Web Systems Fundamentals and Databases (Grundläggande webbsystem och databaser) DI4020 11hp

Lab Exercise 3

Course responsible:
Wagner Ourique de Morais

Lab Exercise 3 – DBMS programming

Introduction

A database is an organized and indexed collection of data. A Database Management System (DBMS) is system software that provides tools and capabilities for creating, managing, and administering databases. DBMSs also offer mechanisms for concurrency control, access authorization, and data recovery, enhancing the efficiency and reliability of data storage and retrieval. Due to their maturity and robustness, DBMSs are widely used across various applications[1].

Structured Query Language (SQL)

Relational DBMSs, in particular, utilize a standardized special-purpose declarative language called Structured Query Language (SQL) to define and manipulate data. SQL was developed in the 1970s by Donald Chamberlin and Raymond Boyce at IBM.

SQL commands such as CREATE, ALTER, and DROP are used to define the database structure, including tables, attributes, domains, and relational constraints. This aspect of SQL is known as Data Definition Language (DDL), which establishes how data is organized within the database.

Another key component of SQL is Data Manipulation Language (DML), which enables users to retrieve, insert, update, and delete data using commands like INSERT, SELECT, UPDATE, and DELETE—commonly referred to as CRUD operations. Together, DDL and DML form the foundation of SQL, providing a structured approach to both database management and data operations.

When an SQL statement is submitted to the DBMS, the system parses and validates its syntax and structure. Additionally, the DBMS enforces access control, ensuring that the client executing the SQL command has the necessary privileges. Only after these checks are completed does the query processor execute the SQL statement.

SQL is case-insensitive.

PostgreSQL[2]

PostgreSQL is a cross-platform, free and open-source object relational DBMS that complies with the SQL standard and implements its own SQL dialect, PL/pgSQL (Procedural Language/PostgreSQL). "*PostgreSQL can be used, modified, and distributed by anyone free of charge for any purpose, be it private, commercial, or academic*"[3].

MySQL[4]

MySQL is a cross-platform, free, and open-source relational DBMS that complies with the SQL standard and supports procedural extensions such as Stored Procedures, Triggers, and Events. MySQL supports multiple storage engines, with InnoDB being the default, providing ACID compliance, transactions, and foreign key constraints. Other storage engines, such as MyISAM, offer faster read operations but lack transactional support.

PostgreSQL and Mysql are widely used for web applications, enterprise solutions, and cloud-based services due their high performance, scalability, and reliability. MySQL is commonly used in high-traffic

---

[1] de Morais, W. O. (2015). *Architecting Smart Home Environments for Healthcare: A Database-Centric Approach*. Halmstad University Press.
[2] https://www.postgresql.org
[3] https://www.postgresql.org/docs/current/intro-whatis.html
[4] https://www.mysql.com

web applications like WordPress, Facebook, and Twitter, while PostgreSQL is often preferred for applications requiring complex queries, advanced indexing, and strict data integrity.

Objective

As preparation for Lab Exercise 4, students will practice SQL in Lab Exercise 3 to become familiar with the language and learn how to define and manipulate a small schema.

Lab Exercise 3 includes 25 tasks, and you are expected to write and execute the SQL code to solve each of these tasks.

Preparation

This Lab Exercise is aligned with the content presented and discussed in Lectures 6 to 8. Thus, please consider:

- Preparation and suggested material.
- The concepts and examples presented in the Lectures.
- A good SQL resource is the SQL Tutorial in W3Schools[5], where you find examples to rely on if you are stuck.
- In Lab exercise 3, students will use the MySQL DBMS installed at the ddi.hh.se Server Environment. To access it, visit the School Server Environment[6] and execute the included steps that that web page.
    - Get familiar with MySQL's phpMyAdmin
        - phpMyAdmin interface[7]
        - phpMyAdmin SQL console[8]

Deliverable

- Tables created and populated at your ddi.hh.se server.

