TP-1

Université Grenoble Alpes

16.03.2023

bahareh.afshinpour@univ-grenoble-alpes.fr

Main reference:
A First Course in Database Systems (and associated material) by
J. Ullman and J. Widom, Prentice-Hall

VPN

Le VPN

L'utilisation du VPN vous permet à partir de n'importe quel réseau d'accéder aux ressources informatiques du LIG.

Le VPN est désormais proposé par l'Université Grenoble Alpes. La documentation se trouve ici.

Version courte :

- Le VPN de l'Université Grenoble Alpes est ici https://vpn.grenet.fr
- Connectez vous en tant que ***Personnel UGA***

Vous pouvez aussi utiliser les bastions ssh

Find your Oracle password

- **Le mot de passe Oracle n'est pas celui de votre compte universitaire.**
- You can find our documentation here (in French): https://im2ag-wiki.univ-grenoble-alpes.fr/doku.php?id=environnements:oracle

Connexion Oracle à partir de septembre 2022

Il faut vous connecter en ssh sur le serveur **im2ag-oracle.univ-grenoble-alpes.fr** avec vos login et mot de passe universitaires :

```
ssh login@im2ag-oracle.univ-grenoble-alpes.fr
```

Ensuite, prenez connaissance de votre mot de passe pour les bases de données Oracle. Il se trouve dans un fichier texte à la racine de votre HOME : **~/oracle.txt**. Votre HOME est monté sur le serveur Oracle, vous pouvez donc utiliser la commande suivante pour afficher votre mot de passe Oracle :

```
cat ~/oracle.txt
```

To connect to Oracle (DataGrip)

- DataGrip
- Please note that DataGrip is not free, but teachers and students of UGA can have the full version for free if you register on their website with your UGA address as a student/teacher account : https://www.jetbrains.com/datagrip/

- **hostname**:im2ag-oracle.univ-grenoble-alpes.fr
- **port**:1521
- **servicename**:im2ag

SQL – Structured Query Language

- The most commonly used relational DBMS'S query and modify the database through a language called SQL.
- The portion of SQL that supports queries has capabilities very close to relational algebra.

TP

NIH NATIONAL CANCER INSTITUTE

Search

About Cancer | Cancer Types | Research | Grants & Training | News & Events | About NCI

Home > About NCI > NCI Organization > CCG > Research > Structural Genomics > The Cancer Genome Atlas Program

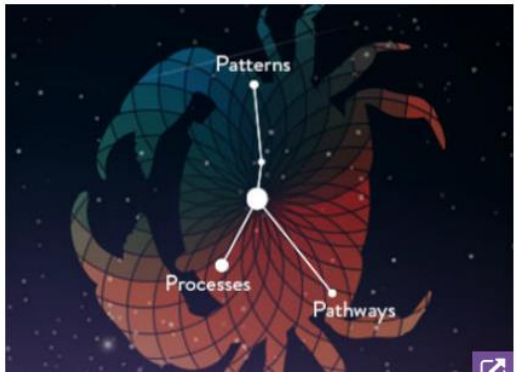
TCGA

- Program History
- TCGA Cancers Selected for Study
- Publications by TCGA
- Using TCGA
- Contact

The Cancer Genome Atlas Program

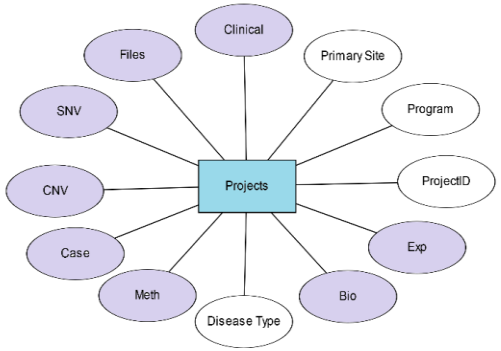
The Cancer Genome Atlas (TCGA), a landmark cancer genomics program, molecularly characterized over 20,000 primary cancer and matched normal samples spanning 33 cancer types. This joint effort between NCI and the National Human Genome Research Institute began in 2006, bringing together researchers from diverse disciplines and multiple institutions.

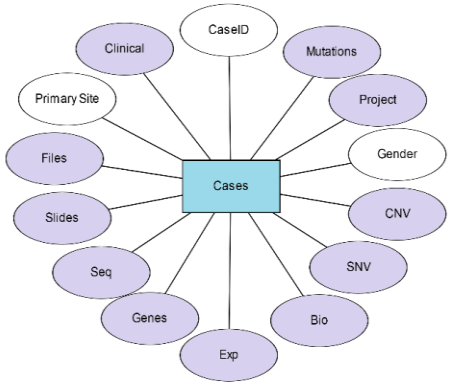
Over the next dozen years, TCGA generated over 2.5 petabytes of genomic, epigenomic, transcriptomic, and proteomic data. The data, which has already led to improvements in our ability to diagnose, treat, and prevent cancer, will remain publicly available for anyone in the research community to use.

