

**DB-Main Manual Series**

---

# **CODE GENERATION FOR ORACLE 8**

**VERSION 6.5 - MARCH 2002**



---

**The University of Namur - LIBD**



---

**PART I      TECHNICAL ELEMENTS..... 5**

---

CHAPTER 1	SQL TECHNIQUES.....	9
1.1	Comments.....	9
1.2	Predicates of domain, column and table.....	9
1.3	Views with check option.....	10
1.4	Stored procedures.....	10
1.5	Triggers.....	10
CHAPTER 2	INTEGRITY CONSTRAINTS.....	11
2.1	Constraints on columns.....	11
2.1.1	Stable attribute.....	11
2.1.2	Domain.....	12
2.1.3	Default value.....	13
2.2	Constraints on groups.....	14
2.2.1	Primary identifier.....	14
2.2.2	Secondary identifier.....	15
2.2.3	Foreign key.....	15
2.2.4	Inclusion.....	16
2.2.5	Local existence.....	16
2.2.6	Maximum cardinality.....	18
CHAPTER 3	CODE FOR ORACLE 8.....	19
3.1	Syntax.....	19
3.1.1	Comments.....	19
3.1.2	Native techniques.....	19
3.1.3	Checks.....	19
3.1.4	Views with check option.....	19
3.1.5	Triggers.....	20
3.1.6	Stored procedures.....	20
3.2	Integrity constraints.....	23
3.2.1	Stable attribute.....	24
3.2.2	Domain.....	24
3.2.3	Default value.....	26
3.2.4	Primary identifier.....	28
3.2.5	Secondary identifier.....	29
3.2.6	Foreign key.....	30
3.2.7	Inclusion constraints.....	31
3.2.8	Local existence.....	33
3.2.9	Maximum cardinality.....	38

---

**PART II THE ASSISTANTS .....41**

---

CHAPTER 4	PHYSICAL DESIGN ASSISTANT FOR ORACLE 8 .....	43
4.1	Levels.....	43
4.2	Default settings .....	44
4.3	The physical design assistant.....	44
4.3.1	Remove settings.....	45
4.3.2	Global settings .....	45
4.3.3	Default settings (current schema) .....	47
4.3.4	Local settings (current schema).....	47
CHAPTER 5	PARAMETRIC SQL GENERATOR FOR ORACLE 8 .....	51
5.1	The code generator assistant.....	53
5.1.1	Default generation .....	53
5.1.2	Change output files.....	54
5.1.3	Change constraints to generate .....	55

# **PART I**

## **TECHNICAL ELEMENTS**



The generation of code of creation of an Oracle 8 database and management of the integrity constraints related to it is possible within DB-MAIN via two assistants : an assistant of relational physical design for Oracle and a parametric SQL generator. The assistant of relational physical design sets the options for a future generation of code for Oracle 8 : the technique of implementation for each type of constraint (or even for each particular constraint) and the creation of index. The parametric SQL generator sets the option of generation itself : output files, which will contain the code, and constraints to generate.

This parametric generation is thus not only a translation of the totality of the constraints in the DDL (Data Definition Language) of Oracle, which is unable to translate a certain number of complex constraints, but to express these constraints according to various techniques, which will give additional procedural fragments. The various constraints of the physical schema could thus be coded according to six different techniques : the comments (which are not really a technique of implementation, but which can be used to announce the existence of a constraint), native techniques, checks, views with check option, triggers, and stored procedures.

This document is structured as follows : in this first part, we examine the technical elements necessary to the generation. In a first chapter, we present the mechanisms offered by the RDBMS (Relational database management systems) to manage the integrity constraints. The second chapter will describe the integrity constraints contained in a physical schema of a relational database, their consequence on the actions on data, and the way in which they are expressed in DB-MAIN. Finally, in the third chapter, we show the code generated for the constraints described in the second chapter. The second part of this document present the two assistants of DB-MAIN.



# Chapter 1

## SQL techniques

The relational DBMS offer, via SQL language, declaratory and generic mechanisms making it possible to define relational schemas of a great richness of representation. *Declared constructs*, called so native techniques, are very few :

1. **tables**;
2. **domains**;
3. **columns**, their type and their default value;
4. **mandatory columns** (not null);
5. primary (primary key) and secondary (unique or unique index) **identifiers**;
6. **foreign keys** (foreign key) and their strategy of reaction to the violation of the referential integrity (delete mode and update mode).

The *generic mechanisms* permit to program the integrity constraints. These mechanisms are : comments, predicates, views with check option, triggers and stored procedures.

### 1.1 Comments

A comment is of course not a mechanism to manage an integrity constraint. When this "technique" is used, the integrity constraints are present in the code in the form of comments containing directives for the programmer or the user. It is of course the simplest technique "to program", but it does not bring any security on the validation of the data, and is thus to use only if the programmer is assured that the data will be always valid.

### 1.2 Predicates of domain, column and table

A predicate is a logical expression associated with a domain, a column, a table or a database. In this last case, the expression defines an assertion, which is in general not implemented by the editors of DBMS, and which we will be unaware of. The predicate appears as a traditional SQL condition, such as one would find it in the "where" clause of a select, delete or update query. The data must check this condition at any moment. More precisely, this condition will be evaluated after each modification of the object to which the predicate is attached or at the end of each transaction :

- *predicate of domain* : checked after creation of a line or modification of the value of a column defined on this domain;
- *predicate of column* : checked after creation of a line or modification of the value of the column;

- *predicate of table* : checked after creation or modification of a line of the table.

There are local and global predicates. A *local predicate* is evaluated for a line without reference to the other lines of the table or another table. A *global predicate* references during the evaluation other lines that the line on which it is defined. Rare are the DBMS which deals with the global predicates. A predicate, whatever it is local or global, supervises only the modifications of the table or the column with which it is associated.

### 1.3 Views with check option

A view authorizing the updates, and thus defined by a single table request, constitutes, via the `where` clause, a filter which selects the lines to be delivered. If the definition of the view includes the `with check option` clause, this filter is also active at the time of the updates. This clause doesn't allow a query to add lines or modify lines so that they wouldn't appear in the view.

### 1.4 Stored procedures

A stored procedure is a sequence of precompiled instructions whose execution can be required by a user, an application program, a trigger or another stored procedure. Stored in the database, it can be considered as a single and common resource for all the applications, thus avoiding a duplication of code. It makes it possible to define complex processing, in particular with regard to the integrity or the dynamic behavior of the data. There doesn't exist yet any standard of commonly accepted procedural language, each DBMS proposing its own language.

### 1.5 Triggers

A trigger is a procedural mechanism attached with a table and made up of a section of code accompanied by the conditions which lead to its execution. Its general form (event-condition-action or ECA) is the following :

```
before/after E
when C
begin
  A
end
```

and is interpreted in the following way : *if an event **E** (insert, delete, update) occurs, and if the condition **C** is satisfied, then action **A** is executed either before or after the event **E**.*

Just like for the stored procedures, there does not exist yet any common procedural language for the triggers. However, all the triggers have a common general structure.

## Chapter 2

# Integrity constraints

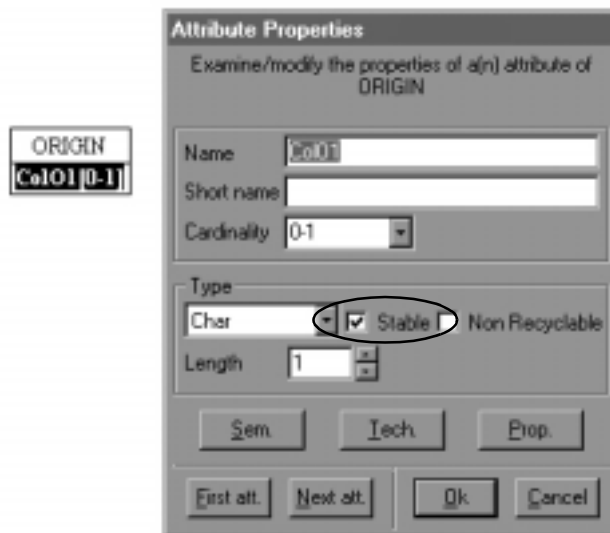
In this chapter, we describe systematically the constraints of integrity taken into account by the generator. After a short definition, the operations presenting a risk of violation of the constraint are presented, as well as the reactions following a violation of constraint. The code generated by the generator will be the implementation of the rules defined in this chapter.

