

Data Models and Data Bases Assignment

Vitrai Gábor – ABIOWE

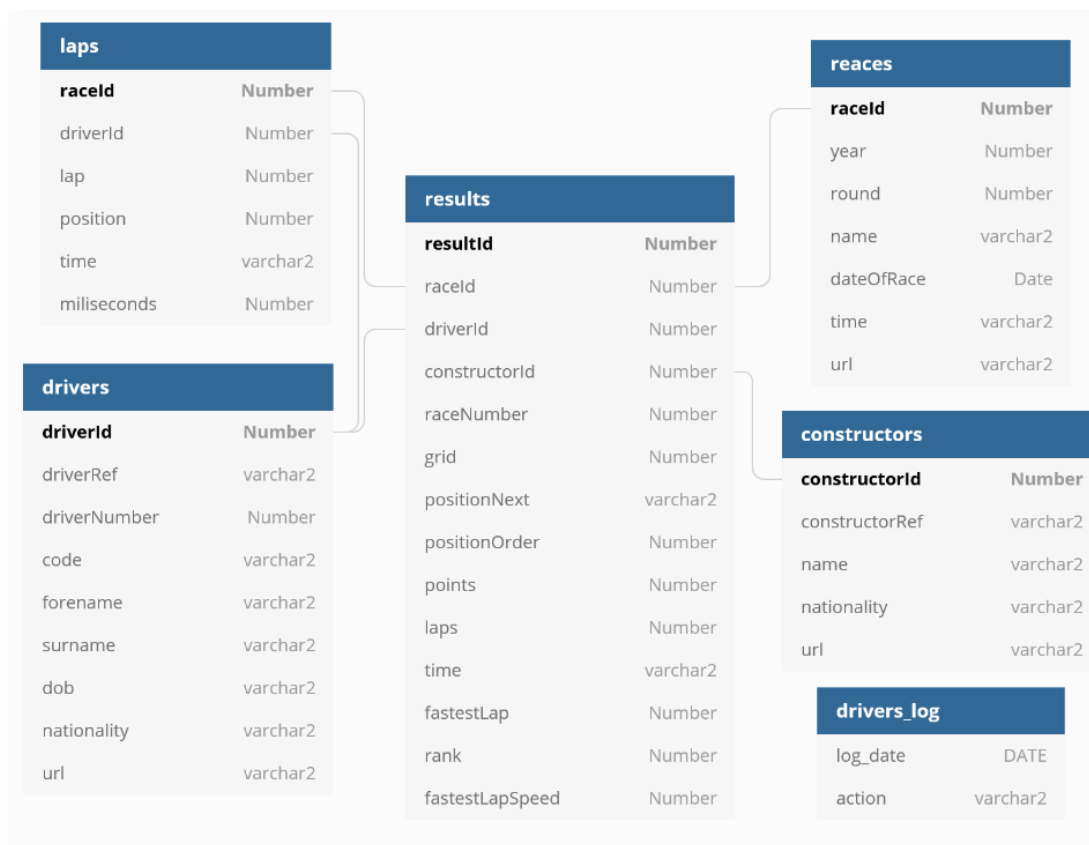
Getting a data set

Starting the project my initial idea was to get a dataset from one of the high school graduation exams sheets, but unfortunately, those data sets are quite short. I started looking on Kaggle, and I found a F1 dataset. I wanted some meaningful information, so I was very happy about the topic. From the dataset I extracted 5 tables with good conditions, so every table is connected to each other in some way. (I also added a 6th table for data logging of the trigger's functionality)

The six table I have:

- Drivers
- Races
- Lap Times
- Results
- Constructors
- (Drivers log)

I selected this dataset because one of the tables has over 400.000 records, and another one has around 50.000, while the others have around 1000 records. I believe these numbers fulfil the conditions of the assignment. I removed some unnecessary features for simplicity. As a next step I created the ER diagram of my tables and marked the links between them.



