C-408 DAY-4

Foundations of SRE -RDBMS and SQL

Priyansh Kakani 13-02-2025

- today we have learned about generating the statements in an automated way as we can create multiple insert statements using the select command.
  - SELECT CONCAT('INSERT INTO employees (emp_id, name, salary, department) VALUES (',emp_id, ', ',QUOTE(name), ', ',salary, ', ',QUOTE(department), ');') AS insert_statement FROM source_employees;
  - In the above example we are copying data into employees from the source_employees so this will give multiple insert statements which we can run in a single run command.
- If we want to add a new attribute into all the insert commands so you can do it using a procedure it will be easy and many of the tasks are done using procedures.
- Also to create a index to the table there are 2 ways
  - Inside the table index_key is indexing the table
    - CREATE TABLE STUDENT(
      ID INT PRIMARY KEY,
      INDEX INDEX_KEY(ID)
      );
  - CREATE INDEX INDEX_KEY ON STUDENT(ID,NAME);
  - Here the index_key is index file consisted of id and name
- The index are used to get a better performance when it comes to searching sorting or any other task you want to do.
- EXPLAIN select * from employee where name='priyansh';
  - This is a simple command that deletes the record where the name is priyansh but also it gives some performance evaluation to the user where user can check the following
    - **id**: An identifier for the query step.
    - **select_type**: The type of query (e.g., simple, primary, or complex).
    - **table**: The table that the database is accessing (in this case, username).
    - **type**: The join type or how the table is accessed (e.g., ALL means a full table scan, which is less efficient than indexed access).
    - **possible_keys**: Any indexes that could be used for the query.
    - **key**: The actual index used for the query (if any).
    - **rows**: The estimated number of rows the query will examine.
    - **Extra**: Any additional information about how the query is processed (e.g., whether a temporary table is used).
- **ON DELETE CASCADE is** used when we want to delete a record from the parent table so automatically it gets deleted from the child table also.
- **ON DELETE SET NULL is** used when we want to delete a record from the parent table so it automatically doesn't delete in child but the row is as it is but the column will now contain NULL for the deleted one.

- **ON UPDATE SET NULL is** used when we want to update a record from the parent table so it doesn't automatically updates in child one. the column in child will now contain NULL for the deleted one.
- There are some cases that must be remembered.
    - If we don't use on delete cascade or any other constraint then we can't delete parent 's row it will throw an error. If we delete from child then then result will be deleted but not from parent table.
- To create a looping in procedure we can create like this

```
DELIMITER $$

CREATE PROCEDURE print_numbers_up_to_n(IN max_value INT)
BEGIN
    DECLARE counter INT DEFAULT 1;

    -- Loop while counter is less than or equal to max_value
    WHILE counter <= max_value DO
        -- You can perform any action here (e.g., printing the number)
        SELECT counter;

        -- Increment the counter
        SET counter = counter + 1;
    END WHILE;
END $$

DELIMITER ;
```

- **TRIGGERS** in SQL is a special kind of stored procedure that automatically runs when certain events occur on a table or view, such as **INSERT**, **UPDATE**, or **DELETE.**
    - e.g. if we delete a entry from the database so to create a record of logging .we use this so when the tuple is deleted it is inserted into a log_table so we can find in future if we want some specific data entry.

  EXAMPLE- we have created a trigger and it will run before the delete operation on table courses it will run in loop until the records are not finished and it will insert log into table with current timestamp and a specific text and by old.course_id we can access the old course_id.

```
create trigger before_course_deletion
before delete on COURSES
for each row
insert into enrollments_logging(enrollment_id, action, change_date)
values(old.COURSE_ID, 'DELETE BASED ON DELETION FROM COURSE AND STUDENT', NOW());
```

- candidate keys are just theoretical concepts they don't exist into the real life scenarios.
- **COALESCE** is a SQL function that returns the first non-NULL value from a list of arguments. It's useful for handling NULL values in queries, allowing you to replace them with a default value. Here is an example- this selects value=no dept when department name=NULL.

```
SELECT
    e.name,
    COALESCE(d.department_name, 'No Department') as department,
    e.salary,
```

GitHub
GIT CHEAT SHEET

Git is the free and open source distributed version control system that's responsible for everything GitHub related that happens locally on your computer. This cheat sheet features the most important and commonly used Git commands for easy reference.

INSTALLATION & GUIS

With platform specific installers for Git, GitHub also provides the ease of staying up-to-date with the latest releases of the command line tool while providing a graphical user interface for day-to-day interaction, review, and repository synchronization.

GitHub for Windows
https://windows.github.com

GitHub for Mac
https://mac.github.com

For Linux and Solaris platforms, the latest release is available on the official Git web site.

Git for All Platforms
http://git-scm.com

SETUP

Configuring user information used across all local repositories

`git config --global user.name "[firstname lastname]"`

set a name that is identifiable for credit when review version history

`git config --global user.email "[valid-email]"`

set an email address that will be associated with each history marker

`git config --global color.ui auto`

set automatic command line coloring for Git for easy reviewing

SETUP & INIT

Configuring user information, initializing and cloning repositories

`git init`

initialize an existing directory as a Git repository

`git clone [url]`

retrieve an entire repository from a hosted location via URL

STAGE & SNAPSHOT

Working with snapshots and the Git staging area

`git status`

show modified files in working directory, staged for your next commit

`git add [file]`

add a file as it looks now to your next commit (stage)

`git reset [file]`

unstage a file while retaining the changes in working directory

`git diff`

diff of what is changed but not staged

`git diff --staged`

diff of what is staged but not yet committed

`git commit -m "[descriptive message]"`

commit your staged content as a new commit snapshot

BRANCH & MERGE

Isolating work in branches, changing context, and integrating changes

`git branch`

list your branches. a * will appear next to the currently active branch

`git branch [branch-name]`

create a new branch at the current commit

`git checkout`

switch to another branch and check it out into your working directory

`git merge [branch]`

merge the specified branch's history into the current one

`git log`

show all commits in the current branch's history

- DONE some leetcode questions today