CASE

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

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.

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

can be re-written to be:

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

Example with HAVING

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

can be re-written as

SELECT DISTINCT student,
  ( SELECT SUM(mark) FROM CS_marks CSM WHERE CS_marks.student = CSM.student ) Total
FROM CS_marks WHERE
  ( SELECT SUM(mark) FROM CS_marks CSM WHERE CS_marks.student = CSM.student ) > 100;
NULLs

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's 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's 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 rows return NULL apart from COUNT() which returns 0.

NULL's and three-valued logic

SQL expression’s can evaluate to true,false or NULL. 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>t</td>
<td>t</td>
<td>u</td>
<td>f</td>
</tr>
<tr>
<td>u</td>
<td>u</td>
<td>u</td>
<td>f</td>
</tr>
<tr>
<td>f</td>
<td>f</td>
<td>f</td>
<td>f</td>
</tr>
</tbody>
</table>
If our WHERE clause evaluates to True then the row is included in our query otherwise (if it is false or unknown) it is discarded.

**Default Attribute Values**

When defining a table using CREATE TABLE the default value of an attribute is NULL. This can be changed in the CREATE TABLE command:

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

INSERT INTO CS_marks (student,course) VALUES ('Paul Smith','DBS');

INSERT INTO CS_marks (student,course) VALUES ('Paul Smith','DBS')
```

Inserts the values 'Paul Smith','DBS' and 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

```sql
SELECT TOP 10 FROM CS_marks ORDER BY mark
```

In MySQL you can write

```sql
SELECT FROM CS_marks ORDER BY mark LIMIT 10
```

In Oracle things are more complicated.

When Oracle executes 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

```sql
SELECT Student,Mark,ROWNUM FROM CS_marks;
```

Gives:

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>MARK</th>
<th>ROWNUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paul Smith</td>
<td>43</td>
<td>1</td>
</tr>
<tr>
<td>Rachel Sewell</td>
<td>57</td>
<td>2</td>
</tr>
<tr>
<td>Helen Treacy</td>
<td>72</td>
<td>3</td>
</tr>
<tr>
<td>Paul Smith</td>
<td>65</td>
<td>4</td>
</tr>
<tr>
<td>Rachel Sewell</td>
<td>42</td>
<td>5</td>
</tr>
</tbody>
</table>
while SELECT Student, Mark, ROWNUM FROM CS_marks ORDER BY Mark;

Gives:

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>MARK</th>
<th>ROWNUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rachel Sewell</td>
<td>42</td>
<td>5</td>
</tr>
<tr>
<td>Paul Smith</td>
<td>43</td>
<td>1</td>
</tr>
<tr>
<td>Rachel Sewell</td>
<td>57</td>
<td>2</td>
</tr>
<tr>
<td>Paul Smith</td>
<td>65</td>
<td>4</td>
</tr>
<tr>
<td>Helen Treacy</td>
<td>72</td>
<td>3</td>
</tr>
</tbody>
</table>

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);
```

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>MARK</th>
<th>ROWNUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rachel Sewell</td>
<td>42</td>
<td>1</td>
</tr>
<tr>
<td>Paul Smith</td>
<td>43</td>
<td>2</td>
</tr>
<tr>
<td>Rachel Sewell</td>
<td>57</td>
<td>3</td>
</tr>
<tr>
<td>Paul Smith</td>
<td>65</td>
<td>4</td>
</tr>
<tr>
<td>Helen Treacy</td>
<td>72</td>
<td>5</td>
</tr>
</tbody>
</table>

---

**SQL Syntax 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
- DELETE
- INSERT [ (attribute-commalist)]
- UPDATE [ (attribute-commalist) ]
- REFERENCES [ (attribute-commalist) ]

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:

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

Write rules for the following:

(a) User John SELECT privileges over the entire relation.

(b) User Fred UPDATE privileges over the TAX column only.

(c) How would you grant user Pope full privileges over tuple 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.

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

**SQL REVOKE syntax**

```
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 execute 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

```
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.