Lab 4
Aggregation, Grouping, ROLLUP, and CUBE in SQL

CSE 4508

Relational Database Management System Lab

October 15, 2024

Contents

1 Topics

1.1 Semi-Structured Data

Regular expressions allow us to search for complex patterns in string data. Common use cases include extracting or filtering data based on specific patterns.
Example: Extracting a 4-digit year from string data using `REGEXP_SUBSTR`.

```
-- Create and insert data into table
DROP TABLE t1;
CREATE TABLE t1 (data VARCHAR2(50));
INSERT INTO t1 VALUES ('FALL 2014');
INSERT INTO t1 VALUES ('2014 CODE-B');
INSERT INTO t1 VALUES ('CODE-A 2014 CODE-D');
INSERT INTO t1 VALUES ('ADSHLHSALK');
INSERT INTO t1 VALUES ('FALL 2004');
COMMIT;

-- Select rows containing the year >= 2014
SELECT *
FROM t1
WHERE TO_NUMBER(REGEXP_SUBSTR(data, '\d{4}')) >= 2014;
```

Input:

```
FALL 2014
2014 CODE-B
CODE-A 2014 CODE-D
ADSHLHSALK
FALL 2004
```

Output:

```
FALL 2014
2014 CODE-B
CODE-A 2014 CODE-D
```

1.2 GROUP BY and HAVING

The `GROUP BY` clause is used for aggregating data, while the `HAVING` clause filters groups based on aggregate conditions.

Example: Summing sales by department and filtering groups with sales over 1000.

```
-- Drop table if it exists
DROP TABLE order_details;

-- Create table order_details
CREATE TABLE order_details (
    department VARCHAR2(50),
    sales NUMBER
);

-- Insert data into order_details
INSERT INTO order_details (department, sales) VALUES ('Electronics
   ', 1200);
INSERT INTO order_details (department, sales) VALUES ('Furniture',
   900);
```

```
INSERT INTO order_details (department, sales) VALUES ('Electronics
    ', 1500);
INSERT INTO order_details (department, sales) VALUES ('Furniture',
    2000);
INSERT INTO order_details (department, sales) VALUES ('Clothing',
    1100);

-- Commit the transaction
COMMIT;
```

```
-- Group sales by department and filter by total sales > 1000
SELECT department, SUM(sales) AS total_sales
FROM order_details
GROUP BY department
HAVING SUM(sales) > 1500;
```

Input:

Department	Sales
Electronics	1200
Furniture	900
Electronics	1500
Furniture	2000
Clothing	1100

Output:

Department	Total Sales
Electronics	2700
Furniture	2900

1.3 ROLLUP for Subtotals

The ROLLUP operator generates subtotals for specified columns in a GROUP BY query.

Example: Calculating subtotals for sales by department.

```
-- Generate subtotals using ROLLUP
SELECT department, SUM(sales) AS total_sales
FROM order_details
GROUP BY ROLLUP(department)
ORDER BY department;
```

Output:

Department	Total Sales	
Clothing	1100	
Electronics	2700	
Furniture	2900	
	6700	-- Grand Total

1.4 CUBE for Cross-Tabulation

The CUBE operator calculates subtotals for all possible combinations of grouping columns.

Example: Counting employees by department and designation using CUBE.

```
-- Drop table if it exists
DROP TABLE employees;

-- Create table employees with department names
CREATE TABLE employees (
    employee_id NUMBER,
    department VARCHAR2(50),
    designation VARCHAR2(50),
    hire_date DATE
);

-- Insert data into employees
INSERT INTO employees (employee_id, department, designation,
   hire_date)
VALUES (1, 'Sales', 'Manager', TO_DATE('15-02-2005', 'DD-MM-YYYY'))
   ;

INSERT INTO employees (employee_id, department, designation,
   hire_date)
VALUES (2, 'Sales', 'Manager', TO_DATE('21-01-2007', 'DD-MM-YYYY'))
   ;

INSERT INTO employees (employee_id, department, designation,
   hire_date)
VALUES (3, 'Human Resources', 'Assistant', TO_DATE('10-05-2010', '
   DD-MM-YYYY'));

-- Commit the transaction
COMMIT;
```

```
-- Count employees with subtotals for all combinations
SELECT department, designation, COUNT(*) AS total
FROM employees
GROUP BY CUBE(department, designation)
ORDER BY department, designation;
```

Output:

```
Department          Designation     Total
Human Resources     Assistant        1
Human Resources                      1
Sales               Manager          2
Sales                                2
                                     3  -- Grand Total
```