We take into account as well the constraints relating to columns as the constraints relating to groups. The first ones are the domain constraints, the stable attributes, and the default values. The second ones are the identifiers (primary and secondary), the foreign keys, the inclusion constraints, the constraints of local existence, and the constraints of maximum cardinality.

## 2.1 Constraints on columns

### 2.1.1 Stable attribute

**Definition 2.1** *The value of column Col01 of table ORIGIN must not be changed after it has received a value.*



**Figure 2.1** - Representation of a stable attribute in DB-MAIN

Event	Condition and Action
Update of ColO1	old.ColO1 not null $\Rightarrow$ abort

**Table 2.1** - Stable attribute

### 2.1.2 Domain

**Definition 2.2** *The values of column ColO of table ORIGIN belong to a list of values or to an interval.*

Since version 5 of DB-MAIN, constraints of domain can be represented via a meta-property called "Value constraint". In that meta-property, the property "value" is inserted :

```
#value=
value1
value2
...
valuen
#
```

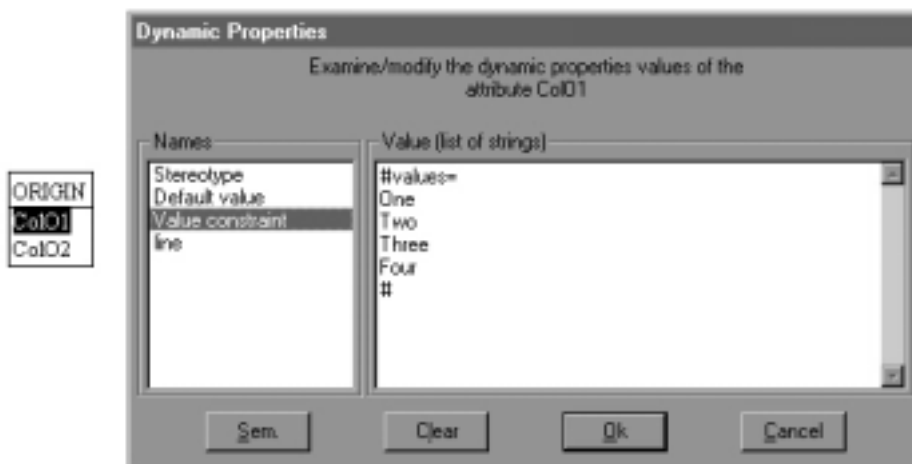
That means that the value of the column must be value1 or value2 or ... or valuen.

Likewise, the user can insert a property "interval" :

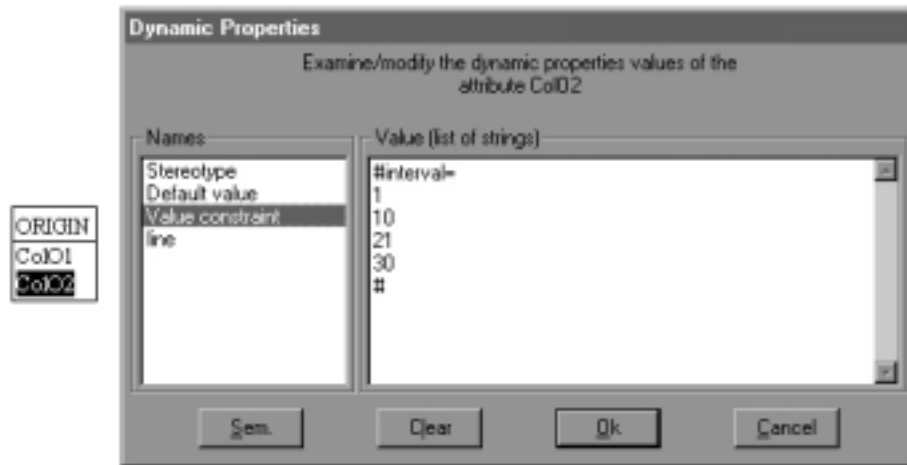
```
#interval=
value1
value2
...
valuen
#
```

In an interval, n must be an even number. Indeed, during generation, the values contained in the textual property "interval" will be taken two by two. So, the value of the column must be included between value1 and value2 or between value3 and value4, etc. An interval can only be defined on integers.

The values given by the user must correspond to the type of the corresponding column.



**Figure 2.2** - Representation of a constraint of domain (list of values) in DB-MAIN (the column ColO1 is of an alphanumeric type).



**Figure 2.3** - Representation of a constraint of domain (list of intervals) in DB-MAIN (the column ColO1 is of a numeric type).

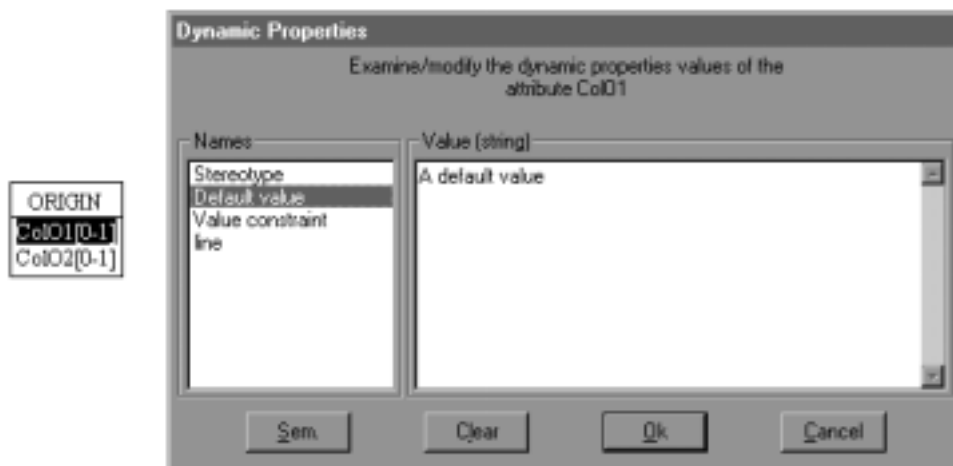
Event	Condition and Action
Insert into Origin	new.ColO1 (ColO2) does not belong to domain ⇒ abort
Update of ColO1 (ColO2)	new.ColO1 (ColO2) does not belong to domain ⇒ abort

**Table 2.2** - Domain

### 2.1.3 Default value

**Definition 2.3** *If there is no value (null) in a column ColO in table Origin, it is given a default value specified by the user.*

Since the version 5 of DB-MAIN, default value constraints can be represented via a meta-property called "Default value", which contains the default value chosen by the user. Every column can have a default value, but that one will be used only if the column is given no value. The value given by the user must correspond to the type of the column.



**Figure 2.4** - Representation of default value in DB-MAIN (column ColO1 is of alphanumeric type).

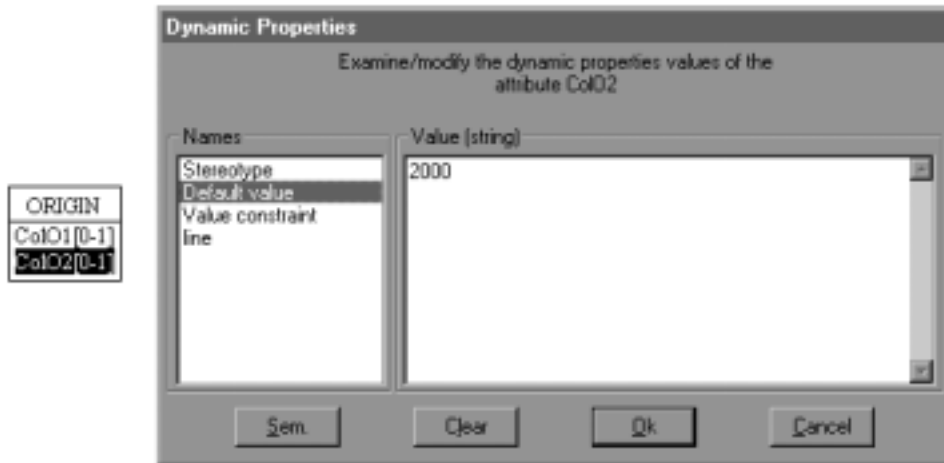


Figure 2.5 - Representation of default value in DB-MAIN (column ColO2 is of numeric type).

