03-23	27
	Elizabeth	Brown	1997-01-11	24
	Ann	Devon	1995-04-24	25
	Ariel	Cruz	1993-02-12	28
	Giovanni	Rovelli	1990-09-19	30
	Marie	Bertrand	1998-09-29	22
	Philip	Cramer	1996-07-17	24
	Michael	Holz	1996-02-25	25

You can use TIMESTAMPDIFF(unit, datetime_expr1, datetime_expr2)

Student age

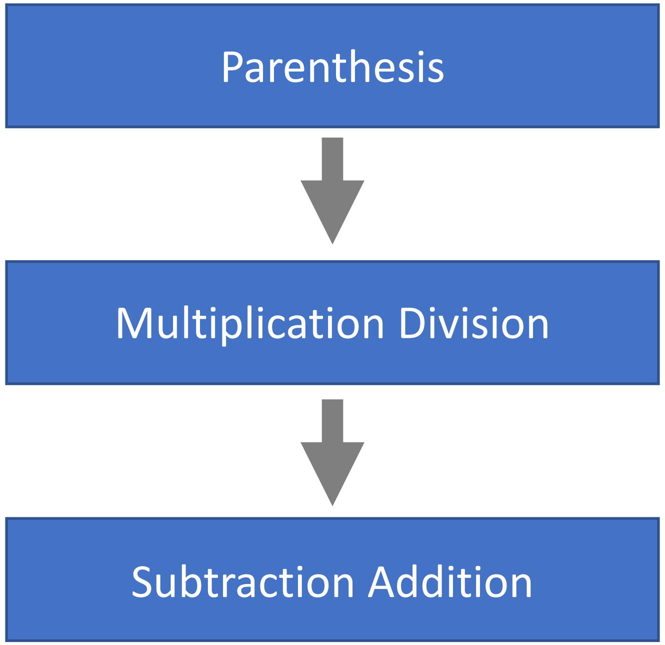
```
SELECT FirstName,
       LastName,
       Birthdate,
        TIMESTAMPDIFF(YEAR, Birthdate,now()) AS Age
FROM Students
```

* TIMESTAMPDIFF(unit, datetime_expr1, datetime_expr2)