NOTE:

- All students in the group are required execute the included SQL code on their respective ddi.hh.se server.
- **The group is responsible for making sure that the submitted results do not include cheating and plagiarism issues.**

Deadline

- Tuesday, February 18th by 13h.

[5] http://www.w3schools.com/sql/default.asp
[6] https://wagner24.ddi.hh.se/di4020/servenv
[7] https://www.tutorialspoint.com/phpmyadmin/phpmyadmin_databases.htm
[8] https://www.tutorialspoint.com/phpmyadmin/phpmyadmin_sql.htm

SQL Data Definition Language (DDL)

Data Definition Language (DDL) consists of SQL commands used to define, modify, and manage database structures such as tables, schemas, and constraints.

Main DDL Commands

- CREATE – Creates new database objects (tables, views, indexes, etc.).
- ALTER – Modifies the structure of an existing object.
- DROP – Deletes an object (table, database, etc.).
- TRUNCATE – Removes all records from a table but retains the structure.

CREATE TABLE

The CREATE clause is used to create new tables, views or other functions such as triggers or stored procedures.

Syntax

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

1. Execute the code below to create the country table.

```
-- Create the country table
CREATE TABLE country (
    country_id INT,
    name VARCHAR(80)
);
```

Considering the SQL statement above, it does not enforce any constraints, leading to potential data integrity issues. A properly structured table needs a unique identifier for each row, i.e., a Primary Key.

2. Execute the code below to create the cities table.

```
-- Create the cities table
CREATE TABLE cities (
    city_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(80) UNIQUE NOT NULL,
    country_id INT
);
```

Considering the SQL statement above, it guarantees of uniqueness for **city_id** and **name**, but does not enforce any referential constraints for **country_id**, leading to potential orphaned records, where a city references a non-existent country. A properly structured database enforces enforce referential integrity for relationships.

ALTER TABLE

The ALTER statement is used change (add, delete, or modify) the schema configuration such as table or attribute names, data types or adding new attributes and constraints.

Syntax

- To add a column

```
ALTER TABLE table_name ADD column_name datatype constraints;
```

- To modify a column

```
ALTER TABLE table_name MODIFY column_name new_datatype constraints;
```

Note: Some databases (like PostgreSQL) use `ALTER` instead of `MODIFY`.

- To rename a column

```
ALTER TABLE table_name RENAME COLUMN old_col_name TO new_col_name;
```

Note: Some databases (like MySQL) use use `CHANGE` instead of `RENAME`.

- To drop a column

```
ALTER TABLE table_name DROP COLUMN column_name;
```

- To add a Primary Key

```
ALTER TABLE table_name ADD PRIMARY KEY (column_name);
```

- To add a Primary Key

```
ALTER TABLE table_name ADD PRIMARY KEY (column_name);
```

- To add a Primary Key

```
ALTER TABLE table_name ADD PRIMARY KEY (column_name);
```

- To add a Foreign Key

```
ALTER TABLE table_name ADD FOREIGN KEY (column_name) REFERENCES
other_table(other_column);
```

3. Execute the code below to change the country table.

```
-- Add constraints to country table separately

ALTER TABLE country MODIFY country_id INT AUTO_INCREMENT;

ALTER TABLE country MODIFY name VARCHAR(80) NOT NULL;

ALTER TABLE country ADD PRIMARY KEY (country_id);

ALTER TABLE country ADD UNIQUE (name);
```

Considering the SQL statement above, it guarantees uniqueness for **country_id** and **name**, and automatically generates sequential values for **country_id**.

4. Execute the code below to change the cities table.

```
-- Add constraints separately, now referencing the country table

ALTER TABLE cities ADD FOREIGN KEY (country_id) REFERENCES
country(country_id) ON UPDATE CASCADE ON DELETE RESTRICT;
```

Considering the SQL statement above, it enforces referential integrity and enforce relationships between cities and country. It also includes referential actions (ON DELETE RESTRICT, ON UPDATE CASCADE).