Data models/schemas:

Drivers table:

This table contains the data of the F1 drivers from 1950 until 2017. Each driver has a unique identifier, some of them has a reference which is how the audience usually calls them. Every driver can have a number they choose, a code which is the first 3 letters of their name. The other attributes are simply forename, surname, day of birth, nationality, and an URL link for their Wikipedia article.

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID
1	DRIVERID	NUMBER(38,0)	Yes	(null)	1
2	DRIVERREF	VARCHAR2(26 BYTE)	Yes	(null)	2
3	DRIVERNUMBER	NUMBER(38,0)	Yes	(null)	3
4	CODE	VARCHAR2(26 BYTE)	Yes	(null)	4
5	FORENAME	VARCHAR2(26 BYTE)	Yes	(null)	5
6	SURNAME	VARCHAR2(26 BYTE)	Yes	(null)	6
7	DOB	DATE	Yes	(null)	7
8	NATIONALITY	VARCHAR2(26 BYTE)	Yes	(null)	8
9	URL	VARCHAR2(128 BYTE)	Yes	(null)	9

	COLUMN_NAME	DATA_TYPE
1	RACEID	NUMBER(38,0)
2	YEAR	NUMBER(4,0)
3	ROUND	NUMBER(38,0)
4	NAME	VARCHAR2(50 BYTE)
5	DATEOFRACE	DATE
6	TIME	VARCHAR2(26 BYTE)
7	URL	VARCHAR2(128 BYTE)

Races table:

This schema consists of the details of the different races. Every race is distinguished by an ID. The table lists the year of the race, the number of rounds, the name of the specific race as most important data. Further attributes are the date of the race, the time of the official beginning, and the URL to the Wikipedia article.

Laps table:

Laps belong to the races; the tables are connected by the raceId and driverId attributes. Otherwise, every record contains the lap time and position of the given driver.

	COLUMN_NAME	DATA_TYPE	NULLABLE
1	RACEID	NUMBER(38,0)	Yes
2	DRIVERID	NUMBER(38,0)	Yes
3	LAP	NUMBER(38,0)	Yes
4	POSITION	NUMBER(38,0)	Yes
5	TIME	VARCHAR2(26 BYTE)	Yes
6	MILLISECONDS	NUMBER(38,0)	Yes

	COLUMN_NAME	DATA_TYPE
1	CONSTRUCTORID	NUMBER(38,0)
2	CONSTRUCTORREF	VARCHAR2(26 BYTE)
3	NAME	VARCHAR2(26 BYTE)
4	NATIONALITY	VARCHAR2(26 BYTE)
5	URL	VARCHAR2(128 BYTE)

Constructors table:

Another interesting list is the constructor table which next to the IDs, contains the reference / shorter name of the constructor (Example: BMW Sauber = bmw_sauber) with the proper name, nationality, and another URL with the Wikipedia link.

Results table:

The final and most important table is the results table. This contains 4 id-s for each of the previously mentioned tables, these are unique IDs. (ResultsId, raceId, driverId, constructorId)

Aside these information, the tables contains information about the Formula 1 race results: Number of race (RACENUMBER), Start grid of each race for each driver (GRID), Final position of each driver/each race (POSITIONTEXT), Final position of each driver/each race filtered by disqualifications (POSITIONTEXT), points gained (POINTS), number of laps (LAPS), final time of the player (TIME), the fastest lap of the driver (FASTESTLAP), the global rank of the race driver for each race (RANK), the time of the fastest lap (FASTESTLAPSPEED).