Event	Condition and Action
Insert into Origin	new.ColO1 (ColO2) is null $\Rightarrow$ new.ColO1 (ColO2) = <Default value>
Update of ColO1	new.ColO1 (ColO2) is null $\Rightarrow$ new.ColO1 (ColO2) = <Default value>

Table 2.3 - Default value

## 2.2 Constraints on groups

### 2.2.1 Primary identifier

**Definition 2.4 :** The number of lines of table Origin having a same value of the columns composing the identifier GrO (ColO1) must be lower or equal to one

ORIGIN
<b>ColO1</b>
ColO2
<b>id:ColO1</b>

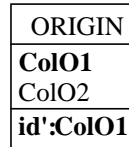
Figure 2.6 - Representation of a primary identifier in DB-MAIN

Event	Condition and Action
Insert into Origin	number of lines having the same value as new.GrO1 before insert = 1 $\Rightarrow$ abort
Update of GrO	number of lines having the same value as new.GrO1 before update = 1 $\Rightarrow$ abort

Table 2.4 - Primary identifier

### 2.2.2 Secondary identifier

**Definition 2.5** *The number of lines of table Origin having a same value of the columns composing the identifier GrO (ColO1) must be lower or equal to one.*



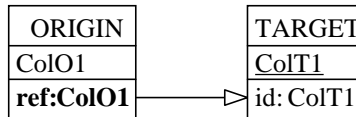
**Figure 2.7** - Representation of a secondary identifier in DB-MAIN

Instruction	Reaction
Insert into Origin	number of lines having the same value as new.GrO1 before insert = 1 $\Rightarrow$ abort
Update of GrO	number of lines having the same value as new.GrO1 before update = 1 $\Rightarrow$ abort

**Table 2.5** - Secondary identifier

### 2.2.3 Foreign key

**Definition 2.6** *Each instance of the foreign key GrO (ColO1) of table Origin must be an instance of the identifier GrT (ColT1) of table Target.*



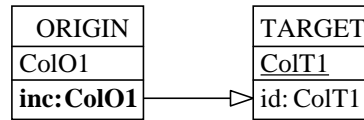
**Figure 2.8** - Representation of a foreign key in DB-MAIN

Event	Condition and Action
Insert into Origin	The new value of GrO does not appear like value of identifier GrT $\Rightarrow$ abort $\vee$ insert into Target (null)
Update of GrO	The new value of GrO does not appear like value of identifier GrT $\Rightarrow$ abort $\vee$ insert into Target (null)
Update of GrT	The old value of GrT was referenced by GrO $\Rightarrow$ abort (restrict) $\vee$ update of GrO (cascade) $\vee$ update of GrO (set null)
Delete from Target	The old value of GrT was referenced by GrO $\Rightarrow$ abort (restrict) $\vee$ delete from Origin (cascade) $\vee$ update of GrO (set null)

**Table 2.6** - Foreign key

## 2.2.4 Inclusion

**Definition 2.7** Each instance of the group  $GrO$  ( $ColO1$ ) of table *Origin* must be an instance of the group  $GrT$  ( $ColT1$ ) of table *Target*.



**Figure 2.9** - Representation of an inclusion constraint in DB-MAIN

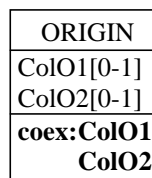
Event	Condition and Action
Insert into Origin	The new value of $GrO$ does not appear like value of identifier $GrT \Rightarrow$ abort $\vee$ insert into Target (null)
Update of $GrO$	The new value of $GrO$ does not appear like value of identifier $GrT \Rightarrow$ abort $\vee$ insert into Target (null)
Update of $GrT$	The old value of $GrT$ was referenced by $GrO \Rightarrow$ abort (restrict) $\vee$ update of $GrO$ (cascade) $\vee$ update of $GrO$ (set null)
Delete from Target	The old value of $GrT$ was referenced by $GrO \Rightarrow$ abort (restrict) $\vee$ delete from Origin (cascade) $\vee$ update of $GrO$ (set null)

**Table 2.7** - Inclusion constraint

## 2.2.5 Local existence

### 2.2.5.1 Coexistence

**Definition 2.8** The components of the group  $GrO$  ( $ColO1$ ,  $ColO2$ ) must be simultaneously present or absent for any instance of the table *Origin*.



**Figure 2.10** - Representation of a coexistence constraint in DB-MAIN

Event	Condition and Action
Insert into Origin	$(\text{new.ColO1 null} \wedge \text{new.ColO2 not null}) \vee (\text{new.ColO1 not null} \wedge \text{new.ColO2 null}) \Rightarrow$ abort
Update of $ColO1$ , $ColO2$	$(\text{new.ColO1 null} \wedge \text{new.ColO2 not null}) \vee (\text{new.ColO1 not null} \wedge \text{new.ColO2 null}) \Rightarrow$ abort

**Table 2.8** - Coexistence constraint

### 2.2.5.2 Exclusivity

**Definition 2.9** Among the components of the group *GrO* (*ColO1*, *ColO2*), at most one must be present for any instance of the table *Origin*.

ORIGIN
ColO1[0-1]
ColO2[0-1]
<b>excl:ColO1</b> <b>ColO2</b>

**Figure 2.11** - Representation of an exclusivity constraint in DB-MAIN

Event	Condition and Action
Insert into Origin	$\text{new.ColO1 not null} \wedge \text{new.ColO2 not null} \Rightarrow \text{abort}$
Update of ColO1, ColO2	$\text{new.ColO1 not null} \wedge \text{new.ColO2 not null} \Rightarrow \text{abort}$

**Table 2.9** - Exclusivity constraint

### 2.2.5.3 At-least-one

**Definition 2.10** Among the components of the group *GrO* (*ColO1*, *ColO2*), at least one must be present for any instance of the table *Origin*.

ORIGIN
ColO1[0-1]
ColO2[0-1]
<b>at-1st-1:ColO1</b> <b>ColO2</b>

**Figure 2.12** - Representation of an at-least-one constraint in DB-MAIN

Event	Condition and Action
Insert into Origin	$\text{new.ColO1 null} \wedge \text{new.ColO2 null} \Rightarrow \text{abort}$
Update of ColO1, ColO2	$\text{new.ColO1 null} \wedge \text{new.ColO2 null} \Rightarrow \text{abort}$

**Table 2.10** - At-least-one constraint

### 2.2.5.4 Exactly-one

**Definition 2.11** Among the components of the group GrO (ColO1, ColO2), exactly one must be present for any instance of the table ORIGIN.

ORIGIN
ColO1[0-1] ColO2[0-1]
<b>exact-1:ColO1</b> <b>ColO2</b>

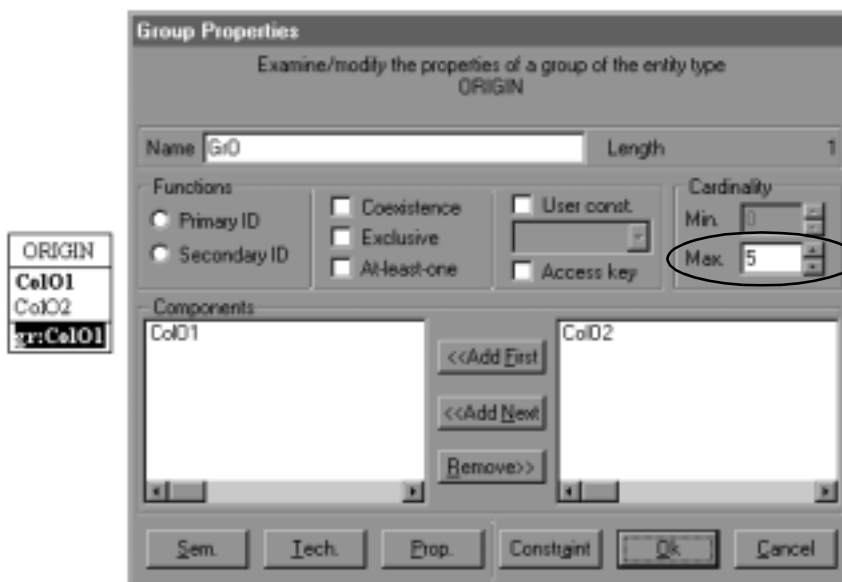
**Figure 2.13** - Representation of an exactly-one constraint in DB-MAIN

Event	Condition and Action
Insert into Origin	$(\text{new.ColO1 null} \wedge \text{new.ColO2 null}) \vee (\text{new.ColO1 not null} \wedge \text{new.ColO2 not null}) \Rightarrow \text{abort}$
Update of ColO1, ColO2	$(\text{new.ColO1 null} \wedge \text{new.ColO2 null}) \vee (\text{new.ColO1 not null} \wedge \text{new.ColO2 not null}) \Rightarrow \text{abort}$

**Table 2.11** - Exactly-one constraint

### 2.2.6 Maximum cardinality

**Definition 2.12** The number of lines of table ORIGIN having a same value for the columns composing the group GrO (ColO1) must be less or equal to an integer Max, fixed by the analyst.