ON DELETE RESTRICT prevents deletion of a country if there are cities referencing it.

Using ON UPDATE CASCADE, if the country_id of a country is updated, all related cities will have their country_id updated automatically.

DROP TABLE

DROP The DROP TABLE statement is used to drop an existing table in a database.

Syntax

```
DROP TABLE table_name;
```

5. Execute the code below to drop the country table.

```
-- Drop the country table

DROP TABLE country;
```

Considering the SQL statement above, which attempts to drop a table that other tables reference through a foreign key constraint, the statement will result in an error:

```
-- #3730 - Cannot drop table 'country ' referenced by a foreign key
constraint 'country_ibfk_1' on table country. Cannot delete or update
a parent row: a foreign key constraint fails.
```

6. Execute the code below to drop the cities table.

```
-- Drop the cities table

DROP TABLE cities;
```

Considering the SQL statement above, it executes successfully. Consequently, the cities table no longer exists in the database, but we need it. Let’s recreate it.

7. Execute the code below to create the country table.

```
-- -- Create the cities table, including constraints

CREATE TABLE cities (

    city_id INT AUTO_INCREMENT PRIMARY KEY,

    name VARCHAR(80) UNIQUE NOT NULL,

    country_id INT,

    FOREIGN KEY (country_id) REFERENCES country(country_id)

    ON DELETE RESTRICT

    ON UPDATE CASCADE

);
```

Considering the SQL statement above, note that it includes all constraints found in a properly structured table. Prefer this approach when writing SQL code to create tables, as it is easier to read, maintain, and avoids the need for a separate ALTER TABLE statement.

Create table weather

A relational model represents database tables as relations (sets of tuples) and defines attributes, keys, and constraints.

8. Following the previous examples of table creation, write an SQL statement to create a table for the weather relation

weather (weather_id, city_id, temp_lo, temp_hi, prcp, date, description)
Primary Key: weather_id
Foreign Key: city_id references cities(city_id)
Domain Constraints

- weather_id: Must be a unique integer, auto-increments from 1.
- city_id: Must match a valid city_id in the Cities table (ensures referential integrity).
- temp_lo & temp_hi: follow real-world temperature values.
- prcp: Cannot be negative (precipitation cannot be less than zero).
- date: Must be a valid calendar date.
- description: Free-form text.

SQL Data Manipulation Language (DML)

SQL Data Manipulation Language (DML) consists of commands that allow you to insert, update, delete, and retrieve data from a database. DML operations modify the actual data stored in tables without altering the table structure. DML commands include:

- INSERT – Adds new rows of data into a table.
- UPDATE – Modifies existing data in a table.
- DELETE – Removes data from a table.
- SELECT – Retrieves data from tables.

INSERT – Adding Data to Tables

The INSERT statement is used to add new records into a table.

Syntax

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

9. Insert Countries

This statement bellow will insert one row into the country table with the country name 'Sweden'.

```
INSERT INTO country (name) VALUES ('Sweden');
```

The statement below will insert multiple rows into the country table in one operation. If many rows are to be inserted, this is more efficient than inserting each row individually.

```
INSERT INTO country (name) VALUES
    ('Brazil'),
    ('United Kingdom'),
    ('Germany');
```

10. Insert Cities

```
INSERT INTO cities (name, country_id) VALUES
    ('Halmstad', (SELECT country_id FROM country WHERE name =
'Sweden'));

INSERT INTO cities (name, country_id) VALUES
    ('Stockholm', (SELECT country_id FROM country WHERE name =
'Sweden')),
    ('Porto Alegre', (SELECT country_id FROM country WHERE name =
'Brazil')),
    ('London', (SELECT country_id FROM country WHERE name =
'United Kingdom')),
    ('Berlin', (SELECT country_id FROM country WHERE name =
'Germany'));
```

The SQL statements above insert cities. Pay attention on the subqueries `(SELECT country_id FROM country WHERE name = 'country_name')` dynamically retrieves the country_id of the indicated `'country_name'` from the country table, ensuring the correct foreign key reference.