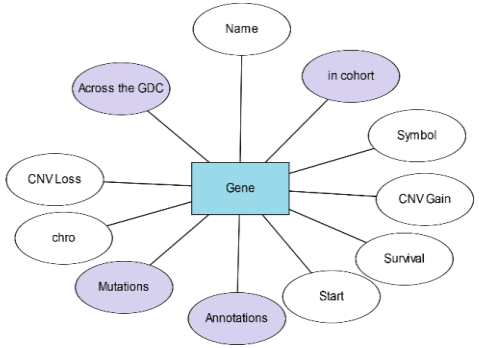


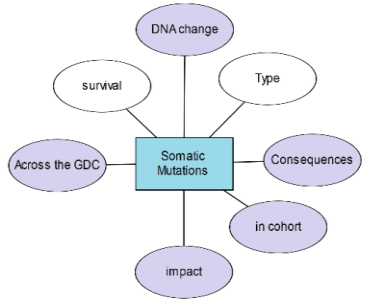
https://www.cancer.gov/about-nci/organization/ccg/research/structural-genomics/tcga

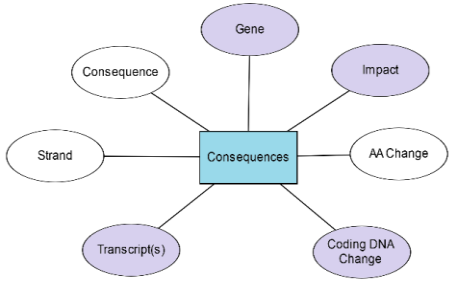
ER Model Description

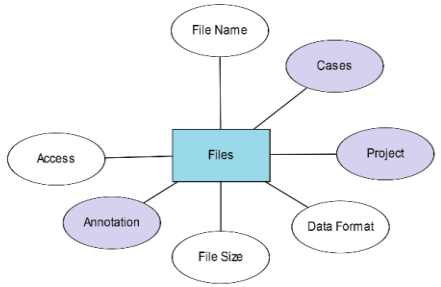


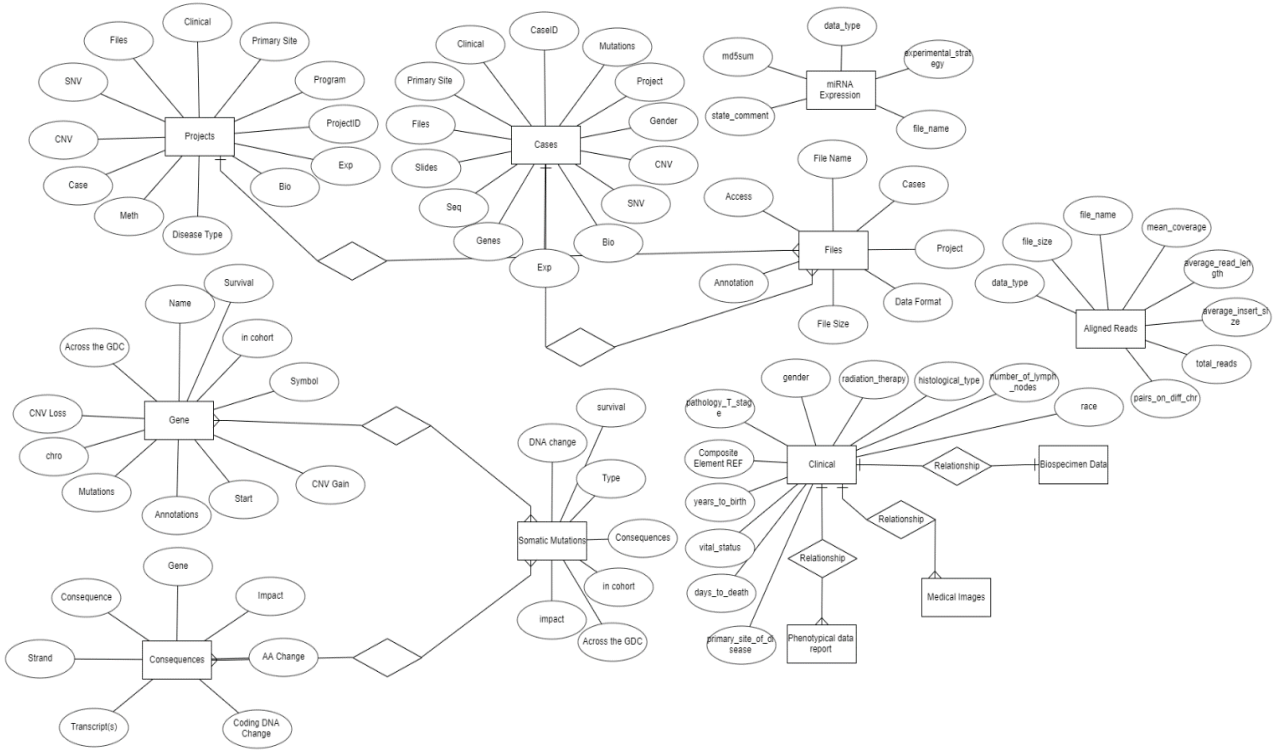












CREATE TABLE

```
CREATE TABLE table_name (
   column1 datatype,
   column2 datatype,
   column3 datatype,
  ....
);
```