	COLUMN_NAME	DATA_TYPE
1	RESULTID	NUMBER(38,0)
2	RACEID	NUMBER(38,0)
3	DRIVERID	NUMBER(38,0)
4	CONSTRUCTORID	NUMBER(38,0)
5	RACENUMBER	NUMBER(38,0)
6	GRID	NUMBER(38,0)
7	POSITIONTEXT	VARCHAR2(26 BYTE)
8	POSITIONORDER	NUMBER(38,0)
9	POINTS	NUMBER(38,0)
10	LAPS	NUMBER(38,0)
11	TIME	VARCHAR2(26 BYTE)
12	FASTESTLAP	NUMBER(38,0)
13	RANK	NUMBER(38,0)
14	FASTESTLAPSPEED	NUMBER(38,3)

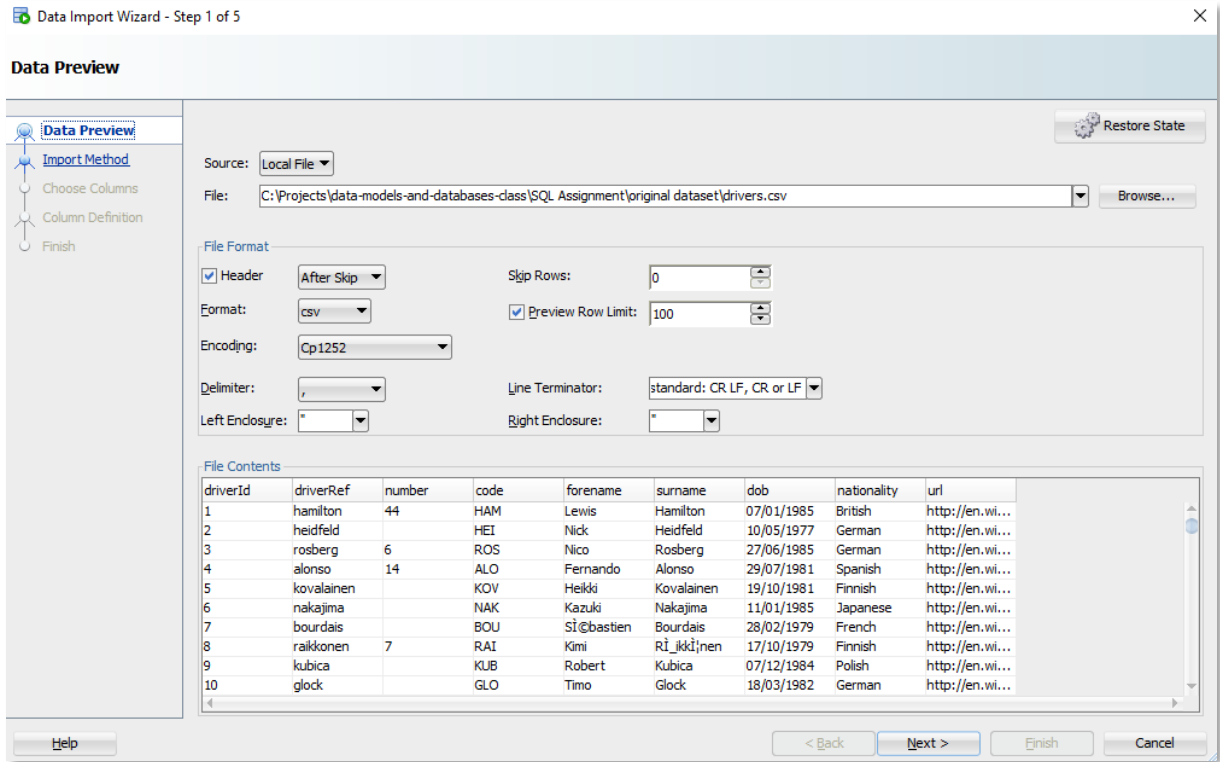
Drivers log:

	COLUMN_NAME	DATA_TYPE
1	LOG_DATE	DATE
2	ACTION	VARCHAR2(50 BYTE)

This is a simple table, containing the logs of the trigger detailed later in this document. This table only consists of 2 attributes. LOG_DATE, and ACTION. The first one is the system date upon triggering the trigger, while the second one is the type of modification on the **drivers table**. In out case, this can be either Insert or Update command.

Loading the data:

To load the data, I used the built-in tool of the Oracle Developer tool. After joining to the database server, I used the "Import Data..." tool. This allowed me to import my data with huge flexibility.