Distinct

```
SELECT DISTINCT Region
FROM Colleges
```

Arithmetic Operator Precedence

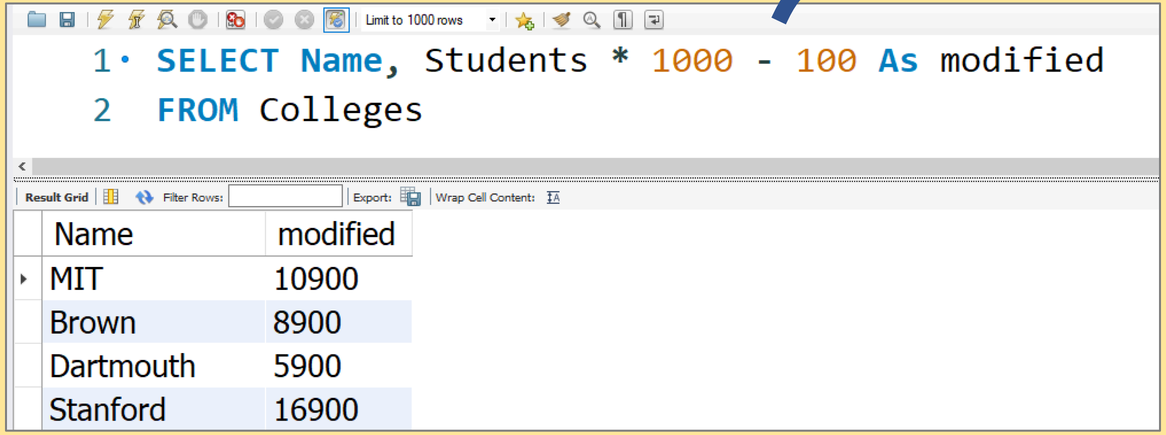


Parenthesis

Multiplication Division

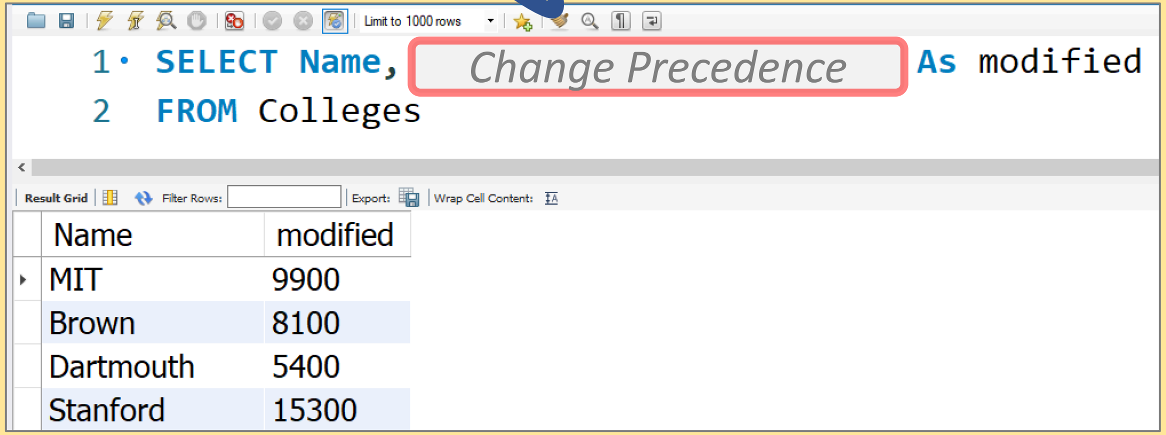
Subtraction Addition

Active learning: change precedence to match output



```
SELECT Name, Students * 1000 - 100 As modified
FROM Colleges
```

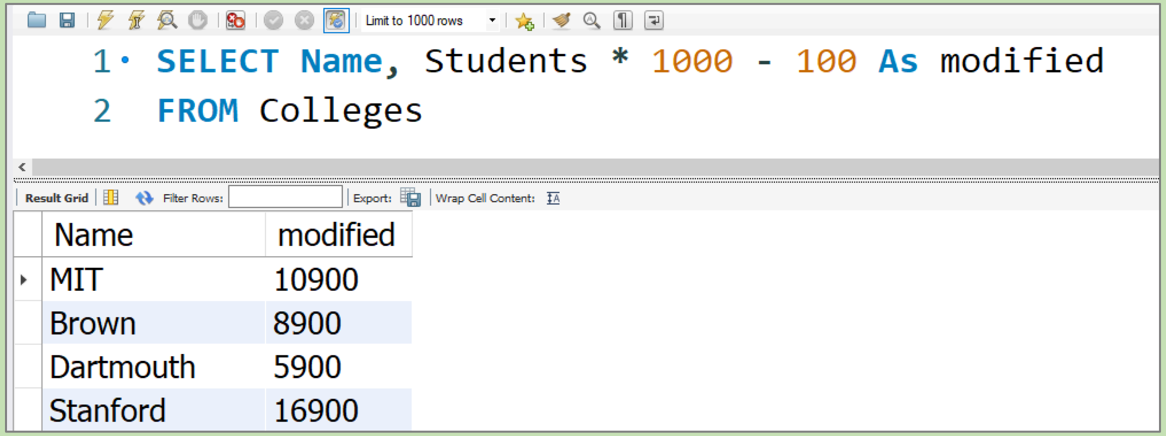
Name	modified
MIT	10900
Brown	8900
Dartmouth	5900
Stanford	16900



```
SELECT Name, Change Precedence As modified
FROM Colleges
```

