CS 252: Fundamentals of Relational Databases: SQL5

Dr. Alexandra I. Cristea
http://www.dcs.warwick.ac.uk/~acristea/

Interim Summary

Topics covered:
- Indexes, Sequences,
- Relational Algebra and Advanced Joins

This lecture

- Query re-write
- NULLs and 3-valued logic
- Top N Analysis
- Security Privileges

CASE

The CASE operation returns one of a specified set of scalar values depending on some condition.

```sql
CASE
WHEN CD_Year.year >= 2000 THEN 'Noughties'
WHEN CD_Year.year >= 1990 THEN 'Nineties'
WHEN CD_Year.year >= 1980 THEN 'Eighties'
WHEN CD_Year.year >= 1970 THEN 'Eighties'
ELSE 'Before my time'
END
```

Redundancy of GROUP BY

For every select expression that involves GROUP BY or HAVING there is an equivalent expression that does not.

```sql
SELECT student, SUM(mark) Total FROM CS_marks GROUP BY student;
```

can be re-written to be:

```sql
SELECT DISTINCT student, ( SELECT SUM(mark) FROM CS_marks CSM WHERE CS_marks.student = CSM.student ) Total FROM CS_marks;
```

Example with HAVING

```sql
SELECT student, SUM(mark) Total FROM CS_marks GROUP BY student HAVING SUM(mark) > 100;
```

can be re-written as

```sql
SELECT student, Total FROM ( SELECT student, SUM(mark) AS Total FROM CS_marks GROUP BY student ) t WHERE Total > 100
```
NULL
What does NULL mean?
SQL represents the fact that some piece of information is missing by the use of a special marker 'NULL'.
Mark of student Tim is NULL
This means:
• We know that Tim exists
• We know that Tim has a mark
• We don’t know what the mark is

NULL and scalar expressions
If x or y (or both) is NULL then
x + y
x - y
x * y
x / y
all evaluate to NULL

NULL and aggregate functions
• Aggregate functions ignore NULL's apart from COUNT(*)
• So SUM(column) is not the same as adding up all the values in the column with '+'
• Aggregate functions where the input column contains no values return NULL
• COUNT(*) returns 0 when counting no rows.

NULL and three-valued logic
SQL conditions can evaluate to true, false or unknown. This is a three-valued logic and our logic operators AND/OR/NOT need to be defined over all 3 values:

<table>
<thead>
<tr>
<th></th>
<th>t</th>
<th>u</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>t</td>
<td>t</td>
<td>u</td>
</tr>
<tr>
<td></td>
<td>u</td>
<td>u</td>
<td>f</td>
</tr>
<tr>
<td>NOT</td>
<td>t</td>
<td>f</td>
<td>u</td>
</tr>
<tr>
<td></td>
<td>f</td>
<td>t</td>
<td>u</td>
</tr>
</tbody>
</table>

If our WHERE condition evaluates to True then the row is included in the result; otherwise (if it is false or unknown) it is discarded.

Default Column Values
When defining a table using CREATE TABLE the default value of a column is NULL. This can be changed in the CREATE TABLE command:

```
CREATE TABLE CS_marks
(
    student VARCHAR(20),
    course CHAR(3),
    mark INTEGER DEFAULT 50
);
```

```
INSERT INTO CS_marks (student,course) VALUES ('Paul Smith','DBS');
```
Inserts the row ('Paul Smith', 'DBS', 50) into the CS_marks table.
TOP N Analysis

We can sort the results of a SELECT statement with ORDER BY and ORDER BY ... DESC.

Once we have defined an ORDER we may only want to return the top 10 records.

In SQL Server you can write:
```
SELECT TOP 10 FROM CS_marks ORDER BY mark
```

In MySQL you can write:
```
SELECT FROM CS_marks ORDER BY mark
```

LIMIT 10

In Oracle things are more complicated.

When Oracle evaluates a select query it appends a column to the result called 'ROWNUM' before any ORDER BY, GROUP BY or DISTINCT clauses are run. So the query:
```
SELECT Student, Mark, ROWNUM FROM CS_marks;
```

Gives:
```
STUDENT MARK ROWNUM
----------------------------------------
Paul Smith 43 1
Rachel Sewell 57 2
Helen Treacy 72 3
Paul Smith 65 4
Rachel Sewell 42 5
```

which isn't very helpful, but if we force Oracle to append the ROWNUM after it has done the order then we can use it to limit the rows we get.

```
SELECT Student, Mark, ROWNUM FROM (SELECT Student, Mark, ROWNUM FROM CS_marks ORDER BY Mark);
```

Gives us the lowest two marks:
```
STUDENT MARK ROWNUM
-----------------------------------
Rachel Sewell 42 1
Paul Smith 43 2
```

SQL Syntax for Security Rules

```
GRANT [privilege-commalist | ALL PRIVILEGES]
ON object-name
TO [authorisation_id_list | PUBLIC]
[WITH GRANT OPTION]
```

Each privilege is one of the following:
- SELECT
- INSERT
- UPDATE
- REFERENCES

The REFERENCES allows privileges to be granted on named table(s) in integrity constraints of CREATE TABLE.

The GRANT OPTION allows the named users to pass the privileges on to other users.
**Exercise:**

Base relation STATS looks like this:

```plaintext
STATS (USERID, SEX, DEPENDENTS, JOB, SALARY, TAX, AUDITS)
```

**Primary Key:** (USERID)

Write rules for the following:

(a) User John SELECT privileges over the entire table.
(b) User Fred UPDATE privileges over the TAX column only.
(c) How would you grant user Pope full privileges over rows for job type 'Priest' only?

---

**Grant and Revoke**

If a user A grants privileges to user B, then they can also revoke them e.g.

```sql
REVOKE ALL PRIVILEGES ON STATS FROM John;
```

**SQL REVOKE syntax**

```sql
REVOKE [GRANT OPTION FOR] [privilege_list | ALL PRIVILEGES]
ON object_name
FROM [authorisation_list|PUBLIC] [RESTRICT|CASCADE]
```

If RESTRICT option is given then the command is not executed if any dependent rules exist i.e. those created by other users through the WITH GRANT OPTION.

CASCADE will force a REVOKE on any dependent rules.

---

**Listing your security privileges**

Under Oracle you can list your security privileges by using

```sql
SELECT * FROM SESSION_PRIVS;
```

```
PRIVILEGE
----------------------------------------
CREATE SESSION
ALTER SESSION
CREATE TABLE
CREATE CLUSTER
CREATE SYNONYM
CREATE VIEW
CREATE SEQUENCE
CREATE DATABASE LINK
8 rows selected.
```

---

**Interim Summary**

- Query re-write
- NULLs and 3-valued logic
- Top N Analysis
- Security Privileges