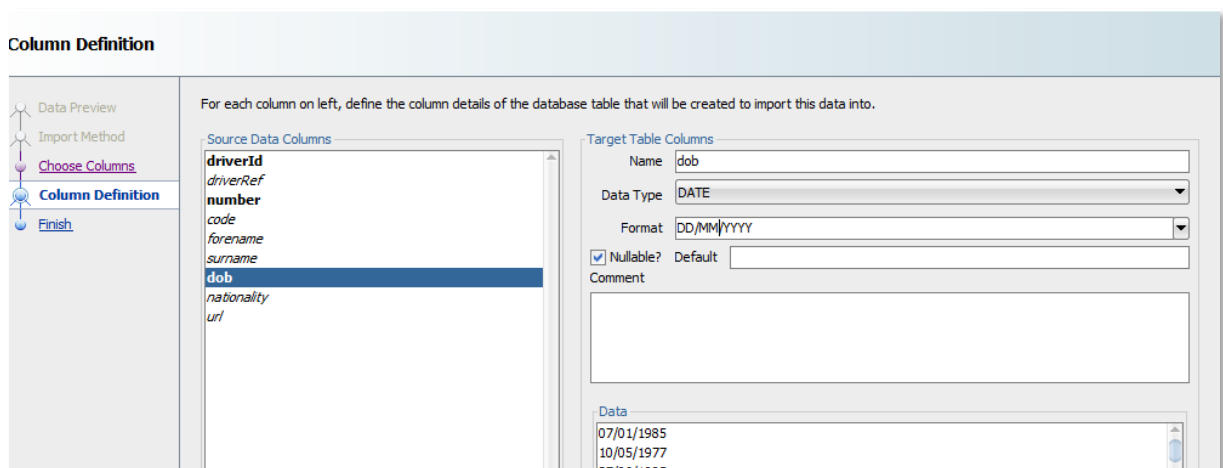


The screenshot shows the 'Data Import Wizard - Step 1 of 5' window. The 'Data Preview' tab is active, displaying a preview of the data from the file 'C:\Projects\data-models-and-databases-class\SQL Assignment\original dataset\drivers.csv'. The 'File Format' section shows 'Header' checked, 'Format' set to 'csv', 'Encoding' set to 'Cp1252', 'Delimiter' set to ',', and 'Line Terminator' set to 'standard: CR LF, CR or LF'. The 'File Contents' section shows a table with 10 rows of driver data.

driverId	driverRef	number	code	forename	surname	dob	nationality	url
1	hamilton	44	HAM	Lewis	Hamilton	07/01/1985	British	http://en.wi...
2	heidfeld		HEI	Nick	Heidfeld	10/05/1977	German	http://en.wi...
3	rosberg	6	ROS	Nico	Rosberg	27/06/1985	German	http://en.wi...
4	alonso	14	ALO	Fernando	Alonso	29/07/1981	Spanish	http://en.wi...
5	kovalainen		KOV	Heikki	Kovalainen	19/10/1981	Finnish	http://en.wi...
6	nakajima		NAK	Kazuki	Nakajima	11/01/1985	Japanese	http://en.wi...
7	bourdais		BOU	Stéphen	Bourdais	28/02/1979	French	http://en.wi...
8	raikkonen	7	RAI	Kimi	Räikkönen	17/10/1979	Finnish	http://en.wi...
9	kubica		KUB	Robert	Kubica	07/12/1984	Polish	http://en.wi...
10	glock		GLO	Timo	Glock	18/03/1982	German	http://en.wi...

As I already had the input tables in a **csv** format, it was easy to just select the **delimiter**. I carefully selected the datatypes for each column and imported the data. I only had to manually modify some preferences in the case of non-allowed attribute names, and in case of date formats.

Manually setting the correct formats.