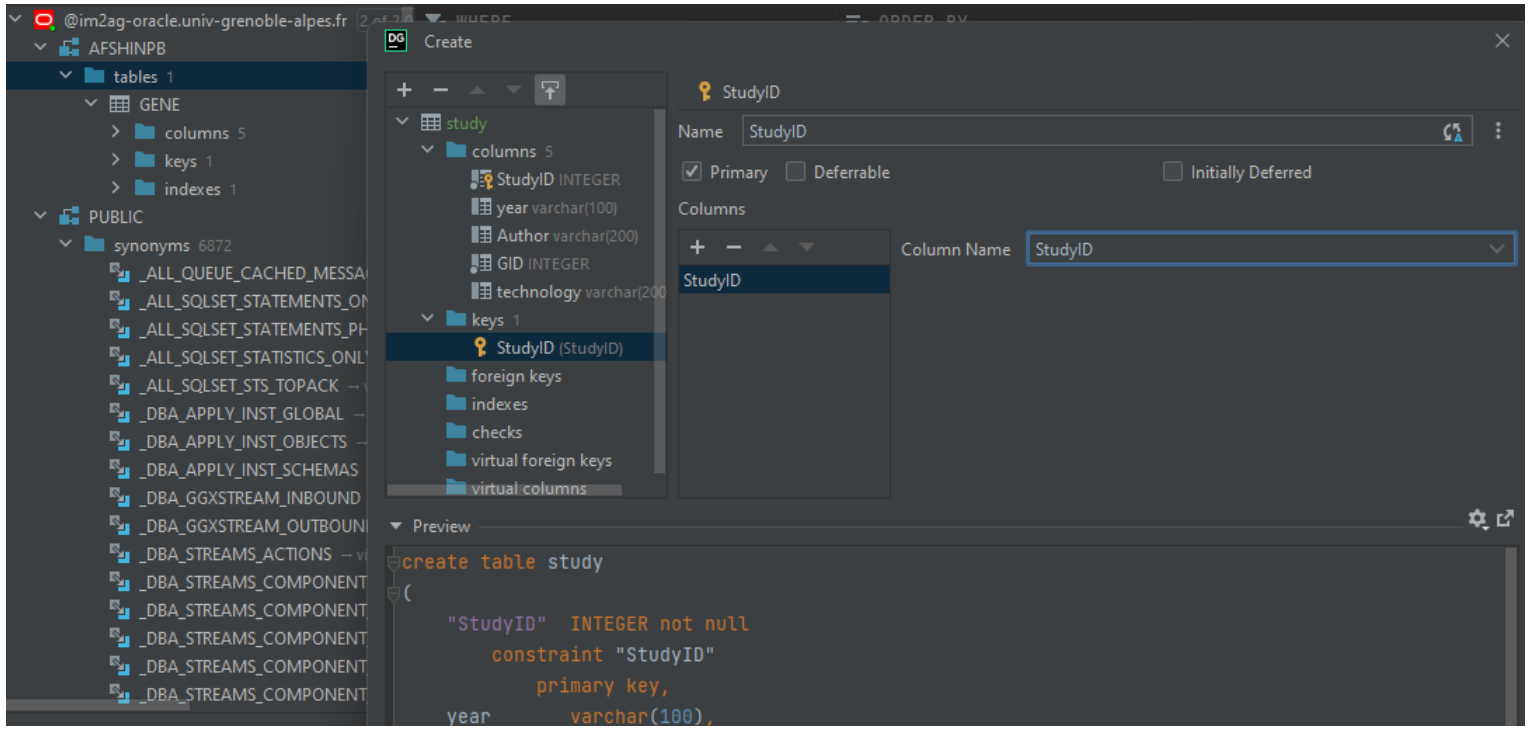
```
CREATE TABLE Persons (
    ID int NOT NULL PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

A table consists of multiple columns.

It consists of perhaps an ID column, perhaps some kind of a name column, maybe some type of a description column.

CREATE TABLE

```
Create table Gene(GId integer NOT NULL
PRIMARY KEY ,name varchar(100) ,
symbol varchar(100),s integer,
chromosome varchar(150) );
```



Anatomy of SQL statement

- SQL statement
  - INSERT (add a new row in a table)
  - UPDATE(modify data in a table)
  - DELETE(Remove a row from a table)

These are part of the DML statement.
DML is data manipulation language.

  - Select (you might want to select data or retrieve data from an existing table.)
  - FROM clause : It combined with the SELECT clause(statement), would allow you to specify which table or tables you want to retrieve data from.
  - WHERE : WHERE clause lets you filter the data that you want to return.

Insert statement

- You specify insert into,
  - the name of the table.
  - the columns that you want to insert the data into.
  - the keyword values,
  - the data that you want to insert.

The thing is, you have to be careful about how you structure this.
The format, the syntax, the parentheses, the commas, the quotes, the semicolons, everything matters.

```
INSERT INTO table_name (column1, column2, column3, ...)VALUES (value1, value2, value3, ...);
```

```
INSERT INTO Customers (Id, CustomerName, Address, City, PostalCode, Country)
VALUES (50,'Cardinal', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

The table name is not case-sensitive.

Without the semicolon, the insert statement would not be processed because Oracle SQL would not be able to understand where the INSERT statement ends.

Insert statement

```
INSERT INTO Gene(GId,name,symbol,s, chromosome)
values(11998,'tumor protein p53','P53',25.760 ,'chr17');

insert into Gene(GId,name,symbol,s, chromosome)
values(20856,'ELN Antisense RNA 1','ELN-AS1', 23.98,'chr7');
```

WHERE ORDER BY

	GID	NAME	SYMBOL	S	CHROMOSOME
1	11998	tumor protein p53	P53	26	chr17
2	7809	nerve growth factor …	NGFR	20	chr17
3	1205	ADCY5	ADCY5	30	chr11
4	20856	ELN Antisense RNA 1	ELN-AS1	24	chr7
5	77899	nerve growth factor …	NGFR	20	chr17

To GUI or Not To GUI?

It is better to learn something new from scratch, without much help from integrated development environments (IDEs), in my experience because that is the quickest method to understand how a certain platform operates.