**Figure 2.14** - Representation of a maximum cardinality constraint in DB-MAIN

Event	Condition and Action
Insert into Origin	Number of lines having same values as new.ColO1 before insert = Max $\Rightarrow$ abort
Update of GrO	Number of lines having same values as new.ColO1 before update = Max $\Rightarrow$ abort

**Table 2.12** - Maximum cardinality

## Chapter 3

# Code for Oracle 8

### 3.1 Syntax

#### 3.1.1 Comments

Comments are free text, each line preceded by the string "--", which indicates a line of comment in Oracle.

#### 3.1.2 Native techniques

For each constraint, that type of code will be generated :

```
alter table <table name> add constraint <constraint name>
    <constraint>;
```

#### 3.1.3 Checks

Predicates in Oracle 8 are limited to local checks, i.e. defined on a table and reacting to modifications made on columns composing the constraint. The constraint must not reference other elements than the current line.

For each constraint, that type of code will be generated :

```
alter table <table name> add constraint <constraint name>
    check (<constraint>;
```

#### 3.1.4 Views with check option

There is at most one view with check option per table. If there are several constraints on the same table, each constraint will be in the condition of selection (*where*), each constraint being related by a logical and. If the queries are too complex, Oracle does not matter of the view, without raising any error. Likewise, we don't know when a constraint becomes too complex. The code of a view with check option looks like this :

```

create or replace view <V_table name> as
select * from <table name>
where (<condition 1>)
and (<condition 2>)
and (...)
and (<condition n>)
with check option;

```

### 3.1.5 Triggers

The action of the trigger checks the constraint, and, if it is violated, raises an exception predefined in Oracle. That exception must be trapped and processed by the user. If not, Oracle raises an error. For each constraint, a trigger like the following one will be generated :

```

create or replace trigger <trigger name>
<event>
[for each row]
[when <condition>]
begin
<action>
end;

```

### 3.1.6 Stored procedures

To check the integrity constraints, stored procedures call test procedures (one or more per constraint), which can raise exceptions (one or more per constraint). Those test procedures and exceptions are declared in two packages. The test procedures could be used directly by a user who would deal with the checking of integrity of his/her database.

```

create or replace package dbm_exceptions is

exception1 exception;
...
exceptionn exception;

end dbm_exceptions;

create or replace package dbm_tests is

procedure test1[(param1 in param1type, ..., paramn in paramntype)];
...
procedure testn[(param1 in param1type, ..., paramn in paramntype)];

end dbm_tests;

create or replace package body dbm_tests is

procedure test1[(param1 in param1type, ..., paramn in paramntype)] is
[variables]
begin
...
raise exception1;
...
end test1;

...

```

```

procedure testn[(param1 in param1type, ..., paramn in paramntype)] is
[variables]
begin
  ...
  raise exceptionn;
  ...
end testn;

end dbm_tests;

```

The generation of stored procedures consist in the generation of a package containing the procedures of insert, update and delete (three procedures per table).

```

create or replace package dbm_procedures is

procedure insert_table1(new_table1 in out table1%rowtype);
procedure update_table1(new_table1 in out table1%rowtype, old_table1 in
out table1%rowtype);
procedure delete_table1(old_table1 in out table1%rowtype);
...
procedure insert_tablen(new_tablen in out tablen%rowtype);
procedure update_tablen(new_tablen in out tablen%rowtype, old_tablen in
out tablen%rowtype);
procedure delete_tablen(old_tablen in out tablen%rowtype);

end dbm_procedures;

```

The parameters of the procedures are variables using a type definition, %rowtype, derived from the definition of a table. Column components of a row are accessed using the dot notation. The variable new\_table1 has all the components that have been defined in the table table1. The insert procedures have one parameter, which contain the values of the inserted line. The update procedures have two parameters, the first one containing the values of a line before modification, the second one containing the new values of that line. The delete procedures have one parameter, containing the values of the line to delete.

Those stored procedures can be executed either interactively in Oracle (execute PACKAGE\_NAME.PROCEDURE\_NAME(parameters)), either in an application program (in this case, the way of call depends on the language).

### 3.1.6.1 Insert

The first step is to call the test procedures to check all the integrity constraints defined on the current table. If the test procedures don't raise any exception, the line can be inserted in the table.

```

procedure insert_TABLE_NAME (new_TABLE_NAME in out TABLE_NAME%rowtype)
is
begin
-- Tests

<tests>

-- Insert
insert into TABLE_NAME values (new_TABLE_NAME.COL1, new_TABLE_NAME.COL2,
..., new_TABLE_NAME.COLn);

end insert_TABLE_NAME;

```

### 3.1.6.2 Update

In the update procedure, a test is done on a column (or a group of columns) containing a constraint only if the value of that column (or group of columns) has changed.

```

procedure update_TABLE_NAME(old_TABLE_NAME in TABLE_NAME%rowtype,
new_TABLE_NAME in out TABLE_NAME%rowtype) is
begin
  -- Tests

  if ((new_TABLE_NAME.COL1 <> old_TABLE_NAME.COL1)
or (new_TABLE_NAME.COL1 is null and old_TABLE_NAME.COL1 is not null)
or (new_TABLE_NAME.COL1 is not null and old_TABLE_NAME.COL1 is null))
then

  <tests on COL1>

end if;

...

if ((new_TABLE_NAME.COLn <> old_TABLE_NAME.COLn)
or (new_TABLE_NAME.COLn is null and old_TABLE_NAME.COLn is not null)
or (new_TABLE_NAME.COLn is not null and old_TABLE_NAME.COLn is null))
then

  <tests on COLn>

end if;

-- Update

update TABLE_NAME set COL1 = new_TABLE_NAME.COL1,
COL2 = new_TABLE_NAME.COL2,
...
COLn = new_TABLE_NAME.COLn
where COL1 = old_TABLE_NAME.COL1
and COL2 = old_TABLE_NAME.COL2
...
and COLn = old_TABLE_NAME.COLn;

end update_TABLE_NAME;

```

If the table TABLE\_NAME has an identifier (e.g. (COL1, COL2)), the where clause is more simple :

```

-- Update

update TABLE_NAME set COL1 = new_TABLE_NAME.COL1,
COL2 = new_TABLE_NAME.COL2,
...
COLn = new_TABLE_NAME.COLn
where COL1 = old_TABLE_NAME.COL1
and COL2= old_TABLE_NAME.COL2;

end update_TABLE_NAME;

```

### 3.1.6.3 Delete

The first step is to call the test procedures to check all the integrity constraints defined on the current table. If the test procedures don't raise any exception, the line can be deleted from the table.

```

procedure delete_TABLE_NAME (old_TABLE_NAME in TABLE_NAME%rowtype) is
begin
  -- Tests

  <tests>

  -- Delete

  delete
  from TABLE_NAME
  where COL1 = old_TABLE_NAME.COL1
  and COL2 = old_TABLE_NAME.COL2
  ...
  and COLn = old_TABLE_NAME.COLn;

end delete_TABLE_NAME;

```

If the table TABLE\_NAME has an identifier (e.g. (COL1, COL2)), the where clause is more simple :

```

-- Delete

delete
from TABLE_NAME
where COL1 = old_TABLE_NAME.COL1
and COL2 = old_TABLE_NAME.COL2;

end delete_TABLE_NAME;

```

## 3.2 Integrity constraints

Implementation techniques given by default to each type of constraint for Oracle 8 are :

- **native** for the identifiers (primary and secondary) and the foreign keys;
- **check** for the domains and constraints of local existence;
- **trigger** for default values, stable attributes, inclusion constraints and maximum cardinality.

Not every technique can implement every constraint. The following table shows the different possibilities of generation.

Constraint	Comment	Native	Check	View	Trigger	Procedure
Stable attribute	X				X	X
Domain	X		X	X	X	X
Default value	X				X	X
Primary identifier	X	X			X	X
Secondary identifier	X	X			X	X
Foreign key	X	X			X	X
Inclusion	X				X	X
Local existence	X		X	X	X	X
Maximal cardinality	X				X	X

**Table 3.1** - Constraints and techniques of implementation

The following code has been generated on the schemas described in the previous chapter.