The screenshot shows the 'Column Definition' step of the Data Import Wizard. It displays the 'Source Data Columns' and the 'Target Table Columns' configuration. The 'dob' column is selected, and its details are shown in the 'Target Table Columns' section.

Source Data Columns	Target Table Columns
driverId	Name: dob
driverRef	Data Type: DATE
number	Format: DD/MM/YYYY
code	<input checked="" type="checkbox"/> Nullable? Default:
forename	Comment:
surname	
dob	Data: 07/01/1985, 10/05/1977, 27/06/1985
nationality	
url	

Views:

- **View 1:** Join at least three tables and add in a where clause!

The first **view** is quite simple. It joins three tables and lists the **fastest lap times** of **every constructor** (the drivers “**employed**” by each constructor) for every **constructor**, in the year of **2010**. Its name is FASTEST_LAPS_OF_CONSTRUCTORS_WITH_PILOTS.

The code looks the following:

```
CREATE OR REPLACE VIEW FASTEST_LAPS_OF_CONSTRUCTORS_WITH_PILOTS AS
SELECT races.year, races.name as RaceName, constructors.name, results.fastestLapSpeed
FROM constructors INNER JOIN
    (drivers INNER JOIN
        (races INNER JOIN results ON races.raceId = results.raceId)
        ON drivers.driverId = results.driverId)
ON constructors.constructorId = results.constructorId
WHERE ((races.year) = 2010);
```

The query works the following way: It **selects** the name, year of the races available, in addition with the constructor’s name, and the **fastest lap speeds**. The given tables are joined with inner joins, and the Ids of the given entities are linked together. As a final step the year is **queried** with a **where clause**.

Extract from the output:

2010	British Grand Prix	Sauber	221.182
2010	British Grand Prix	Renault	218.982
2010	British Grand Prix	Virgin	210.725
2010	German Grand Prix	Ferrari	217.005

- **View 2:** Join at least two tables and a subquery and use somewhere a grouping with having!

This is the most **complex query** in this assignment. It **joins two tables and a subquery** (*drivers, races, driverId from subquery*) The subquery returns the driverIds where the driver finished on the on the podium. (1st, 2nd or 3rd place).

These data tables will be **linked together with inner joins** on the given ids. At the end the result is summarized with a group by on the given attributes, and an additional layer of filtering is added with a having. This will filter the Hungarian Grand Prix only.

Source code:

```
CREATE OR REPLACE VIEW POL_POSITION_DRIVERS_ON_HUNGARORING AS
SELECT drivers.forename, drivers.surname, drivers.nationality, ID, Position
FROM drivers INNER JOIN
    (races INNER JOIN
        (SELECT driverId as ID, POSITIONTEXT as Position
         FROM results WHERE POSITIONTEXT = '1' OR POSITIONTEXT = '2' OR POSITIONTEXT = '3')
        ON races.raceId = ID)
ON drivers.driverId = ID
GROUP BY drivers.forename, drivers.surname, races.name, drivers.nationality, ID, Position
HAVING ((races.name) = 'Hungarian Grand Prix');
```

Extract from the output:

FORENAME	SURNAME	NATIONALITY	ID	POSITION
Ayrton	Senna	Brazilian	102	1
Jacky	Ickx	Belgian	235	2
Jacky	Ickx	Belgian	235	1
Johnny	Herbert	British	65	1
Ayrton	Senna	Brazilian	102	3
Johnny	Herbert	British	65	2
Timo	Glock	German	10	3

- **View 3:** Query using a set operator!

This is a simple query, featuring **2 select queries**, with a **UNION** set operator. The view returns the id's of the drivers who is either Italian (nationality) or has an Italian car (constructor's nationality).

Source code:

```
CREATE OR REPLACE VIEW ITALIAN_DRIVERS AS
(SELECT DISTINCT results.driverid FROM drivers
 INNER JOIN results ON results.driverId = drivers.driverId
 WHERE drivers.nationality = 'Italian')
UNION
(SELECT DISTINCT results.driverid FROM constructors
 INNER JOIN results ON results.constructorId = constructors.constructorId
 WHERE constructors.nationality = 'Italian');
```

The first query joins the results table with the drivers table and filters for nationality, while the second one makes an inner join with the constructors and results tables. After this, a where clause is added for the constructor's nationality.