11. Insert Weather

```
INSERT INTO weather (city_id, temp_lo, temp_hi, prcp, date) VALUES
    ((SELECT city_id FROM cities WHERE name = 'Halmstad'), -1, 6,
0.25, '1994-11-27'),
    ((SELECT city_id FROM cities WHERE name = 'Stockholm'), -5, 5,
0.0, '1994-11-27'),
    ((SELECT city_id FROM cities WHERE name = 'Porto Alegre'), 20,
35, 0.1, '1994-11-27'),
    ((SELECT city_id FROM cities WHERE name = 'London'), 6, 10,
2.0, '1994-11-27'),
    ((SELECT city_id FROM cities WHERE name = 'Berlin'), 5, 15,
0.5, '1994-11-27');
```

The SQL statements above also includes subqueries to dynamically retrieve the city_id of the indicated city name from the country table to ensure the correct foreign key reference.

UPDATE – Modifying Existing Data

The UPDATE statement is used to modify existing records in a table.

Syntax

```
UPDATE table_name

SET column1 = value1, column2 = value2, ...

WHERE condition;
```

12. Update weather descriptions based on temperature

```
UPDATE weather
SET description = 'Cold and rainy'
WHERE temp_lo < 0 AND prcp > 0.25;
```

DELETE – Removing Data from Tables

The DELETE statement is used to remove existing records from a table.

Syntax

```
DELETE FROM table_name

WHERE condition;
```

Delete all records from `weather` where it rains to much (above 2 mm)

```
DELETE FROM weather WHERE prcp > 2.0;
```

SELECT – Retrieving Data from Tables

The SELECT statement is used to fetch data from tables.

Syntax

```
SELECT column1, column2, ...

FROM table_name

WHERE condition;
```

13. Select all columns from a table, e.g., all columns from the country table.

```
SELECT * FROM country;
```

14. Select specific columns, e.g., only the name column from the country table.

```
SELECT name FROM country;
```

15. Select with a Condition (WHERE), e.g., details of Sweden from the country table.

```
SELECT * FROM country WHERE name = 'Sweden';
```

16. Select with Sorting (ORDER BY), e.g., all countries sorted alphabetically (A-Z).

```
SELECT * FROM country ORDER BY name ASC;
```

17. Count Rows in a Table, e.g., the number of countries in the table.

```
SELECT COUNT(*) FROM country;
```

Joins Between Tables

Queries can access multiple tables at once or access the same table in such a way that multiple instances of the table are being processed at the same time. Joins are necessary when working with normalized databases that separate data into multiple tables. SQL joins are used to combine data from multiple tables based on a common column. This allows us to retrieve related information across different tables.

Types of Joins

- `JOIN or INNER JOIN`: Returns only matching rows in both tables.
  - Cities with weather records.
- `LEFT JOIN`: Returns all rows from the left table, and matching rows from the right table.
  - All cities, even if no weather data exists.
- `RIGHT JOIN`: Returns all rows from the right table, and matching rows from the left table.
  - All weather records, even if no city exists.

JOIN or INNER JOIN

A JOIN or INNER JOIN returns only rows where both tables have matching values.

18. Joins the weather table and the cities table to retrieve all weather records along with city and country names.

```
SELECT cities.name AS city, country.name AS country,
weather.temp_lo, weather.temp_hi, weather.prcp, weather.date
FROM weather
JOIN cities ON weather.city_id = cities.city_id
JOIN country ON cities.country_id = country.country_id;
```

Only cities with weather data are returned.

The query retrieves weather data along with the city name and country name. It joins tables weather, cities, and country, and uses explicit table names instead of tables aliases for clarity. However, it uses column aliases to rename columns in the output. An alias in SQL is a temporary name given to a table or column in a query to make it shorter, clearer, or easier to read. A table alias is a short name for a table, useful in joins and complex queries.