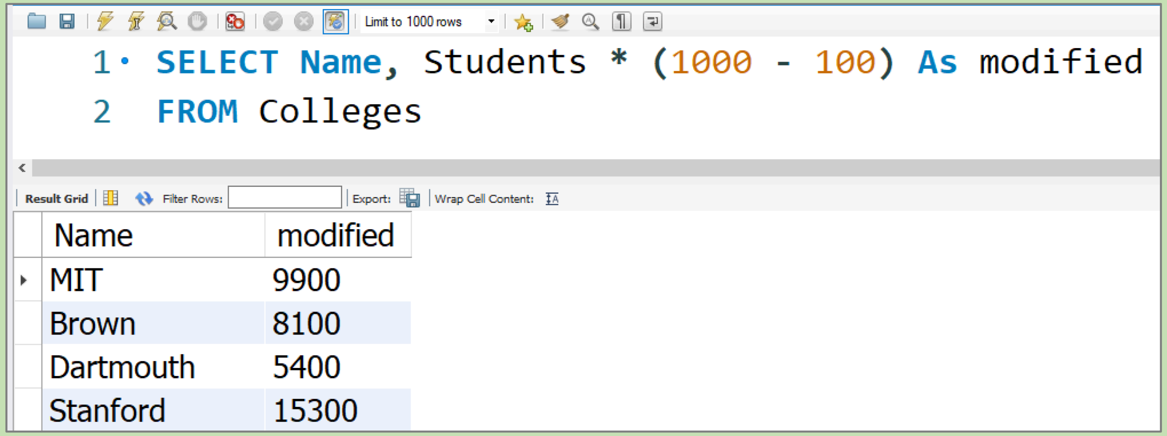
Name	modified
MIT	9900
Brown	8100
Dartmouth	5400
Stanford	15300

Precedence example



```sql
SELECT Name, Students * 1000 - 100 As modified
FROM Colleges
```

Name	modified
MIT	10900
Brown	8900
Dartmouth	5900
Stanford	16900



```sql
SELECT Name, Students * (1000 - 100) As modified
FROM Colleges
```

Name	modified
MIT	9900
Brown	8100
Dartmouth	5400
Stanford	15300

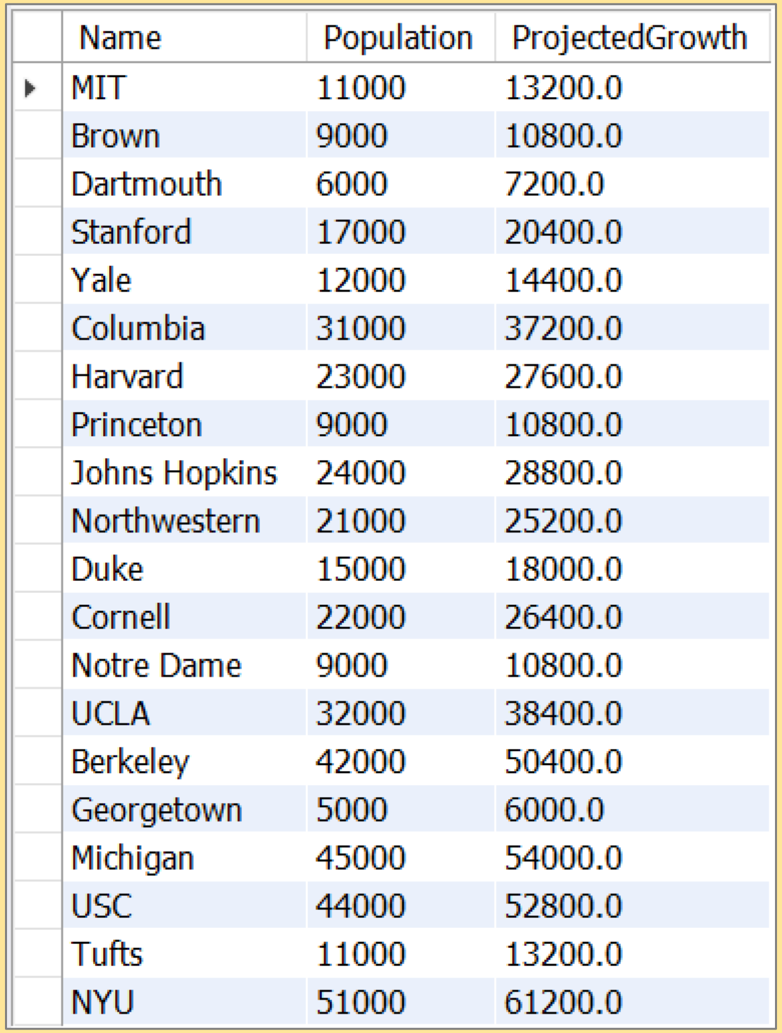
Active learning

Return:

College name

Student population * 1000

Projected growth - 20% increase



	Name	Population	ProjectedGrowth
▸	MIT	11000	13200.0
	Brown	9000	10800.0
	Dartmouth	6000	7200.0
	Stanford	17000	20400.0
	Yale	12000	14400.0
	Columbia	31000	37200.0
	Harvard	23000	27600.0
	Princeton	9000	10800.0
	Johns Hopkins	24000	28800.0
	Northwestern	21000	25200.0
	Duke	15000	18000.0
	Cornell	22000	26400.0
	Notre Dame	9000	10800.0
	UCLA	32000	38400.0
	Berkeley	42000	50400.0
	Georgetown	5000	6000.0
	Michigan	45000	54000.0
	USC	44000	52800.0
	Tufts	11000	13200.0
	NYU	51000	61200.0

Active learning

```
SELECT
		Name,
	Students * 1000 AS Population,
	Students * 1000 * 1.2 AS ProjectedGrowth
```

WHERE

The where clause

The where clause specifies the search conditions

```
SELECT columns_list
FROM table_list
WHERE search_conditions
```

Where region is equal to

```
SELECT *
FROM Students
WHERE Region = 'TX'
```

Search condition categories

- Comparison operators – e.g. =,<,>)
- Logical operators – e.g. AND,OR, NOT
- Ranges – between and not between
- Lists – IN, NOT IN
- Unknown values – IS NULL, IS NOT NULL
- Character matches – LIKE, NOT LIKE

Comparison operators

WHERE *expression comparison_operator expression*

Name	**Description**
>	Greater than operator
>=	Greater than or equal operator
<	Less than operator
<>, !=	Not equal operator
<=	Less than or equal operator
<=>	NULL-safe equal to operator
=	Equal operator

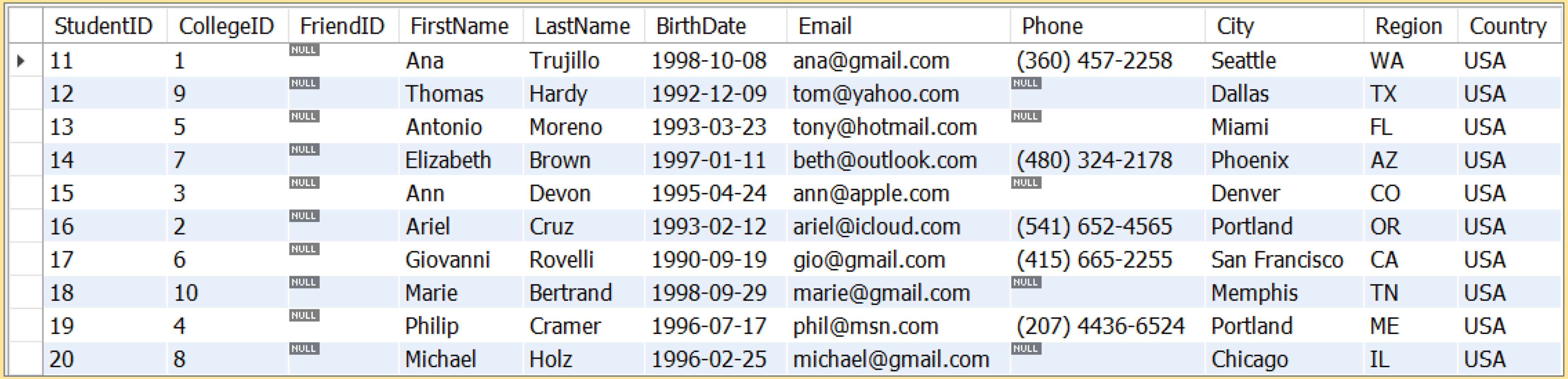
Where region is not equal to …

```
SELECT *
FROM Students
WHERE Region <> 'TX'
```

* Not case sensitive

Active learning

Find students born after January 1st, 1990



	StudentID	CollegeID	FriendID	FirstName	LastName	BirthDate	Email	Phone	City	Region	Country
▸	11	1	NULL	Ana	Trujillo	1998-10-08	ana@gmail.com	(360) 457-2258	Seattle	WA	USA
	12	9	NULL	Thomas	Hardy	1992-12-09	tom@yahoo.com	NULL	Dallas	TX	USA
	13	5	NULL	Antonio	Moreno	1993-03-23	tony@hotmail.com	NULL	Miami	FL	USA
	14	7	NULL	Elizabeth	Brown	1997-01-11	beth@outlook.com	(480) 324-2178	Phoenix	AZ	USA
	15	3	NULL	Ann	Devon	1995-04-24	ann@apple.com	NULL	Denver	CO	USA
	16	2	NULL	Ariel	Cruz	1993-02-12	ariel@icloud.com	(541) 652-4565	Portland	OR	USA
	17	6	NULL	Giovanni	Rovelli	1990-09-19	gio@gmail.com	(415) 665-2255	San Francisco	CA	USA
	18	10	NULL	Marie	Bertrand	1998-09-29	marie@gmail.com	NULL	Memphis	TN	USA
	19	4	NULL	Philip	Cramer	1996-07-17	phil@msn.com	(207) 4436-6524	Portland	ME	USA
	20	8	NULL	Michael	Holz	1996-02-25	michael@gmail.com	NULL	Chicago	IL	USA