### 3.2.1 Stable attribute

#### 3.2.1.1 Comments

```
-- Col01 in ORIGIN : stable attribute
```

#### 3.2.1.2 Triggers

```
create or replace trigger T_ORI_Col_STABLE_AUR
after update of Col01 on ORIGIN
for each row
when (new.Col01 <> old.Col01 and old.Col01 is not null)
begin
raise_application_error(-20003,'Violation of stable attribute constraint') ;
end;
```

#### 3.2.1.3 Stored procedures

In the package DBM\_TESTS :

```
procedure test_ORIGIN_Col01_STABLE(v_Col01 in ORIGIN.Col01%type) is
begin
if v_Col01 is not null then
raise dbm_exceptions.ORIGIN_Col01_STABLE;
end if;
end test_ORIGIN_Col01_STABLE;
```

In the procedure UPDATE\_ORIGIN :

```
-- Tests

if ((new_Col01 <> old_Col01) or (new_Col01 is null and old_Col01 is not
null) or (new_Col01 is not null and old_Col01 is null)) then
dbm_tests.test_ORIGIN_Col01_STABLE(old_Col01);
end if;
```

### 3.2.2 Domain

#### 3.2.2.1 Comments

```
-- Col01 in ORIGIN : domain
-- Col01 in ('One','Two','Three','Four')
-- Col02 in ORIGIN : domain
-- (Col02 between 1 and 10) or (Col02 between 21 and 30)
```

#### 3.2.2.2 Checks

```
-- Domain (Col01)
alter table ORIGIN add constraint C_ORIGIN_Col01_DOM
check (Col01 in ('One','Two','Three','Four'));
```

```
-- Domain (Col02)
alter table ORIGIN add constraint C_ORIGIN_Col02_DOM
check ((Col02 between 1 and 10) or (Col02 between 21 and 30));
```

### 3.2.2.3 Views with check option

```
create or replace view V_ORIGIN as
select * from ORIGIN
where (Col01 in ('One', 'Two', 'Three', 'Four'))
and ((Col02 between 1 and 10) or (Col02 between 21 and 30))
with check option;
```

### 3.2.2.4 Triggers

```
create or replace trigger T_ORI_Col_DOM_AIUR
after insert or update of Col01 on ORIGIN
for each row
when (new.Col01 not in ('One', 'Two', 'Three', 'Four'))
begin
    raise_application_error(-20002, 'Violation of domain constraint') ;
end;
```

```
create or replace trigger T_ORI_Col_DOM_AIUR
after insert or update of Col02 on ORIGIN
for each row
when (not ((new.Col02 between 1 and 10) or (new.Col02 between 21 and
30)))
begin
    raise_application_error(-20002, 'Violation of domain constraint') ;
end;
```

### 3.2.2.5 Stored procedures

In the package DBM\_TESTS :

```
procedure test_ORIGIN_Col01_DOM(v_Col01 in ORIGIN.Col01%type) is
begin
if not (v_Col01 in ('One', 'Two', 'Three', 'Four')) then
    raise dbm_exceptions.ORIGIN_Col01_DOM;
end if;
end test_ORIGIN_Col01_DOM;

procedure test_ORIGIN_Col02_DOM(v_Col02 in ORIGIN.Col02%type) is
begin
if not ((v_Col02 between 1 and 10) or (v_Col02 between 21 and 30)) then
    raise dbm_exceptions.ORIGIN_Col02_DOM;
end if;
end test_ORIGIN_Col02_DOM;
```

In the procedure INSERT\_ORIGIN :

```
-- Tests

dbm_tests.test_ORIGIN_Col01_DOM(v_Col01);
dbm_tests.test_ORIGIN_Col02_DOM(v_Col02);
```

In the procedure UPDATE\_ORIGIN :

```
-- Tests

if (new_Col01 <> old_Col01) or (new_Col01 is null and old_Col01 is not
null) or (new_Col01 is not null and old_Col01 is null)) then
  dbm_tests.test_ORIGIN_Col01_DOM(new_Col01);
end if;
if ((new_Col02 <> old_Col02) or (new_Col02 is null and old_Col02 is not
null) or (new_Col02 is not null and old_Col02 is null)) then
  dbm_tests.test_ORIGIN_Col02_DOM(new_Col02);
end if;
```

### 3.2.3 Default value

#### 3.2.3.1 Comments

```
-- Col01 in ORIGIN : default value
-- A default value
-- Col02 in ORIGIN : default value
-- 2000
```

#### 3.2.3.2 Native techniques

Oracle 8 permits the declaration of a default value for a column at the creation of a table or at the modification of that table. The corresponding SQL instruction would be :

```
alter table TABLE_NAME modify COLUMN_NAME default VALUE;
```

However, the default value is attributed only if one inserts no value in the corresponding value, and not if one inserts a null value. Thus, the following instruction applied on a table with three columns of which the last one has a default value, will update the last column with the default value. Indeed, that column having not received any value, the default value is given to it :

```
insert into TABLE_NAME (COLUMN1_NAME, COLUMN2_NAME)
values (VALUE1, VALUE2);
```

Otherwise, the insert of a null value does not lead to the update with the default value :

```
insert into TABLE_NAME
values (VALUE1, VALUE2, null);
```

For that reason, we did not implement that technique.

### 3.2.3.3 Triggers

```
create or replace trigger T_ORI_Col_DEF_BIUR
before insert or update of Col01 on ORIGIN
for each row
when (new.Col01 is null)
begin
    :new.Col01:= 'A default value' ;
end;
```

```
create or replace trigger T_ORI_Col_DEF_BIUR
before insert or update of Col02 on ORIGIN
for each row
when (new.Col02 is null)
begin
    :new.Col02:= 2000 ;
end;
```

### 3.2.3.4 Stored procedures

Unlike the other constraints, there is no exception here, because test procedures don't raise exceptions, but update the parameter in entry.

In the package DBM\_TESTS :

```
procedure test_ORIGIN_Col01_DEF(v_Col01 in out ORIGIN.Col01%type) is
begin
    if v_Col01 is null then
        v_Col01:= 'A default value';
    end if;
end test_ORIGIN_Col01_DEF;
procedure test_ORIGIN_Col02_DEF(v_Col02 in out ORIGIN.Col02%type) is
begin
    if v_Col02 is null then
        v_Col02:= 2000;
    end if;
end test_ORIGIN_Col02_DEF;
```

In the procedure INSERT\_ORIGIN :

```
-- Tests

dbm_tests.test_ORIGIN_Col01_DEF(v_Col01);
dbm_tests.test_ORIGIN_Col02_DEF(v_Col02);
```

In the procedure UPDATE\_ORIGIN :

```
if ((new_Col01 <> old_Col01) or (new_Col01 is null and old_Col01 is not
null) or (new_Col01 is not null and old_Col01 is null)) then
dbm_tests.test_ORIGIN_Col01_DEF(new_Col01);
end if;
if ((new_Col02 <> old_Col02) or (new_Col02 is null and old_Col02 is not
null) or (new_Col02 is not null and old_Col02 is null)) then
dbm_tests.test_ORIGIN_Col02_DEF(new_Col02);
end if;
```

## 3.2.4 Primary identifier

### 3.2.4.1 Comments

```
-- GrO (Col01) in ORIGIN : primary identifier
```

### 3.2.4.2 Native techniques

```
-- Primary identifier (GrO)
alter table ORIGIN add constraint N_ORIGIN_GrO_PRIM
primary key (Col01);
```

### 3.2.4.3 Triggers

```
create or replace trigger T_ORI_GrO_PRIM_BIUR
before insert or update of Col01 on ORIGIN
for each row
declare
temp integer;
begin
select count(*) into temp from ORIGIN where ORIGIN.Col01 = new.Col01;
if temp = 1 then
    raise_application_error(-20005,'Violation of primary identifier cons-
traint') ;
end if;
end;
```

### 3.2.4.4 Stored procedures

In the package DBM\_TESTS :

```
procedure test_ORIGIN_GrO_PRIM(v_Col01 in ORIGIN.Col01%type) is
temp integer;
begin
select count (*) into temp from ORIGIN
where ORIGIN.Col01 = v_Col01;
if temp=1 then raise dbm_exceptions.ORIGIN_GrO_PRIM;
end if;
end test_ORIGIN_GrO_PRIM;
```

In the procedure INSERT\_ORIGIN :

```
-- Tests
dbm_tests.test_ORIGIN_GrO_PRIM(v_Col01);
```

In the procedure UPDATE\_ORIGIN :

```
-- Tests
if ((new_Col01 <> old_Col01) or (new_Col01 is null and old_Col01 is not
null) or (new_Col01 is not null and old_Col01 is null)) then
dbm_tests.test_ORIGIN_GrO_PRIM(new_Col01);
end if;
```