Update statement

```
Update products set productQTY=5 where productID=2
```

- The update statement followed by
  - the table in this case, products followed by
  - the keywords set.
  - the column that you want to change
  - the value that you want to change it to
  - an optional but very important where clause that filters what particular row you want to impact as a result of this change.

you really don't want to have an update statement without a where clause

SQL – Structured Query Language

- Perhaps the simplest form of query in SQL asks for those tuples of some one relation that satisfy a condition.
- This simple query, like almost all SQL queries, uses the three keywords. SELECT, FROM, and WHERE that characterize SQL.

```
SELECT *
FROM Movies
WHERE studioName = 'Disney' AND year = 1990;
```

A Trick for Reading and Writing Queries

It is generally easist to examine a select-from-where query by first looking at the FROM clause, to learn which relations are involved in the query. Then, move to the WHERE clause, to learn what it is about tuples that is important to the query. Finally, look at the SELECT clause to see what the output is. The same order — from, then where, then select — is often useful when writing queries of your own, as well.

Projection in SQL

Example 6.2: Suppose we wish to modify the query of Example 6.1 to produce only the movie title and length. We may write

```
SELECT title, length
FROM Movies
WHERE studioName = 'Disney' AND year = 1990;
```

The result is a table with two columns, headed `title` and `length`. The tuples in this table are pairs, each consisting of a movie title and its length, such that the movie was produced by Disney in 1990. For instance, the relation schema and one of its tuples looks like:

title	*length*
Pretty Woman	119
...	...

[1] Thus, the keyword **SELECT** in SQL actually corresponds most closely to the projection operator of relational algebra, while the selection operator of the algebra corresponds to the **WHERE** clause of SQL queries.

Projection in SQL

- Sometimes, we wish to produce a relation with column headers different from the attributes of the relation mentioned in the From clause.
- We may follow the name of the attribute by the keyword AS and an alias, which becomes the header in the result relation.

Example 6.3: We can modify Example 6.2 to produce a relation with attributes `name` and `duration` in place of `title` and `length` as follows.

```
SELECT title AS name, length AS duration
FROM Movies
WHERE studioName = 'Disney' AND year = 1990;
```

The result is the same set of tuples as in Example 6.2, but with the columns headed by attributes `name` and `duration`. For example,

name	*duration*
Pretty Woman	119
...	...

Projection in SQL

- We can use an expression in place of an attribute.

Example 6.4: Suppose we want output as in Example 6.3, but with the length in hours. We might replace the SELECT clause of that example with

```
SELECT title AS name, length*0.016667 AS lengthInHours
```

Then the same movies would be produced, but lengths would be calculated in hours and the second column would be headed by attribute `lengthInHours`, as:

name	*lengthInHours*
Pretty Woman	1.98334
...	...

- Lengths would be calculated in hours
- Then rename

Case Insensitivity

Case Insensitivity

SQL is *case insensitive*, meaning that it treats upper- and lower-case letters as the same letter. For example, although we have chosen to write keywords like `FROM` in capitals, it is equally proper to write this keyword as `From` or `from`, or even `FrOm`. Names of attributes, relations, aliases, and so on are similarly case insensitive. Only inside quotes does SQL make a distinction between upper- and lower-case letters. Thus, `'FROM'` and `'from'` are different character strings. Of course, neither is the keyword `FROM`.

Selection

- WHERE clause <attribute><operator><value>
- We may build expressions by comparing values using the six common comparison operators: =, <> , > , <, <= , >=.

Not equal

```
vol.depart = "Londres"
avion.cap < '300'
avion.type = 'AIRBUS 300'
```

Selection

SQL Queries and Relational Algebra

The simple SQL queries that we have seen so far all have the form:

```
SELECT L
FROM R
WHERE C
```

in which L is a list of expressions, R is a relation, and C is a condition. The meaning of any such expression is the same as that of the relational-algebra expression

$$\pi_L(\sigma_C(R))$$

That is, we start with the relation in the `FROM` clause, apply to each tuple whatever condition is indicated in the `WHERE` clause, and then project onto the list of attributes and/or expressions in the `SELECT` clause.

Selection Example

```
Select pilote.nom
From pilote
Where pilote.prenom = 'Antoine';
```

```
Select pilote.nom
From pilote
Where pilote.prenom = 'Antoine';
```

```
Select pilote.nom
From pilote
Where pilote.prenom = 'Antoine';
```

PILOTE

numpilote	nom	prenom
P0001	Dupuis	Antoine
P0002	Simon	Georges
P0003	François	Luc
P0004	André	Georges
P0005	Arthur	Louis
P0006	Mathieu	François

numpilote	nom	prenom
P0001	Dupuis	Antoine
P0002	Simon	Georges
P0003	François	Luc
P0004	André	Georges
P0005	Arthur	Louis
P0006	Mathieu	François

numpilote	nom	prenom
	Dupuis	

SELECT Statement

Used for queries on single or multiple tables

Clauses of the SELECT statement:

- **SELECT**
  - List the columns (and expressions) to be returned from the query
- **FROM**
  - Indicate the table(s) or view(s) from which data will be obtained
- **WHERE**
  - Indicate the conditions under which a row will be included in the result
- **GROUP BY**
  - Indicate categorization of results
- **HAVING**
  - Indicate the conditions under which a category (group) will be included
- **ORDER BY**
  - Sorts the result according to specified criteria

Multirelation Queries

-Interesting queries often combine data from more than one relation.

-We can address several relations in one query by listing them all in the FROM clause.

-Distinguish attributes of the same name by

“<relation>.<attribute>”

Example

- Using relations Likes(drinker, beer) and Frequents(drinker, bar), find the beers liked by at least one person who frequents Joe's Bar.

```
SELECT beer
FROM Likes, Frequents
WHERE bar = 'Joes Bar' AND
    Frequents.drinker =
  Likes.drinker;
```

Subqueries That Return One Tuple

-If a subquery is guaranteed to produce one tuple, then the subquery can be used as a value.