Active learning

```
SELECT *
FROM Students
WHERE BirthDate > '1990-01-01'
```

Where birth date is greater than

```sql
SELECT *
FROM Students
WHERE BirthDate > '1996-01-01'
```

`* Date format 'YYYY-MM-DD'`

Comparison operators

Name	**Description**
BETWEEN ... AND ...	Value is within a range
COALESCE()	Return the first non-NULL argument
GREATEST()	Return the largest argument
IN()	Value is within a set of values
INTERVAL()	Index of the argument that is less
IS	Test a value against a boolean
IS NOT	Test a value against a boolean
IS NOT NULL	NOT NULL value test
IS NULL	NULL value test
ISNULL()	Test whether the argument is NULL
LEAST()	Return the smallest argument
LIKE	Simple pattern matching
NOT BETWEEN ... AND ...	Value is not within a range of values
NOT IN()	Value is not within a set of values
NOT LIKE	Negation of simple pattern matching
STRCMP()	Compare two strings

Conditions are composed of expressions and operators

Boolean expressions	fetch the data based on matching a single value

SELECT *column*
FROM *table_name*
WHERE *single_value_matching_expression*	

Example: SELECT * FROM Colleges WHERE Region = 'MA'	

Numeric expressions	Perform mathematical operation in a query

SELECT numerical_expression as operation_name
FROM table_name	

Example: SELECT (4 + 3) AS Addition	

Date expressions	Results in datetime value
Example: SELECT now()	

LOGICAL OPERATORS

AND, NOT, OR

Logical operators

Name	**Description**
AND, &&	Logical AND
NOT, !	Negates value
OR, ||	Logical OR
XOR	Logical XOR

AND

```sql
SELECT *
FROM Students
WHERE
    BirthDate > '1990-01-01' AND Region = 'TX'
```

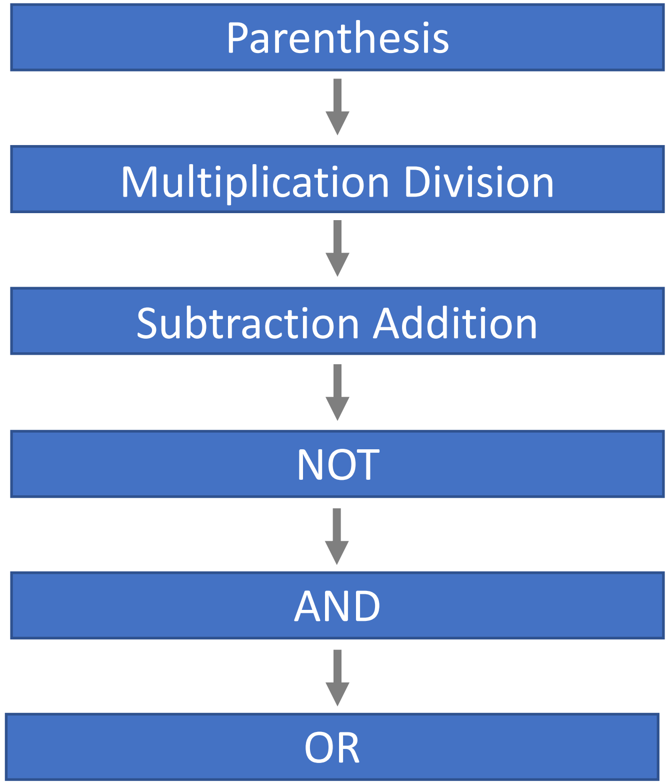
OR

```
SELECT *
FROM Students
WHERE
    BirthDate > '1990-01-01' OR Region = 'TX'
```