## 3.2.5 Secondary identifier

### 3.2.5.1 Comments

```
-- GrO (Col01) in ORIGIN : secondary identifier
```

### 3.2.5.2 Native techniques

```
-- Secondary identifier (GrO)
alter table ORIGIN add constraint N_ORIGIN_GrO_UNI
unique (Col01);
```

### 3.2.5.3 Triggers

```
create or replace trigger T_ORI_GrO_UNI_BIUR
before insert or update of Col01 on ORIGIN
for each row
declare
temp integer;
begin
select count(*) into temp from ORIGIN where ORIGIN.Col01 = new.Col01;
if temp = 1 then
raise_application_error(-20006,'Violation of secondary identifier cons-
traint') ;
end if;
end;
```

### 3.2.5.4 Stored procedures

In the package DBM\_TESTS :

```
procedure test_ORIGIN_GrO_UNI(v_Col01 in ORIGIN.Col01%type) is
temp integer;
begin
select count (*) into temp from ORIGIN
where ORIGIN.Col01 = v_Col01;
if temp=1 then raise dbm_exceptions.ORIGIN_GrO_UNI;
end if;
end test_ORIGIN_GrO_UNI;
```

In the procedure INSERT\_ORIGIN :

```
-- Tests

dbm_tests.test_ORIGIN_GrO_UNI(v_Col01);
```

In the procedure UPDATE\_ORIGIN :

```
-- Tests

if ((new_Col01 <> old_Col01) or (new_Col01 is null and old_Col01 is not
null) or (new_Col01 is not null and old_Col01 is null)) then
dbm_tests.test_ORIGIN_GrO_UNI(new_Col01);
end if;
```

### 3.2.6 Foreign key

For the moment, the code generated for the foreign keys only deals with the default mode of Oracle, i.e. "restrict", i.e. each operation violating an integrity constraint is aborted. There is no propagation of modifications.

#### 3.2.6.1 Comments

```
-- GrO (Col01) in ORIGIN : foreign key
-- references TARGET
```

#### 3.2.6.2 Native techniques

```
-- Foreign key (GrO)
alter table ORIGIN add constraint N_ORIGIN_GrO_REF_OR
foreign key (Col01)
references TARGET;
```

#### 3.2.6.3 Triggers

```
create or replace trigger T_ORI_GrO_REF_OR_AIUR
after insert or update of Col01 on ORIGIN
for each row
declare
temp integer;
begin
select count(*) into temp from TARGET where TARGET.ColT1 = new.Col01;
if temp =0 then
raise_application_error(-20008,'Violation of referential constraint') ;
end if;
end;
```

```
create or replace trigger T_CIB_ORI_GrT_REF_TAR_BUDR
before update of ColT1 or delete on TARGET
for each row
declare
temp integer;
begin
select count(*) into temp from ORIGIN where ORIGIN.ColT1 = old.Col01;
if temp >0 then
raise_application_error(-20008,'Violation of referential constraint') ;
end if;
end;
```

#### 3.2.6.4 Stored procedures

In the package DBM\_TESTS :

```
procedure test_TARGET_GrT_REF_TAR(v_ColT1 in TARGET.ColT1%type) is
temp integer;
begin
select count (*) into temp from ORIGIN
where ORIGIN.Col01 = v_ColT1;
if temp<>0 then raise dbm_exceptions.TARGET_GrT_REF_TAR;
end if;
end test_TARGET_GrT_REF_TAR;
```

```
procedure test_ORIGIN_GrO_REF_OR(v_ColO1 in ORIGIN.ColO1%type) is
temp integer;
begin
select count (*) into temp from TARGET
where TARGET.ColT1 = v_ColO1;
if temp=0 then raise dbm_exceptions.ORIGIN_GrO_REF_OR;
end if;
end test_ORIGIN_GrO_REF_OR;
```

In the procedure INSERT\_ORIGIN :

```
-- Tests

dbm_tests.test_ORIGIN_GrO_REF_OR(v_ColO1);
```

In the procedure UPDATE\_TARGET :

```
-- Tests

if ((new_ColT1 <> old_ColT1) or (new_ColT1 is null and old_ColT1 is not
null) or (new_ColT1 is not null and old_ColT1 is null)) then
dbm_tests.test_TARGET_GrT_REF_TAR(old_ColT1);
end if;
```

In the procedure UPDATE\_ORIGIN :

```
-- Tests

if ((new_ColO1 <> old_ColO1) or (new_ColO1 is null and old_ColO1 is not
null) or (new_ColO1 is not null and old_ColO1 is null)) then
dbm_tests.test_ORIGIN_GrO_REF_OR(new_ColO1);
end if;
```

In the procedure DELETE\_TARGET :

```
-- Tests

dbm_tests.test_TARGET_GrT_REF_TAR(v_ColT1);
```

## 3.2.7 Inclusion constraints

### 3.2.7.1 Comments

```
-- Inclusion :
-- GrO (ColO1) in ORIGIN
-- included in GrT (ColT1) in TARGET
```

### 3.2.7.2 Triggers

```

create or replace trigger T_ORI_GrO_INCL_OR_AIUR
after insert or update of Col01 on ORIGIN
for each row
declare
temp integer;
begin
select count(*) into temp from TARGET where TARGET.ColT1 = new.Col01;
if temp =0 then
raise_application_error(-20011,'Violation of inclusion constraint') ;
end if;
end;

```

```

create or replace trigger T_CIB_ORI_GrT_INCL_TAR_BU DR
before update of ColT1 or delete on TARGET
for each row
declare
temp integer;
begin
select count(*) into temp from ORIGIN where ORIGIN.ColT1 = old.Col01;
if temp >0 then
raise_application_error(-20011,'Violation of inclusion constraint') ;
end if;
end;

```

### 3.2.7.3 Stored procedures

In the package DBM\_TESTS :

```

procedure test_TARGET_GrT_INCL_TAR(v_ColT1 in TARGET.ColT1%type) is
temp integer;
begin
select count (*) into temp from ORIGIN
where ORIGIN.Col01 = v_ColT1;
if temp<>0 then raise dbm_exceptions.TARGET_GrT_INCL_TAR;
end if;
end test_TARGET_GrT_INCL_TAR;
procedure test_ORIGIN_GrO_INCL_OR(v_Col01 in ORIGIN.Col01%type) is
temp integer;
begin
select count (*) into temp from TARGET
where TARGET.ColT1 = v_Col01;
if temp=0 then raise dbm_exceptions.ORIGIN_GrO_INCL_OR;
end if;
end test_ORIGIN_GrO_INCL_OR;

```

In the procedure INSERT\_ORIGIN :

```

-- Tests

dbm_tests.test_ORIGIN_GrO_INCL_OR(v_Col01);

```

In the procedure UPDATE\_TARGET :

```
-- Tests

if ((new_ColT1 <> old_ColT1) or (new_ColT1 is null and old_ColT1 is not
null) or (new_ColT1 is not null and old_ColT1 is null)) then
dbm_tests.test_TARGET_GrT_INCL_TAR(old_ColT1);
end if;
```

In the procedure UPDATE\_ORIGIN :

```
-- Tests

if ((new_ColO1 <> old_ColO1) or (new_ColO1 is null and old_ColO1 is not
null) or (new_ColO1 is not null and old_ColO1 is null)) then
dbm_tests.test_ORIGIN_GrO_INCL_OR(new_ColO1);
end if;
```

In the procedure DELETE\_TARGET :

```
-- Tests

dbm_tests.test_TARGET_GrT_INCL_TAR(v_ColT1);
```

### 3.2.8 Local existence

#### 3.2.8.1 Coexistence

##### a) Comments

```
-- GrO (ColO1, ColO2) in ORIGIN : coexistence
```

##### b) Checks

```
-- Coexistence (GrO)
alter table ORIGIN add constraint C_ORIGIN_GrO_COEX
check ((ColO1 is null and ColO2 is null)
or (ColO1 is not null and ColO2 is not null));
```

##### c) Views with check option

```
create or replace view V_ORIGIN as
select * from ORIGIN
where ((ColO1 is null and ColO2 is null)
or (ColO1 is not null and ColO2 is not null))
with check option;
```

## d) Triggers

```

create or replace trigger T_ORI_GrO_COEX_AUIR
after insert or update of Col01, Col02 on ORIGIN
for each row
when (not((new.Col01 is null and new.Col02 is null) or (new.Col01 is not
null and new.Col02 is not null)))
begin
raise_application_error(-20004,'Violation of existence constraint') ;
end;
```

