Laboratory report

Lab 3: SQL2

Field	Value
Name	Imeri Klevis
Neptun ID	T4XGKO
Exercise code	27-HALL-A
Name of instructor	Tatiana Barbova
Time of laboratory	2023-10-19 16:15
Location of laborator	R4P
User name	T4XGKO
Password	HelloDatabases1@?
Solved exercises	2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8

Solution of exercises

---

exercise 2.1

The simplest way to list all the date for a table is to list all the columns using the `*`.

```
SELECT * FROM subjects;
```

> `*`: means every columns in the order of creation

1. `SELECT *`: ***Projection for ALL***
   - This part of the query specifies that you want to select all columns from the table. The asterisk (*) is a wildcard character that represents all columns in the table.
2. `FROM subjects`:
   - This part of the query specifies the source of the data. It tells the database to retrieve the data from the "subjects" table.

exercise 2.2

```
SELECT neptuncode, subjectname, lecturer
FROM subjects
WHERE day1=day2;
```

Here we have `projected` the specified columns (neptuncode, subjectname, lecturer) form the table `subjects` with the condition that the both the lectures are on the same day.

exercise 2.3

```
SELECT
  s.name,
  e.examdate
FROM
  students s,
  enrollments e
WHERE
  s.student_id = e.student
  AND s.dateofbirth < TO_DATE('1990-01-01', 'yyyy-mm-dd')
ORDER BY
  s.name;
```

> We could have also used `DISTINCT` so we don't get the same rows repeated, because students can have exams for different subject in the dame date. Is is not mentioned so we are no applying it.

1. `FROM students s, enrollments e`: ***Aliases of Tables***
   - This part of the query specifies the tables from which the data is being retrieved. It uses aliases "s" and "e" for the "students" and "enrollments" tables, respectively.
2. `WHERE s.student_id = e.student AND s.dateofbirth < TO_DATE('1990-01-01', 'yyyy-mm-dd')`: ***Restriction***
   1. `s.student_id = e.student` : This is the ***inner join***. We need the inner join because we need to retrive the data correlating both the columns. It specifies that rows will be included in the result only if the "student_id" in the "students" table matches the "student" in the "enrollments" table. So if there is no match is not included.
   2. `s.dateofbirth < TO_DATE('1990-01-01', 'yyyy-mm-dd')`: The function TO_DATE just creates a date given the format and the date. Than we check if the date of the students is less than `01/01/1990` and filter by it.
3. `ORDER BY s.name`: This part of the query orders the result set based on the "name" column from the "students" table in ascending order. `ASC` is by default.

exercise 2.4

```sql
SELECT
  s.name,
  e.examdate
FROM
  students s,
  enrollments e
WHERE
  s.student_id = e.student
  AND s.address is NULL;
```

> Many of the lines of the query will be the same as previously therefore no repeating explanation will be given.

The only thing changing is:

1. `AND s.address is NULL`: ***is NULL***
   - this is the condition that the address is not provided or in other words the value of the address is `NULL`.

exercise 2.5

```
SELECT
    s.name
FROM
    students s,
    enrollments e
WHERE
    s.student_id=e.student(+)
GROUP BY
    s.name
HAVING
    COUNT(e.subject)=0
ORDER BY
    name DESC;
```

> `DESC`: the keyword for ordering by descending.

1. `WHERE s.student_id=e.student(+)`: ***outer join***
   - This part of the query specifies the outer join condition between the "students" and "enrollments" tables. It uses an Oracle-specific notation with `(+)` to perform an outer join. This means it will include all students, even if they do not have corresponding records in the "enrollments" table.
2. `GROUP BY s.name`:
   - The query groups the results by the "name" column from the "students" table.
3. `HAVING COUNT(e.subject) = 0`:
   - The `HAVING` clause is used to filter the grouped results. In this specific condition, it checks the count of subjects (e.subject) within each group. The aggregate function `COUNT(e.subject)` counts the number of subjects (rows) associated with each student.
   - Whenever we are filtering by an aggregate functions we need to use the `HAVING`.
   - If the value at the row is `NULL` then the row is not counted.
   - If `COUNT(e.subject)` is equal to 0, it means there are no subjects associated with that group of students (i.e., no enrollments for students with that name).
   - If `COUNT(e.subject)` is not equal to 0, it means there are subjects associated with that group of students.
4. `ORDER BY name DESC`: - Finally, the query orders the result set by the "name" column in descending order, so the names are listed in reverse alphabetical order.

exercise 2.6

```sql
SELECT
    s.subjectname AS "subjectname",
    COUNT(e.student) AS "number"
FROM
    subjects s,
    enrollments e
WHERE
    s.subject_id=e.subject(+)
    AND e.enrollmentdate >= TO_DATE('2023-01-01', 'yyyy-mm-dd')
GROUP BY
    s.subjectname
ORDER BY
    "number" DESC;
```

> you don't need the to put quotation marks for the aliases after `AS` but in our case the work `number` can not be used because is referring to another keyword of SQL. Therefore we need `"number"`. The only problem is that now the displaying will also be in lowercase. The tutor said there is no problem in the wat you display.