- From Sells(bar, beer, price), find the bars that serve Miller for the same price Joe charges for Bud.

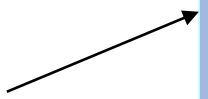
Two queries would surely work:

1. Find the price Joe charges for Bud.
2. Find the bars that serve Miller at that price.

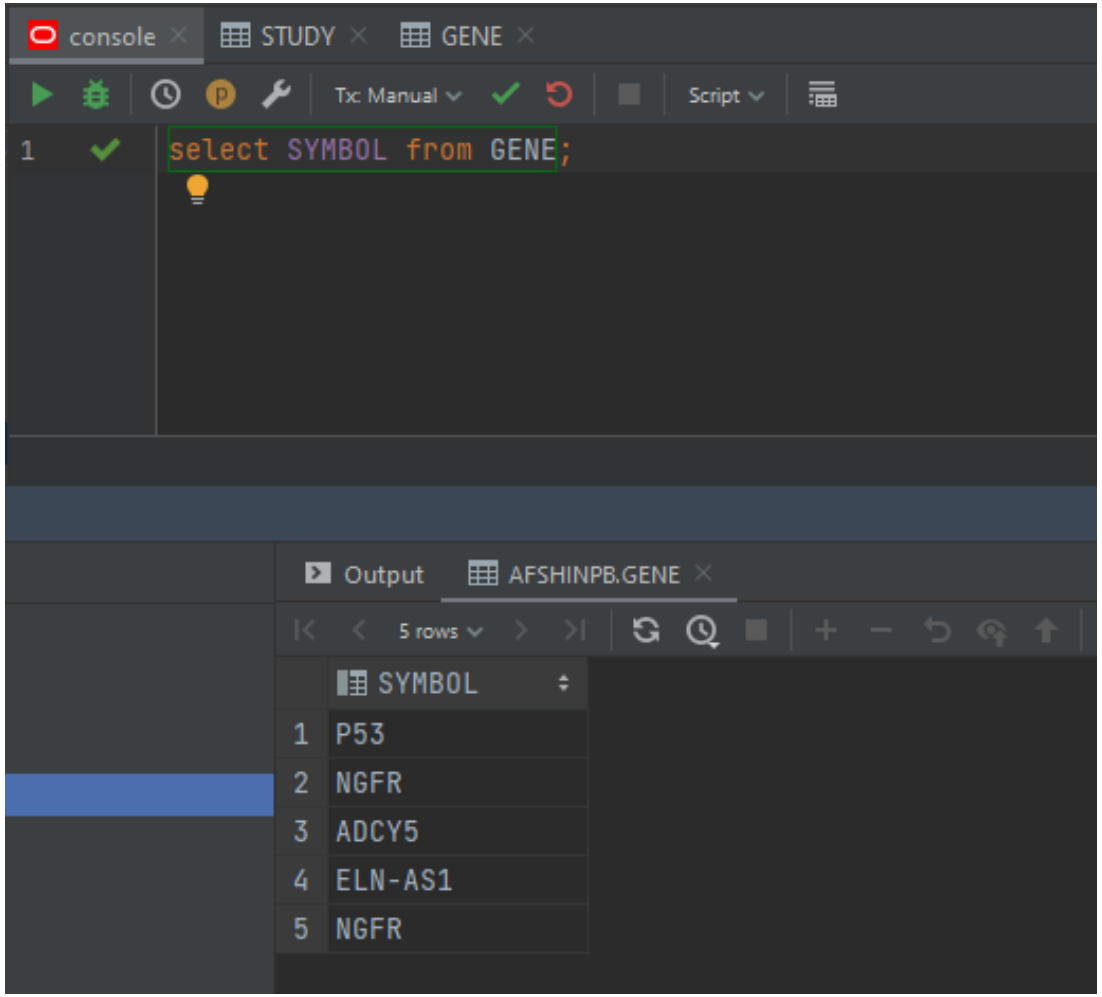
Query + Subquery Solution

```
SELECT bar
FROM Sells
WHERE beer = 'Miller' AND
      price = (SELECT price
               FROM Sells
               WHERE bar = 'Joe''s Bar'
                     AND beer = 'Bud');
```