1.5 ROLLUP with GROUPING Function

ROLLUP: Generates subtotals and grand totals for hierarchical groupings, aggregating data at multiple levels.
GROUPING(): A function that returns 1 for columns that are aggregated and 0 for columns that have regular values. This helps identify whether a NULL in the result set is due to aggregation or represents missing data.

```
SELECT
    category,
    EXTRACT(YEAR FROM sale_date) AS year,
    SUM(total_price) AS total_revenue,
```

```
    GROUPING(category) AS category_group,
    GROUPING(year) AS year_group
FROM
    sales
JOIN
    books ON sales.book_id = books.book_id
JOIN
    categories ON books.category_id = categories.category_id
GROUP BY
    ROLLUP (category, year);
```

Category	Year	Total Revenue	Category Group	Year Group
Fiction	2022	5000	0	0
Fiction	NULL	5000	0	1
NULL	NULL	8000	1	1

Table 1: ROLLUP with GROUPING Function Example Output

2 Tasks

A bus transportation company maintains an operational database tracking trips, buses, drivers, and depots. To perform analytical tasks such as summarizing kilometers traveled, fuel consumption, and passenger counts, the company wants to build a data warehouse. You need design and implement this data warehouse using OLAP operations like ROLLUP, CUBE, and GROUPING for multidimensional analysis.

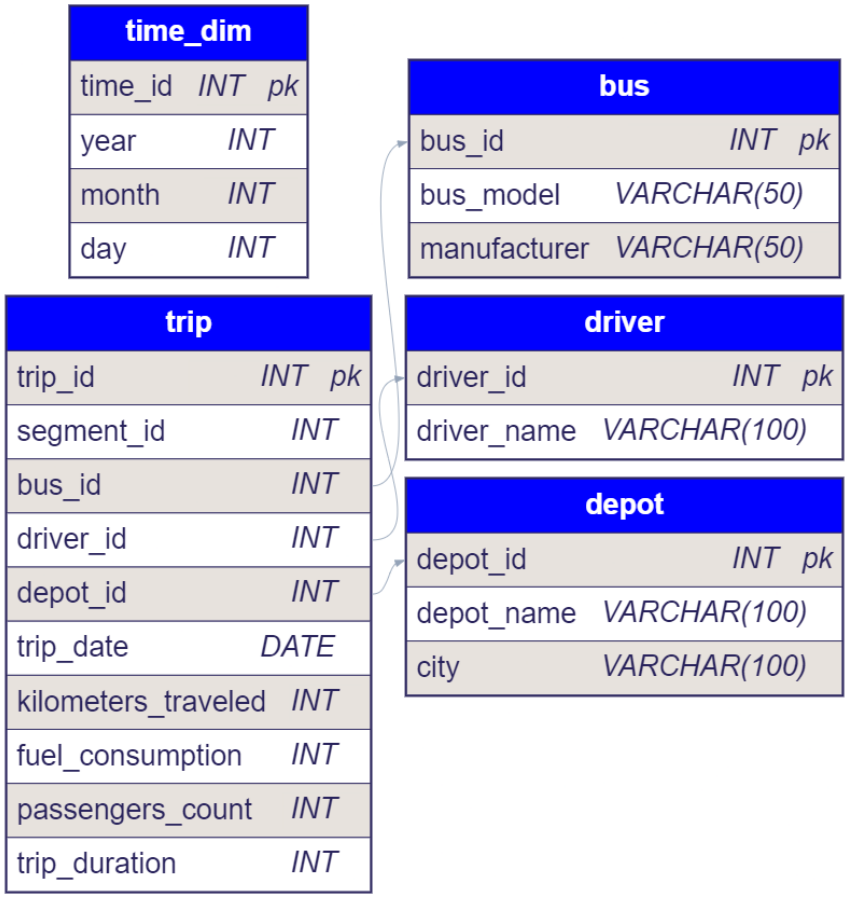

Figure 1: Database Schema

1. Write a query to calculate the total kilometers traveled, fuel consumption, and passengers per bus, grouped by year, month, and day.
2. Use the `ROLLUP` function to calculate subtotals and grand totals for kilometers traveled, fuel consumption, and passengers, grouped by bus model, year, and month.
3. Use the `CUBE` function to generate all possible combinations of subtotals for kilometers traveled, fuel consumption, and passengers, grouped by bus model, depot, and year.
4. Use the `GROUPING` function to distinguish between detailed rows and subtotal rows in a `ROLLUP` query for bus model, year, and month.
5. Calculate the total kilometers traveled, fuel consumption, and passengers, grouped by bus model, depot, and year.