## e) Stored procedures

In the package DBM\_TESTS :

```

procedure test_ORIGIN_GrO_COEX(v_Col01 in ORIGIN.Col01%type, v_Col02 in
ORIGIN.Col02%type) is
begin
  if not ((v_Col01 is null and v_Col02 is null)
or (v_Col01 is not null and v_Col02 is not null)) then
    raise dbm_exceptions.ORIGIN_GrO_COEX;
  end if;
end test_ORIGIN_GrO_COEX;
```

In the procedure INSERT\_ORIGIN :

```

-- Tests

dbm_tests.test_ORIGIN_GrO_COEX(v_Col01, v_Col02);
```

In the procedure UPDATE\_ORIGIN :

```

-- Tests

if ((new_Col01 <> old_Col01) or (new_Col01 is null and old_Col01 is not
null) or (new_Col01 is not null and old_Col01 is null))
or ((new_Col02 <> old_Col02) or (new_Col02 is null and old_Col02 is not
null) or (new_Col02 is not null and old_Col02 is null)) then
  dbm_tests.test_ORIGIN_GrO_COEX(new_Col01,
  new_Col02);
end if;
```

### 3.2.8.2 Exclusivity

#### a) Comments

```

-- GrO (Col01, Col02) in ORIGIN : exclusivity
```

#### b) Checks

```

-- Exclusivity (GrO)
alter table ORIGIN add constraint C_ORIGIN_GrO_EXCL
check ((Col01 is null and Col02 is null)
or (Col01 is not null and Col02 is null)
or (Col01 is null and Col02 is not null));
```

### c) Views with check option

```
create or replace view V_ORIGIN as
select * from ORIGIN
where ((Col01 is null and Col02 is null)
or (Col01 is not null and Col02 is null)
or (Col01 is null and Col02 is not null))
with check option;
```

### d) Triggers

```
create or replace trigger T_ORI_GrO_EXCL_AUIR
after insert or update of Col01, Col02 on ORIGIN
for each row
when (not((new.Col01 is null and new.Col02 is null) or (Col01 is not
null and Col02 is null) or (Col01 is null and Col02 is not null)))
begin
raise_application_error(-20004,'Violation of existence constraint') ;
end;
```

### e) Stored procedures

In the package DBM\_TESTS :

```
procedure test_ORIGIN_GrO_EXCL(v_Col01 in ORIGIN.Col01%type, v_Col02 in
ORIGIN.Col02%type) is
begin
  if not ((v_Col01 is null and v_Col02 is null)
or (v_Col01 is not null and v_Col02 is null)
or (v_Col01 is null and v_Col02 is not null))
then
  raise dbm_exceptions.ORIGIN_GrO_EXCL;
end if;
end test_ORIGIN_GrO_EXCL;
```

In the procedure INSERT\_ORIGIN :

```
-- Tests

dbm_tests.test_ORIGIN_GrO_EXCL(v_Col01, v_Col02);
```

In the procedure UPDATE\_ORIGIN :

```
-- Tests

if ((new_Col01 <> old_Col01) or (new_Col01 is null and old_Col01 is not
null) or (new_Col01 is not null and old_Col01 is null))
or ((new_Col02 <> old_Col02) or (new_Col02 is null and old_Col02 is not
null) or (new_Col02 is not null and old_Col02 is null)) then
  dbm_tests.test_ORIGIN_GrO_EXCL(new_Col01,
  new_Col02);
end if;
```

### 3.2.8.3 At-least-one

#### a) Comments

```
-- GrO (Col01, Col02) in ORIGIN : at-least-one
```

#### b) Checks

```
-- At least one (GrO)
alter table ORIGIN add constraint C_ORIGIN_GrO_ATLO
check (Col01 is not null or Col02 is not null);
```

#### c) Views with check option

```
create or replace view V_ORIGIN as
select * from ORIGIN
where (Col01 is not null or Col02 is not null)
with check option;
```

#### d) Triggers

```
create or replace trigger T_ORI_GrO_ATLO_AUIR
after insert or update of Col01, Col02 on ORIGIN
for each row
when (not((new.Col01 is not null or new.Col02 is not null)))
begin
raise_application_error(-20004,'Violation of existence constraint') ;
end;
```

#### e) Stored procedures

In the package DBM\_TESTS :

```
procedure test_ORIGIN_GrO_ATLO(v_Col01 in ORIGIN.Col01%type, v_Col02 in
ORIGIN_2.Col02%type) is
begin
if not (v_Col01 is not null or v_Col02 is not null) then
raise dbm_exceptions.ORIGIN_GrO_ATLO;
end if;
end test_ORIGIN_GrO_ATLO;
```

In the procedure INSERT\_ORIGIN :

```
-- Tests

dbm_tests.test_ORIGIN_GrO_ATLO(v_Col01, v_Col02);
```

In the procedure UPDATE\_ORIGIN :

```
-- Tests

if ((new_Col01 <> old_Col01) or (new_Col01 is null and old_Col01 is not
null) or (new_Col01 is not null and old_Col01 is null))
or ((new_Col02 <> old_Col02) or (new_Col02 is null and old_Col02 is not
null) or (new_Col02 is not null and old_Col02 is null)) then
    dbm_tests.test_ORIGIN_GrO_ATLO(new_Col01,
    new_Col02);
end if;
```

### 3.2.8.4 Exactly-one

#### a) Comments

```
-- GrO (Col01, Col02) in ORIGIN : exactly-one
```

#### b) Checks

```
-- Exactly one (GrO)
alter table ORIGIN add constraint C_ORIGIN_GrO_EXAC
check ((Col01 is not null and Col02 is null)
or (Col01 is null and Col02 is not null));
```

#### c) Views with check option

```
create or replace view V_ORIGIN as
select * from ORIGIN
where ((Col01 is not null and Col02 is null)
or (Col01 is null and Col02 is not null))
with check option;
```

#### d) Triggers

```
create or replace trigger T_ORI_GrO_EXAC_AUIR
after insert or update of Col01, Col02 on ORIGIN
for each row
when (not((new.Col01 is not null and new.Col02 is null) or (new.Col01 is
null and new.Col02 is not null)))
begin
raise_application_error(-20004,'Violation of existence constraint') ;
end;
```

## e) Stored procedures

In the package DBM\_TESTS :

```

procedure test_ORIGIN_GrO_EXAC(v_Col01 in ORIGIN.Col01%type, v_Col02 in
ORIGIN.Col02%type) is
begin
  if not ((v_Col01 is not null and v_Col02 is null)
or (v_Col01 is null and v_Col02 is not null)) then
    raise dbm_exceptions.ORIGIN_GrO_EXAC;
  end if;
end test_ORIGIN_GrO_EXAC;

```

In the procedure INSERT\_ORIGIN :

```

-- Tests

dbm_tests.test_ORIGIN_GrO_EXAC(v_Col01, v_Col02);

```

In the procedure UPDATE\_ORIGIN :

```

-- Tests

if ((new_Col01 <> old_Col01) or (new_Col01 is null and old_Col01 is not
null) or (new_Col01 is not null and old_Col01 is null))
or ((new_Col02 <> old_Col02) or (new_Col02 is null and old_Col02 is not
null) or (new_Col02 is not null and old_Col02 is null)) then
  dbm_tests.test_ORIGIN_GrO_EXAC(new_Col01,
  new_Col02);
end if;

```

## 3.2.9 Maximum cardinality

### 3.2.9.1 Comments

```

-- GrO (Col01) in ORIGIN : maximum cardinality
-- 5

```

### 3.2.9.2 Triggers

```

create or replace trigger T_ORI_GrO_CARD_BIUR
before insert or update of Col01 on ORIGIN
for each row
declare
temp integer;
begin
select count(*) into temp from ORIGIN where ORIGIN.Col01 = new.Col01;
if temp =5 then
raise_application_error(-20007,'Violation of maximum cardinality cons-
traint') ;
end if;
end;

```

### 3.2.9.3 Stored procedures

In the package DBM\_TESTS :

```
procedure test_ORIGIN_GrO_CARD(v_Col01 in ORIGIN.Col01%type) is
temp integer;
begin
select count (*) into temp from ORIGIN
where ORIGIN.Col01 = v_Col01;
if temp=5 then raise dbm_exceptions.ORIGIN_GrO_CARD;
end if;
end test_ORIGIN_GrO_CARD;
```

In the procedure INSERT\_ORIGIN :

```
-- Tests

dbm_tests.test_ORIGIN_GrO_CARD(v_Col01);
```