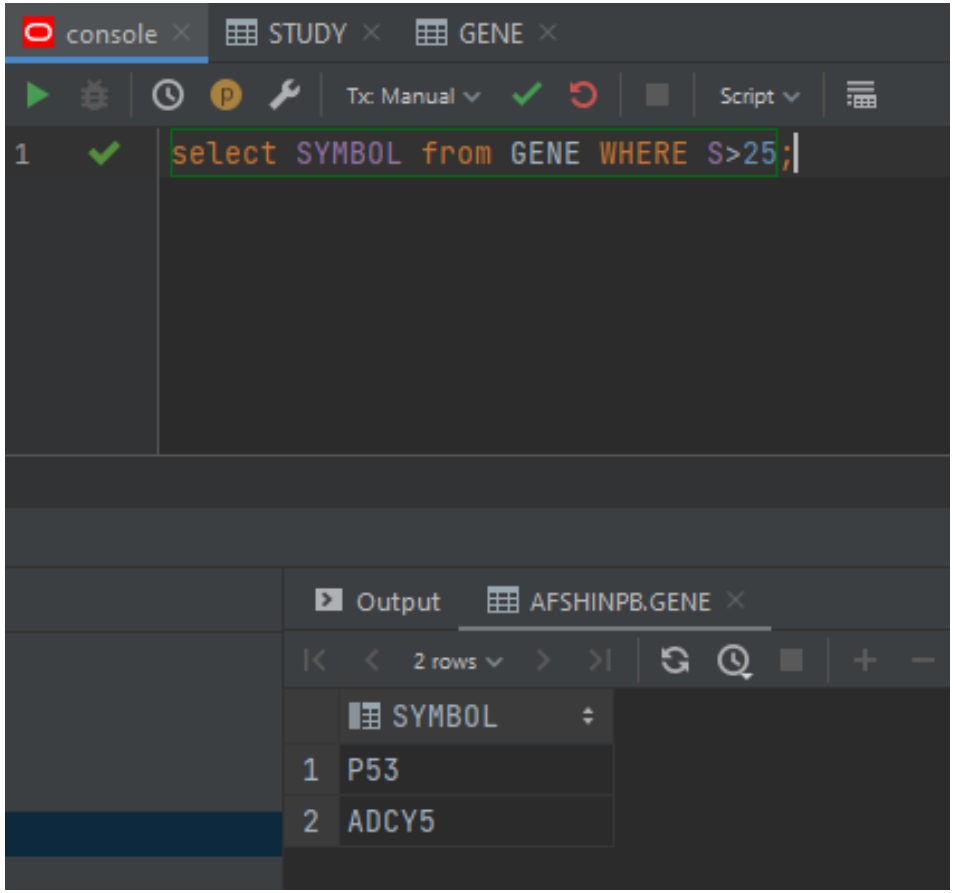
The price at which Joe sells Bud



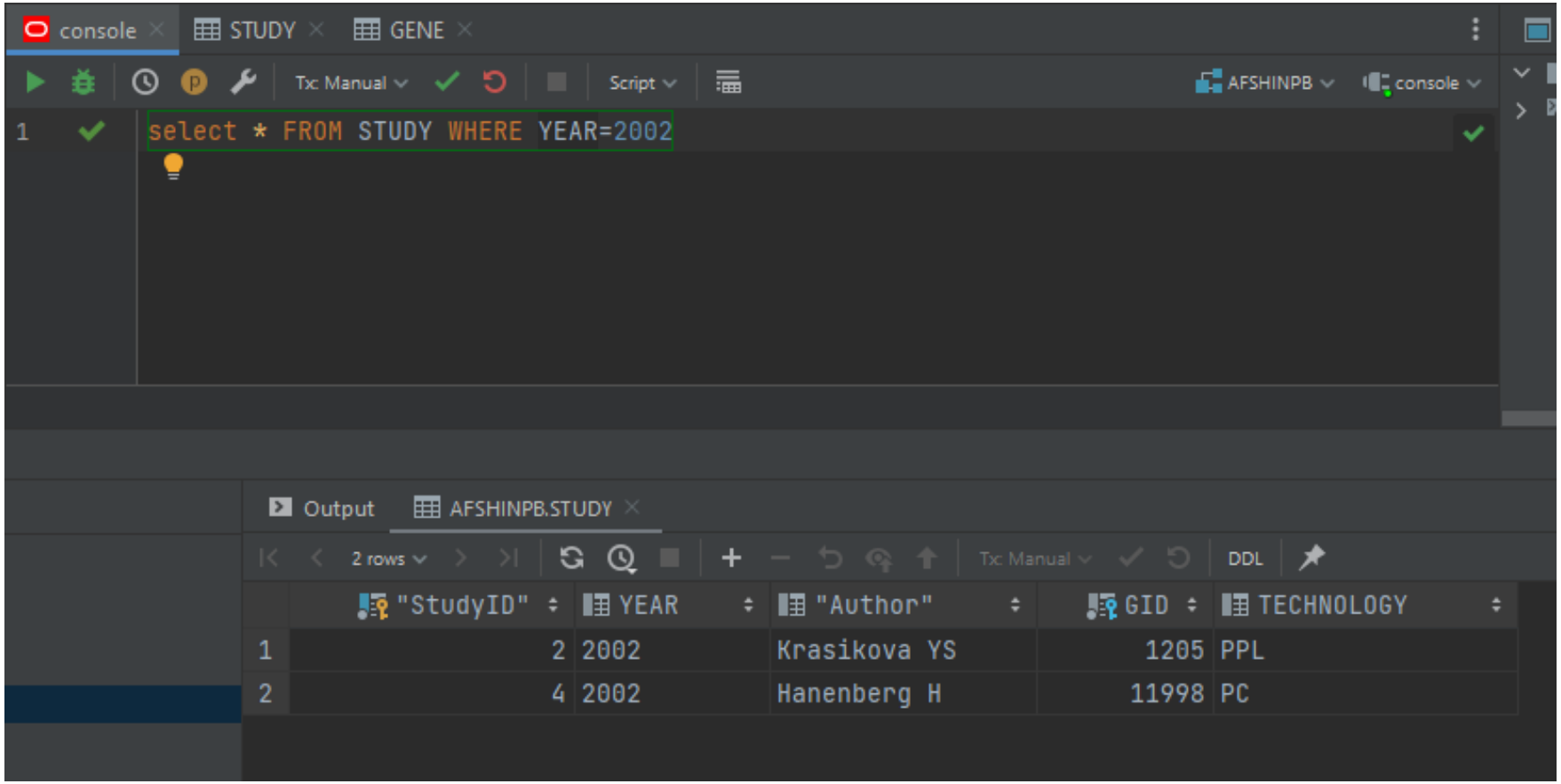
Give the list of Gene's symbols.



