University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: 4

Date: 11/29/2024

Group Number: 27

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Kevin Li	32137903	e7o5s	kevxtroyg@gmail.com
Kiana Li	45863628	t0s7f	kianali1208@gmail.com
Junru Chen	74767625	u2q3r	junruchen2021@163.com

(3.2.3) Short Description

The Astro-Archive project is a comprehensive database-driven application that catalogues and analyzes astronomical data to provide insights into celestial objects. The database organizes detailed information about galaxies, stars, planetary bodies, nebulae, black holes, and halos, enabling structured exploration of the cosmos. Users can interact with this data through advanced queries, additions, and deletions, facilitating a dynamic and user-friendly experience. Each celestial entity is stored with a wide range of attributes, such as the ages and classifications of galaxies, the luminosities and associated galaxies of nebulae, the masses and radii of black holes, and the types and temperatures of planetary bodies and their moons. This system combines relational database concepts with real-world astronomical datasets, offering a platform for both practical applications and theoretical learning. It especially provides a foundation for educational purposes, fostering a deeper understanding of the universe.

The project enables users to execute parameterized queries that meet a variety of analytical requirements. These include projection queries to retrieve specific columns, selection queries to filter data based on conditions, and join queries to analyze relationships between different celestial entities. Aggregation queries provide advanced grouping and summarization capabilities, while division queries allow for identifying entities meeting "all conditions" criteria, such as stars whose systems include all planet types. Users can modify the database by inserting new data, such as adding details about halos or planetary bodies, deleting tuples based on user-defined conditions, and updating tuples with the most recent data.

Our final schema is a little bit different from what we had turned in previously. We added the relation planetary body orbit star and therefore added a new foreign key star name in planetary body entity to reference star entity, which further complemented the relations and made them more similar to the structure of the real universe. In addition, we add on delete cascade in the terrestrial planet, gas giant, and ice giant for their references to the planetary body. Since they are in an ISA relationship, deleting the planetary body will delete the corresponding entity under ISA too. We fixed this bug while we were implementing delete.

Queries:

Insertion

```
INSERT INTO Halo (halo_name, h_luminosity, h_galaxy_name)

VALUES (:halo_name, :h_luminosity, :h_galaxy_name)
```

project_e7o5s_t0s7f_u2q3r/backend/service/service.js   line389-390

Delete

```
DELETE FROM Gas_giant gg
    WHERE gg.gas_name = :delete_pb
DELETE FROM Ice_giant ig
    WHERE ig.ice_name = :delete_pb
DELETE FROM Terrestrial_planet tp
    WHERE tp.tp_name = :delete_pb
DELETE FROM Moon m
    WHERE m.planetary_body_name = :delete_pb
DELETE FROM Planetary_body pb
    WHERE pb.pb_name = :delete_pb
```

project_e7o5s_t0s7f_u2q3r/backend/service/service.js   line427-446

Update

UPDATE Moon

SET ${updateFields.join(", ")}

WHERE planetary_body_name = :planetary_body_name AND moon_name = :moon_name

project_e7o5s_t0s7f_u2q3r/backend/service/service.js line526-528

Selection

SELECT :set_columns

FROM Star s3

WHERE :set_conditions

project_e7o5s_t0s7f_u2q3r/sql/query/default/selection.sql

Projection

SELECT :set_columns

FROM Galaxy g

WHERE g.age > :age - 5 AND g.age < :age + 5

project_e7o5s_t0s7f_u2q3r/sql/query/default/projection.sql

Join

SELECT h.halo_name, g.galaxy_name, g.age

FROM Halo h NATURAL JOIN Galaxy g

WHERE g.age > :gt AND h.h_galaxy_name = g.galaxy_name

project_e7o5s_t0s7f_u2q3r/sql/query/default/join.sql

Aggregation with Group By

select :set_group_by, :aggregation(n.N_LUMINOSITY)

from Nebula n

group by :set_group_by

project_e7o5s_t0s7f_u2q3r/sql/query/default/aggregation_GB.sql

summarizes data by grouping rows based on one or more columns and applying aggregate functions to each group according to the user’s choice in the Nebula table. For example, we can find the average luminosity for each type of nebula and galaxy they are in.

Aggregation with Having

select n.NEBULA_TYPE, :aggregation(n.N_LUMINOSITY)

from Nebula n

group by n.NEBULA_TYPE

having :set_conditions

project_e7o5s_t0s7f_u2q3r/sql/query/default/aggregation_having.sql

filters grouped data based on conditions applied to aggregate functions according to the user's choice after the clause has been processed in the Nebula table. For example, we can find the average luminosity for each type of nebula whose aggregation is greater than a certain value of luminosity.

Nested Aggregation with Group By

select :set_group, :aggregation(n.N_LUMINOSITY)

from Nebula n

group by :set_group

having :aggregation(n.N_LUMINOSITY) :condition (select :aggregate_nested(n1.N_LUMINOSITY) from Nebula n1)

project_e7o5s_t0s7f_u2q3r/sql/query/default/nested.sql

computes aggregate functions within subqueries and uses their results in the outer query to further group and aggregate data according to the user's choice in the Nebula table. For example, we can find the nebula types that their average luminosities are greater than the average luminosity of all nebulae.

Division

```
CREATE OR REPLACE VIEW TEMP_Terrestrial_planet AS

  SELECT t.tp_name as name

  FROM Terrestrial_planet t

--split

CREATE OR REPLACE VIEW TEMP_Ice_giant AS

  SELECT i.ice_name as name

  FROM Ice_giant i

--split

CREATE OR REPLACE VIEW TEMP_Gas_giant AS

  SELECT g.gas_name as name

  FROM Gas_giant g

--split

CREATE OR REPLACE VIEW STAR_PLANET_TYPES AS

  :set_types

--split

SELECT s.star_name

FROM Star s

WHERE NOT EXISTS

  (SELECT sp1.planet_type
    FROM STAR_PLANET_TYPES sp1
    WHERE NOT EXISTS
      (SELECT sp2.planet_type
      FROM STAR_PLANET_TYPES sp2
      WHERE sp2.star_name = s.star_name AND sp2.planet_type =
sp1.planet_type))
```

project_e7o5s_t0s7f_u2q3r/sql/query/default/planet_division.sql

retrieves rows from one table that are related to all rows in another table based on a specific condition according to the user's choice in the Planetary Body table. For example, we can find all star names whose system includes ALL planet types (Terrestrial, Ice Giant, & Gas Giant).