The result will look like this:

DRIVERID
1
2
3
4
5
6

PLSQL Function and Procedure:

- 1 stored **PLSQL function** which queries one of your view, using parameter(s) and cursor!

The HUNGARORING_FILTER gets the drivers with a **specific nationality** from the second view. This function works as a filter which queries the “POL_POSITION_DRIVERS_ON_HUNGARORING” View, which returns the podium drivers with their positions on the Hungarian Grand Prix.

The function looks the following:

```
CREATE OR REPLACE FUNCTION hungaroring_filter(nationality_param VARCHAR2)
RETURN NUMBER
is
    CURSOR curs1 IS SELECT forename, surname, nationality, id, position
    FROM POL_POSITION_DRIVERS_ON_HUNGARORING WHERE nationality = nationality_param;
    rec curs1%ROWTYPE;
    counter int;
BEGIN
    OPEN curs1;
    LOOP
        FETCH curs1 INTO rec;
        EXIT WHEN curs1%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(rec.forename || ', ' || rec.surname || ', -' ||
        rec.nationality || '-, Position: ' || rec.position);
    END LOOP;
    counter := curs1%ROWCOUNT;
    CLOSE curs1;
    RETURN counter;
END;
```

The function creates a **cursor** with the forename, surname, nationality, id, position of the given drivers and it selects them based on the given parameter nationality. Upon calling the function, it will iterate through the table with the help of a **loop**, and with the cursor (**fetch**), it will list out the given drivers who ever finished on the podium at the Hungarian Grand Prix.

The result will look like this:

ullman_tunnel ✕			
Timo, Glock, -German-,	Position: 3		
Timo, Glock, -German-,	Position: 2		
Jacky, Ickx, -Belgian-,	Position: 2		
Jacky, Ickx, -Belgian-,	Position: 1		
Jacky, Ickx, -Belgian-,	Position: 3		

- 1 stored **PLSQL procedure** which modifies your database, using parameter(s) and cursor!

For the **procedure** (generate_code), I decided to do something actual **useful**. I noticed that many drivers **do not have** the DRIVERNUMBER attribute assigned, so I made a procedure which will generate this code for every driver. I also realized that this is a good procedure **to be used in my trigger**.

The procedure gets the surname of every driver and generates the **capitalized substring of the first 3 letters** of the driver's name. This "code" then added in the database. This way drivers without a code gets a code, and drivers who (Let's say) changes his/her name, also gets the code **updated**.

The procedure looks the following:

```
create or replace procedure generate_code is
  cursor curs1 is
    select * from drivers
      where drivers.code is null
      or
      drivers.code != UPPER(SUBSTR(drivers.surname,1,3))
  for update;
  rec curs1%ROWTYPE;
BEGIN
  for rec in curs1 loop
    update drivers set drivers.code = UPPER(SUBSTR(drivers.surname,1,3))
      where current of curs1;
  end loop;
END;
```

The procedure declares a **cursor**, on the set of drivers where the **code is not given**. Then, as a next step after the declaration of a **record variable** (where the current record gets copied), a **for loop iterates through** the given dataset with the help of the **cursor** and **executes** the necessary **built in functions** and ten **updates** the records. We can see the difference below.