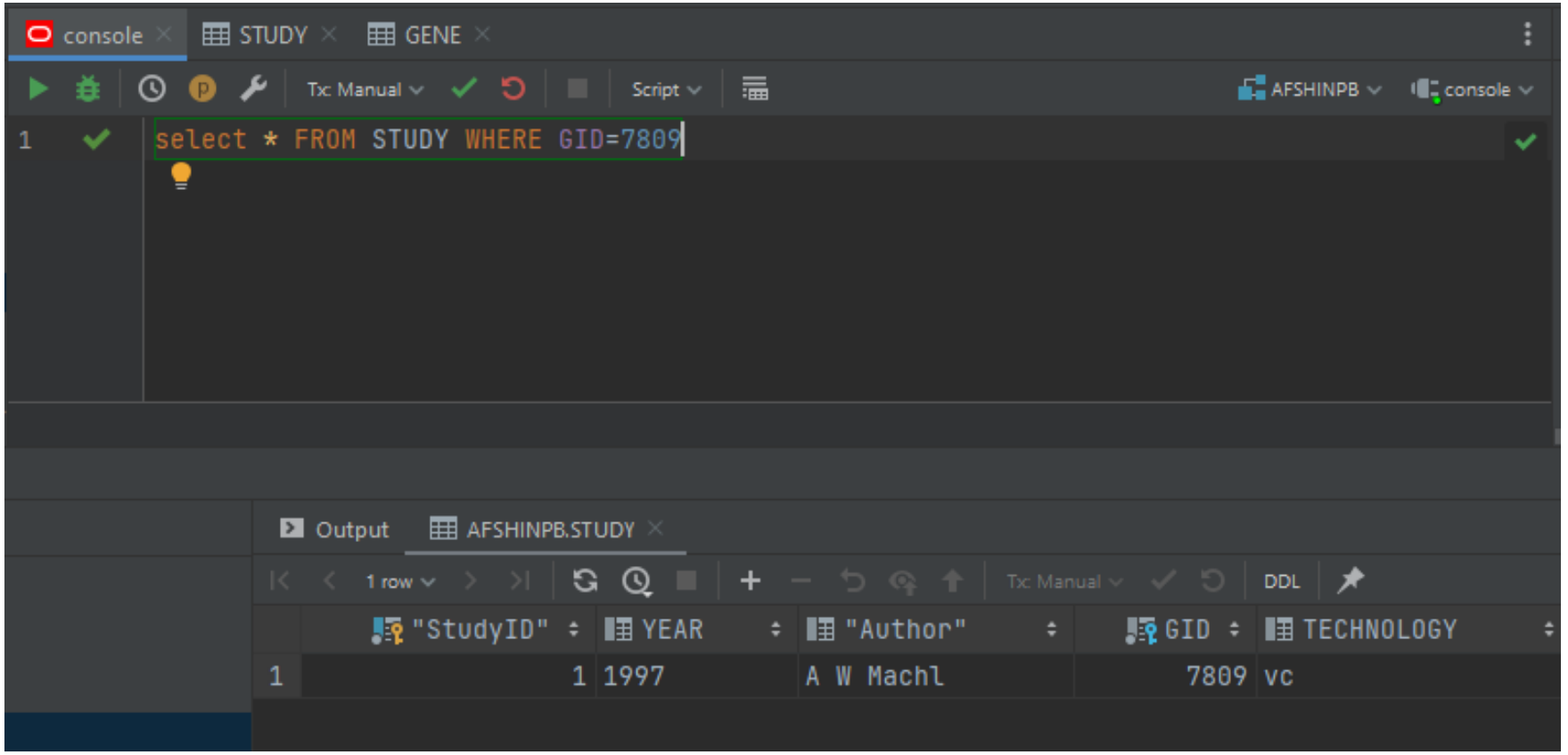
Output the list of genes whose sizes are larger than 25 bases.



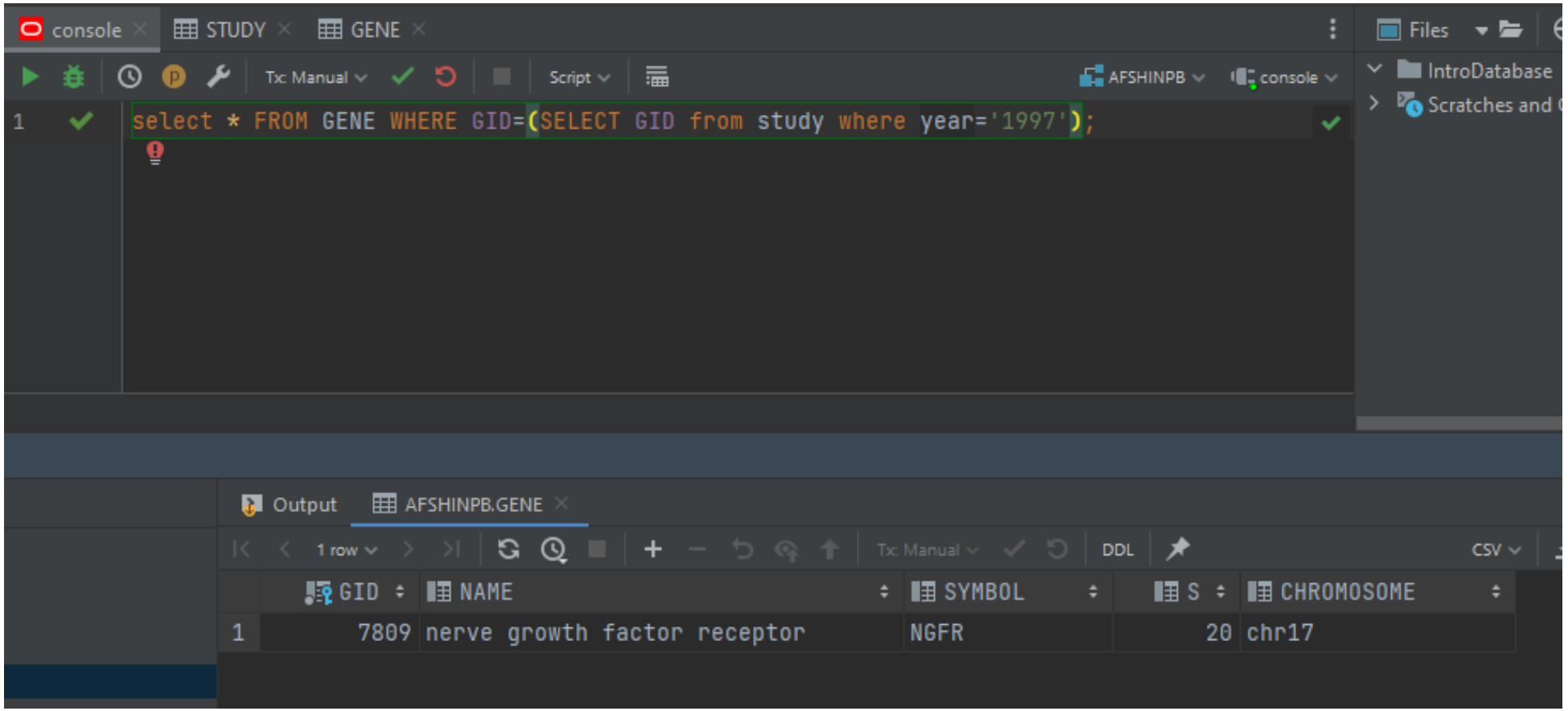
Return the list of authors who studied genes in 2002.