1. `SELECT s.subjectname AS "subjectname", COUNT(e.student) AS "number"`: ***Projection and Aliases***
    - This part of the query selects the "subjectname" from the "subjects" table and uses the `COUNT` function to count the number of students (registrations) associated with each subject. It assigns aliases to the selected columns, renaming them as "subjectname" and "number" in the result.
2. `AND e.enrollmentdate >= TO_DATE('2023-01-01', 'yyyy-mm-dd')`: ***Predicate***
    - This condition filters the results to include only rows where the "enrollmentdate" in the "enrollments" table is greater than or equal to January 1, 2023, ensuring that the data is for the current year.
3. `GROUP BY s.subjectname`:
    - The query groups the results by the "subjectname" column from the "subjects" table. This groups the data by subject name, allowing the count of registrations to be aggregated for each subject.
4. `ORDER BY "number" DESC`: - Finally, the query orders the grouped results in descending order based on the "number" column, which represents the count of registrations. This means the most popular subjects, in terms of registrations, will be displayed first in the result set.

exercise 2.7

```
SELECT
    s.neptuncode,
    s.lecturer,
    COUNT(e.student) AS "number"
FROM
    subjects s,
    enrollments e
WHERE
    s.subject_id = e.subject(+)
GROUP BY
    s.neptuncode,
    s.lecturer
HAVING
    COUNT(e.student) <= 2
ORDER BY
    "number" DESC,
    s.neptuncode ASC;
```

1. `SELECT s.neptuncode, s.lecturer, COUNT(e.student) AS "number"`: ***Projection and Aliases***
   - This part of the query selects the "neptuncode" and "lecturer" columns from the "subjects" table (aliased as "s"). It uses the `COUNT` function to count the number of students (registrations) associated with each subject. The result of the `COUNT` function is given an alias `"number"`.
2. `WHERE s.subject_id = e.subject(+)`: ***outer join.***
   - An outer join is needed because you also need to list the subject with 0 registrations.
3. `GROUP BY s.neptuncode, s.lecturer`:
   - The query groups the results by the "neptuncode" and "lecturer" columns from the "subjects" table. This groups the data by subject code and lecturer, allowing the count of registrations to be aggregated for each subject-lecturer combination.
4. `HAVING COUNT(e.student) <= 2`: ***Filtering by Aggregate Functions***
   - The `HAVING` clause is used to filter the grouped results. It includes only those subject-lecturer combinations where the count of students (registrations) is less than or equal to 2. In other words, it filters out subjects with very low enrollment.p
5. `ORDER BY "number" DESC, s.neptuncode ASC`: - The query orders the results first in descending order based on the "number" column (the count of registrations) and then in ascending order based on the "neptuncode." This means that subjects with the fewest registrations will be displayed first, and in case of ties, they will be sorted by neptuncode in ascending order. -

exercise 2.8

A slight explanation before we start this problem. The "the suggested exam date(s)" is very ambiguous in this context. After the consultation with the lab tutor we concluded that we will display "The `EARLIEST` date". This implies we need to display the minimum of the dates for that subject if the subject has a student who is registering for the first time. Here is the code:

```sql
SELECT
    s.neptuncode,
    s.subjectname,
    CASE
        WHEN SUM(e.firstenroll) >= 1 THEN MIN(e.examdate)
        ELSE NULL
    END AS examdate
FROM
    subjects s,
    enrollments e
WHERE
    s.subject_id = e.subject(+)
GROUP BY
    s.neptuncode,
    s.subjectname;
```

> Here we are using a `CASE` statement which allows you to perform different actions or return different values based on specified conditions. A `CASE` statement provides conditional logic to SQL.

1. `CASE WHEN SUM(e.firstenroll) >= 1 THEN MIN(e.examdate) ELSE NULL END AS examdate`: ***Conditional Aggregation***
    - Involves performing aggregation functions like SUM, COUNT, AVG, MIN, or MAX, but the result of the aggregation is dependent on certain conditions.
    - The `CASE` statement calculates the "examdate" for each subject. If the sum of "firstenroll" is greater than or equal to 1 (indicating that at least one student has registered for the subject for the first time), it selects the minimum (earliest) exam date for that subject. Otherwise, it sets "examdate" to NULL.
    - If you want to select the newest date then you can use `MAX` instead of `MIN`.
2. `WHERE s.subject_id = e.subject(+)`: ***Outer join***
    - Outer join is here important because it ensures that all subjects are included in the result, even if there are no matching enrollments. In this context, it's necessary to list all subjects, whether or not they have enrollments.
3. `GROUP BY s.neptuncode, s.subjectname`:
    - You need to group by all the columns when you have a aggregate function.