Aliases are mainly used for:

- Renaming columns in the result set
  - For example: `cities.name AS city`, which means the name column will now appear as city_name in the result.
- Making queries shorter and more readable (especially with multiple tables).
- Avoiding conflicts when joining tables with similar column names.

19. Find all cities where it rained (`prcp > 0`)

```
SELECT c.name AS city, co.name AS country, w.date, w.prcp
FROM weather w
JOIN cities c ON w.city_id = c.city_id
JOIN country co ON c.country_id = co.country_id
WHERE w.prcp > 0;
```

Now, instead of typing weather, cities, and country multiple times, we use w, c, and co, making the query shorter and easier to read.

LEFT JOIN – All from Left Table, Matching from Right

A LEFT JOIN returns all rows from the left table and matching rows from the right table. If there is no match, NULL is returned.

20. Get All Cities, Even If They Have No Weather Data

```
SELECT c.name AS city, co.name AS country, w.temp_lo, w.temp_hi,
w.prcp, w.date
FROM cities c
LEFT JOIN weather w ON c.city_id = w.city_id
JOIN country co ON c.country_id = co.country_id;
```

Cities with no weather data will show NULL for weather columns.

RIGHT JOIN – All from Right Table, Matching from Left

A RIGHT JOIN returns all rows from the right table and matching rows from the left table.

21. Get All Weather Records, Even If No City Exists

```
SELECT c.name AS city, w.temp_lo, w.temp_hi, w.prcp, weather.date
FROM cities c
RIGHT JOIN weather w ON c.city_id = w.city_id;
```

Ensures all weather records are shown, even if the city is missing.

SQL Aggregates

SQL aggregate functions are used to perform calculations on a set of values and return a single summarized result. Common aggregate functions are:

- COUNT(): Counts the number of rows
- SUM(): Adds up all values in a column
- AVG(): Calculates the average value
- MAX(): Finds the highest value
- MIN(): Finds the lowest value

22. Retrieve the average high temperature.

```
SELECT AVG(temp_hi) AS avg_high_temp FROM weather;
```

Aggregate functions are commonly used with the GROUP BY clause to group data.

23. Find the highest recorded temperature in each country, i.e., group the data by country and returns the highest temperature in each country.

```
SELECT co.name AS country, MAX(w.temp_hi) AS highest_temp
FROM weather w
JOIN cities c ON w.city_id = c.city_id
JOIN country co ON c.country_id = co.country_id
GROUP BY co.name;
```

It is also possible to filter grouped results using the HAVING clause, which is similar to WHERE, but for aggregated results.

24. Find countries with an average precipitation greater than 0.2 mm

```
SELECT co.name AS country, AVG(w.prcp) AS avg_precipitation
FROM weather w
JOIN cities c ON w.city_id = c.city_id
JOIN country co ON c.country_id = co.country_id
GROUP BY co.name
HAVING AVG(w.prcp) > 0.2;
```

You can use multiple aggregate functions in one query.

25. Retrieve the total number of records, average high temperature, highest temperature recorded and the lowest temperature recorded.

```
SELECT COUNT(*) AS total_records,
       AVG(temp_hi) AS avg_temp,
       MAX(temp_hi) AS max_temp,
       MIN(temp_lo) AS min_temp
FROM weather;
```

Results

Based on the SQL statements above, the following tables and data are expected to exist in the database:

- Tables
    - country
    - cities
    - weather
- Expected Data
    - country
        - At least "Sweden" and potentially other countries.
    - cities
        - Cities mapped to their respective country_id, ensuring referential integrity.
    - weather
        - Weather data for various cities.
        - Records include precipitation and temperature data.
        - Some records might have been deleted (e.g., where precipitation > 2 mm).

Congratulations, you have now completed Lab Exercise 3.