Return the list of authors WHO studied on Gene ID 7809



Output the list of genes that we have information about it in 1997

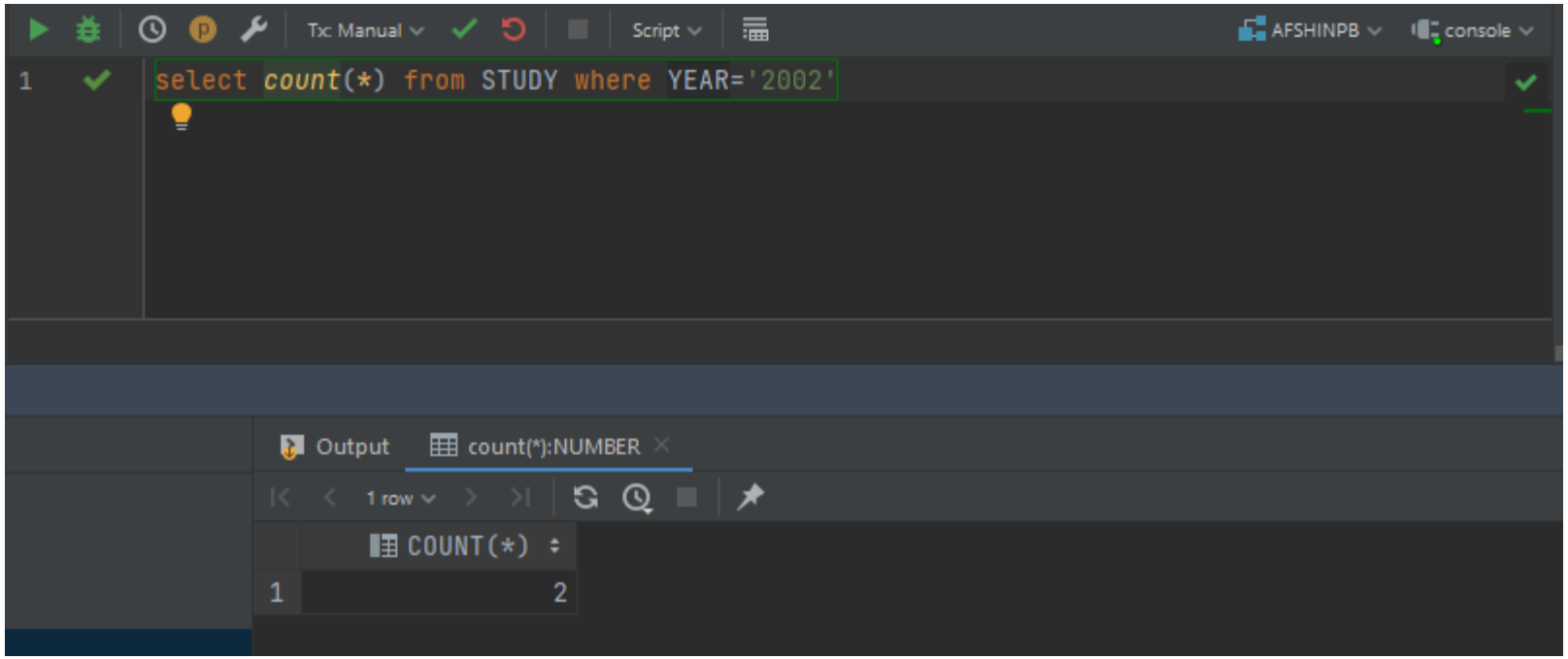


Aggregations

- SUM, AVG, COUNT, MIN, and MAX can be applied to a column in a SELECT clause to produce that aggregation on the column.

- Also, COUNT(*) counts the number of tuples.

How many papers (studies) do we have in the 2002?



Eliminating Duplicates in an Aggregation

- Use **DISTINCT** inside an aggregation.

- Example: find the number of the different chromosomes that we have genes in GENE table: