Appendix E Implementing Derivation Rules

Derivation Rules

Some fact types in a conceptual schema may be identified as derived. By default these fact types will not appear as columns in a relational table, so the values are not established until the time of a query. This sometimes referred to as "lazy" evaluation of the derivation rule. The alternative is to create (or update) the derived value as soon as the necessary information is available (or changed), which is sometimes referred to as "eager" evaluation. The choice between these two alternatives depends on whether the main consideration is the optimization of queries or the optimization of updates.

All derivation rules in a conceptual model should be realized using some appropriate technique in the corresponding database. Tables E.1 and E.2 list the main options available.

Table E.1 Eager evaluation of derivation rules

Technique	*Features*
Trigger	Triggers can be attached to tables that contain data used in derivation rules, so that any modification to the relevant data will automatically update any derived values. It's important to ensure that the triggers cover all possible cases, and this can become complex where the derivation uses data from multiple tables. Triggers are discussed in more detail in Section 13.5 of the book.
Generated Column	The SQL standard specifies generated columns as columns that contain values computed according to some specification that has been provided. Unlike their "lazy" equivalent, these values are actually stored. More information on generated columns is given later in this appendix.
Stored Procedure	A stored procedure could contain code to implement the derivation rule, computing any derived values as necessary when it is called. Inserts, updates or deletes to the data used in the derivation rule can then be routed through the code in the stored procedure. It's important to ensure that there is no "back door" route that would allow this code to be bypassed. Stored procedures are discussed in more detail in Section 13.5 of the book.

Table E.2 Lazy evaluation of derivation rules

Technique	*Features*
View	The derivation rule can be incorporated into the **select** statement defined for the view. Typically, a derived rule will become the basis for the definition of a view column, in which will appear values derived from data stored in base tables. Views are discussed in more detail in Section 13.3. of the book.
Common Table Expression	A Common Table Expression (CTE) can be used in a similar way to a view, the difference being that the derived value is only available locally to the code immediately following the CTE. More information on CTEs is given later in this appendix.
Generated Column	Some database systems (SQL Server being an example) allow the definition of table columns that contain values computed "on the fly". No data will be stored for the column, but when the table is queried the values will appear, having been computed by a suitable expression defined with the table. More information on generated columns is given later in this appendix.
Stored Procedure	A stored procedure could contain code to implement the derivation rule, computing any derived values as necessary when it is called. Queries involving the derived values can then be routed through the stored procedure. Stored procedures are discussed in more detail in Section 13.5 of the book.
User Defined Function (UDF)	A scalar UDF could be used to return a single scalar derived value or a table-valued UDF could be arranged to return a table containing derived values in a column. User defined functions are discussed in more detail in Section 13.5 of the book.

Common Table Expressions

A Common Table Expression (CTE) is effectively a query that produces a named result table—a concept not unlike a view. This query can incorporate a derivation rule to compute derived values that will appear in a column of the result table (a column that is not present in the base table). However, the named result table is only available to the statement that immediately follows and so the derived values must be used directly, before the result table disappears.

As an example, consider the following schema.

Sale (itemNr, unitPrice, quantity, taxRate)

We can derive the taxed sale price for an item as (unitPrice * quantity * taxRate). Assume that we want to write a query that involves itemId, unitPrice, and the sales price with tax. We could do this using a CTE as follows.

```
with TaxedSale as
(select itemId, unitPrice,
      (unitPrice * quantity * taxRate) as  priceWithTax
      from Sale)
select … from TaxedSale …
```

Here "TaxedSale" is defined as a CTE that effectively produces a temporary table containing derived values in the priceWithTax column.

Generated columns

A generated column is one whose values are determined by evaluation of a scalar generation expression. The generation expression can reference base columns of the base table to which it belongs but cannot otherwise access SQL data. When a new row is inserted into a table with a generated column, the associated expression is evaluated and the result becomes the value of the column. Here is an example of a table created with a generated column.

```
create table Employee (
  empId int not null primary key,
  salary decimal(7,2) not null,
  bonus decimal(7,2) not null,
  totalPay generated always as (salary + bonus) )
```

The following statement inserts a new employee into the table.

```
insert into Employee (empId, salary, bonus) values (123, 50000.00, 5000.00)
```

A new row will be added including the value of totalPay (which will have been computed as 50 000.00 + 5 000.00 = 55 000.00). Note that the generated column did not appear in the column list. If a generated column is named in the column list its corresponding value must be specified as the keyword **default**, and nothing else. If a column referenced in a generated column expression is updated, the corresponding generated value is also updated automatically.

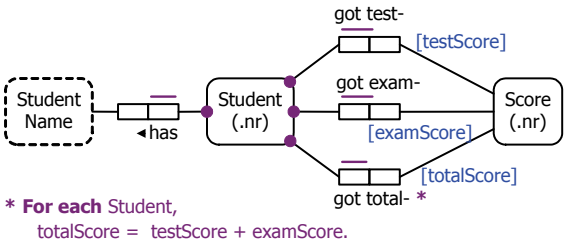
As always, there are likely to be variations between implementations. SQL Server has a slightly different syntax from the standard, using only **as** instead of **generated always as** to introduce the computed column. Also in SQL Server, the default action for such columns is to compute the value on request, rather than store the value in the table. If the value is to be stored, then the keyword **persisted** must be added. The SQL Server equivalent to the table definition above would therefore look like the following.

```
create table Employee (
  empId int not null primary key,
  salary decimal(7,2) not null,
  bonus decimal(7,2) not null,
  totalPay as (salary + bonus) persisted )
```

Generated columns have some restrictions on how they may be used (as keys, for indexing, and so on), but they provide a fairly straightforward way of implementing derivation rules, providing that any values required for the computation of the derived value can all be found in the same table.

Exercise E1

1. The following conceptual model contains a derivation rule.

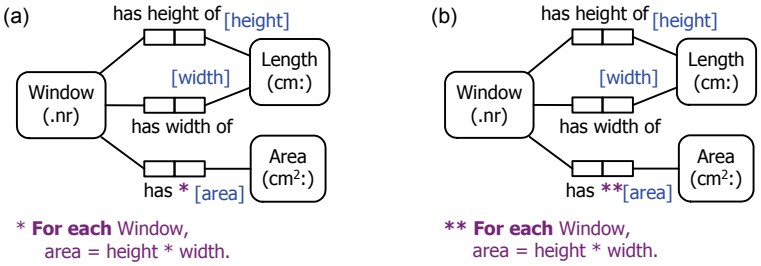


Assume that the following table has been created.

```
create table Student (
 studentNr int not null primary key,
 studentName varchar(50) not null,
 testScore int not null,
 examScore int not null )
```

(a) Implement the derivation rule using a view.
(b) Use a common table expression to list all students with a total score of less than 50.
(c) Define an alternative create table statement that uses a generated column.

2. The following two conceptual models differ in their treatment of a derivation rule.



(a) Implement the derivation rule in (a) using a view.
(b) Implement the derivation rule in (b) using a trigger.