NOT

```
SELECT *
FROM Students
WHERE
  NOT (BirthDate > '1990-01-01')
```

Logical Operator Precedence



Parenthesis

Multiplication Division

Subtraction Addition

NOT

AND

OR

Active learning

Find students born after 1950, from Texas, and Austin

Active learning

```
SELECT *
FROM Students
WHERE
    BirthDate > '1950-01-01'
    AND
    Region = 'TX'
    AND
    City = 'Austin'
```

IN

expr IN (value, …)

OR

```
SELECT *
FROM Students
WHERE
    Region = 'AZ' OR
    Region = 'TX' OR
    Region = 'FL'
```

IN

```
SELECT *
FROM Students
WHERE Region IN ('AZ', 'TX', 'FL')
```

NOT IN

```
SELECT *
FROM students
WHERE region NOT IN ('Z', 'TX', 'FL')
```

Active learning

Students from Austin, Boston, and Chicago

Active learning

```
SELECT *
FROM Students
WHERE
   City IN ('Austin', 'Miami', 'Chicago')
```

BETWEEN

expr BETWEEN ***min*** AND ***max***

Where birth date is between … and …

```sql
SELECT *
FROM Students
WHERE
    StudentID >= 1 AND
    StudentID <= 5
```

Where StudentID is between … and …

```
SELECT *
FROM Students
WHERE StudentID
      BETWEEN 1 AND 5
```

Active learning

Find students born 01/01/1990 to 01/01/2000

Active learning

```
SELECT *
FROM Students
WHERE BirthDate
    BETWEEN '1990-01-01' AND '2000-01-01'
```

NULL

WHERE column IS NULL

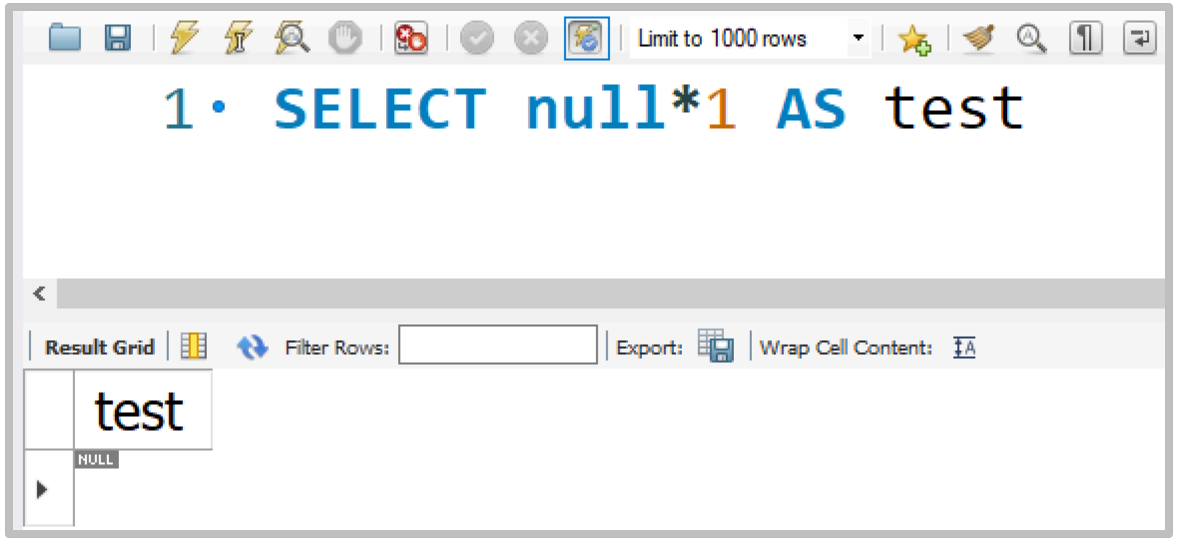
NULL is a placeholder for unknown information. It’s not zero or blank

```sql
SELECT *
FROM Students
WHERE Country IS NULL
```

* IS NOT NULL

Arithmetic operations on *null* are *null*

SELECT null*1 AS test



```
1 • SELECT null*1 AS test
```

```
test
NULL
```

Active learning

Find students with a phone

Active learning

```
SELECT *
FROM Students
WHERE Phone IS NOT NULL
```