Original data:

64	diniz	(null)	(null)	Pedro	Diniz	22-MAY-70	Brazilian
65	herbert	(null)	(null)	Johnny	Herbert	25-JUN-64	British
66	mcnish	(null)	(null)	Allan	McNish	29-DEC-69	British
67	buemi	(null)	BUE	Sébastien	Buemi	31-OCT-88	Swiss
68	takagi	(null)	(null)	Toranosuke	Takagi	12-FEB-74	Japanese
69	badoer	(null)	BAD	Luca	Badoer	25-JAN-71	Italian

Updated data:

64	diniz	(null)	DIN	Pedro	Diniz	22-MAY-70	Brazilian
65	herbert	(null)	HER	Johnny	Herbert	25-JUN-64	British
66	mcnish	(null)	MCN	Allan	McNish	29-DEC-69	British
67	buemi	(null)	BUE	Sébastien	Buemi	31-OCT-88	Swiss
68	takagi	(null)	TAK	Toranosuke	Takagi	12-FEB-74	Japanese
69	badoer	(null)	BAD	Luca	Badoer	25-JAN-71	Italian

Trigger:

As I mentioned earlier, I decided to use the *generate_code* procedure in my required **trigger** (NEW_DRIVER_TRIGGER). This helps to keep the trigger complex in terms of functionality, but still simple on the code side. I also decided to add a simple **if conditional statement**, only for logging purposes. For inserting, the scripts **log** the “New Driver is begin added” and for updating it **logs** the “A Driver is begin modified” text to the *dbms_output*. Aside from this, the trigger also logs if any event happened to the DRIVERS table. Every UPDATE or INSERT event is logged to the DRIVERS_LOG table.

New User’s (**gabor**) CODE successfully generated with automated **Trigger**.

840	759	flinterman	(null)	FLI	Jan	Flinterman	02-OCT-19	Dutch
841	761	crespo	(null)	CRE	Alberto	Crespo	16-JAN-20	Argentine
842	762	rol	(null)	ROL	Franco	Rol	05-JUN-08	Italian
843	763	gabor	44	GAB	vitrai	gabor	09-JAN-97	Hungarian

Trigger Code:

```
CREATE OR REPLACE TRIGGER NEW_DRIVER_TRIGGER
AFTER INSERT OR UPDATE ON DRIVERS
BEGIN
  IF INSERTING THEN
    DBMS_OUTPUT.PUT_LINE('New Driver is begin added');
    INSERT INTO DRIVERS_LOG (log_date, action) VALUES (SYSDATE, 'INSERT');
    generate_code();
  ELSIF UPDATING THEN
    DBMS_OUTPUT.PUT_LINE('A Driver is begin modified');
    INSERT INTO DRIVERS_LOG (log_date, action) VALUES (SYSDATE, 'UPDATE');
    generate_code();
  END IF;
END;
```

After triggering the trigger, the log table contains the event, and the date of the event. The DBMS output also logged the events for the user. For us to see this in the developer tool, we had to enable the output and open the window of the output.

	LOG_DATE	ACTION
1	10-APR-21	UPDATE
2	10-APR-21	INSERT
3	10-APR-21	UPDATE

Content of the Log Table

Dbms Output	
+ Buffer Size: 20000	
ullman_tunnel x	
A Driver is begin modified	
New Driver is begin added	
A Driver is begin modified	

Logs on the output window

Other Conditions:

- Include at least one **insert command** and at least one **delete or update** command in your PLSQL procedure and trigger!
 - Insert command: Can be found in the trigger.
 - Update command: Inside procedure & trigger (by the procedure)
- Use at least **one loop** and **one if** statement in the function, procedure, and trigger!
 - Loop: function, procedure
 - If statement: Trigger
- **Sequence:** One Example sequence: DRIVERID for increasing the id of the driver automatically after importing the full database.

The code is easy to understand, the beginning value was set to be 762 as this is the exact size of the example dataset, but this can be modified.

```
CREATE SEQUENCE "ABIOWE"."DRIVERID" MINVALUE 1 MAXVALUE 100000 INCREMENT BY 1  
START WITH 762 NOCACHE ORDER NOCYCLE NOKEEP NOSCALE GLOBAL ;
```

Other requirements:

- Grant me (lkpeter) the necessary privileges to use your database objects!
Granted:
 - Select, insert, update, delete, execute permissions added to user: lkpeter
- Demonstrate how your objects work with screenshots!
 - Screenshots added