In the procedure UPDATE\_ORIGIN :

```
-- Tests

if ((new_Col01 <> old_Col01) or (new_Col01 is null and old_Col01 is not
null) or (new_Col01 is not null and old_Col01 is null)) then
dbm_tests.test_ORIGIN_GrO_CARD(new_Col01);
end if;
```



## **PART II**

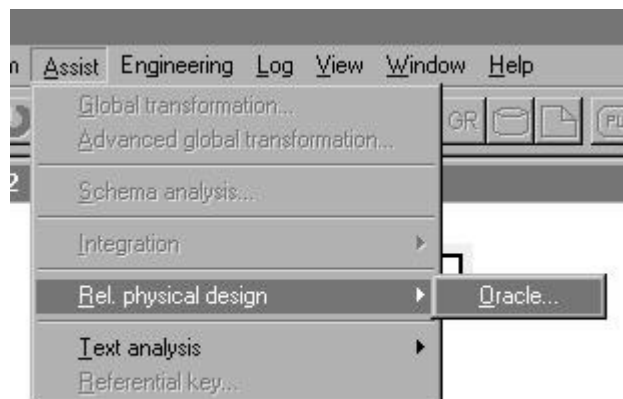
### **THE ASSISTANTS**



## Chapter 4

# Physical design assistant for Oracle 8

The physical design assistant, accessible in DB-Main via the menu "Assist", allows the user to choose the technique of implementation for each type of constraint (or even for each particular constraint) and to create index in the current schema. This assistant can be used on an open logical schema or on a project.



**Figure 4.1** - The menu for physical design assistant

### 4.1 Levels

The modifications on the settings can be done at different levels : corporate, project, schema and local.

- *Corporate* : the settings are saved in a separated file (c:\phydes.ini), used by DB-Main. All projects and schemas will inherit these settings, if there is no more specific setting;
- *Project* : the settings are saved in the project. Each schema in that project will inherit these settings, if there is no more specific setting;
- *Schema* : the settings are saved in the current schema. A schema must be open. Each object in that schema will inherit these settings, if there is no more specific setting;
- *Local* : the settings are saved in the current object (column or group). Only that object will use these settings.

The physical design assistant and the generator for Oracle 8 will use these settings in the following order : for each constraint, they check whether there are local settings. If not, they check the settings in the schema. If there isn't any, they check the settings in the project. If there isn't any, they check the settings in the corporation. If there is no setting at that level, default settings are applied.

## 4.2 Default settings

If the user doesn't specify any setting, the physical design assistant will use the following settings :

- default values will be coded by triggers ;
- constraints on domains will be coded by checks ;
- stable attributes will be coded by triggers ;
- local existence constraints will be coded by checks ;
- foreign keys will be coded by native techniques ;
- primary identifiers will be coded by native techniques ;
- secondary identifiers will be coded by native techniques ;
- maximum cardinality constraints will be coded by triggers ;
- inclusion constraints will be coded by triggers.

## 4.3 The physical design assistant

The user can choose one of the four actions : remove settings, global settings, default settings or local settings. The last two are available only if a schema is open. When the user chooses "Global settings" or "Local settings", a new dialog box is open, where the user can entry his/her settings. The two others actions are default actions. The requested modifications are displayed in the main window, so that the user may verify them before applying or cancelling them.

The exit buttons are :

- *Reset* : undoes all requested modifications
- *Save script...* : saves the text displayed under the title "requested modifications"
- *OK* : closes the dialog box and applies the changes
- *Cancel* : closes the dialog box without applying the changes
- *Help* : displays the help file

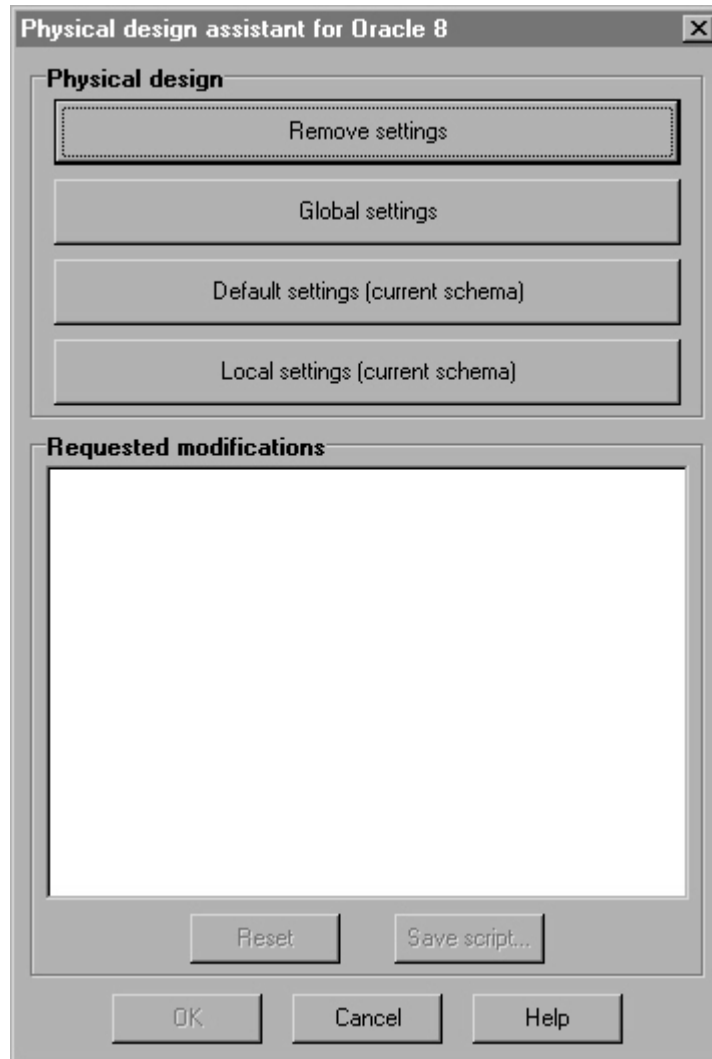


Figure 4.2 - The physical design assistant

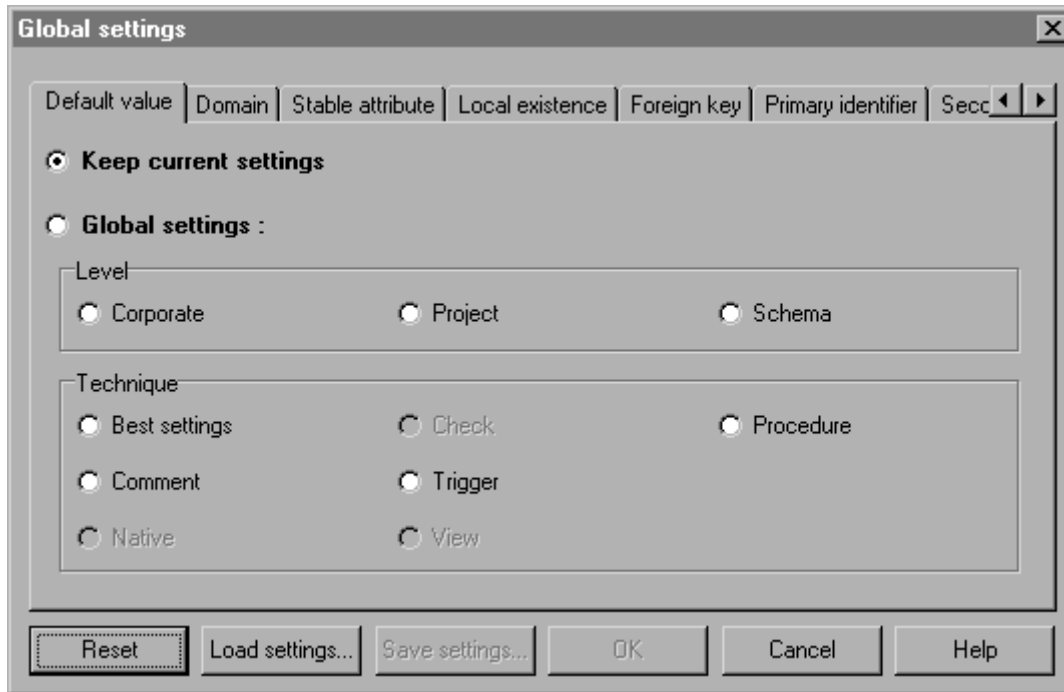
### 4.3.1 Remove settings

This action removes all the user settings. If a project is open, but no schema, it removes all the settings from the corporate and the project level. If a schema is open, it removes all the settings from all levels (corporate, project, schema and local), as well as the indices in that schema.

### 4.3.2 Global settings

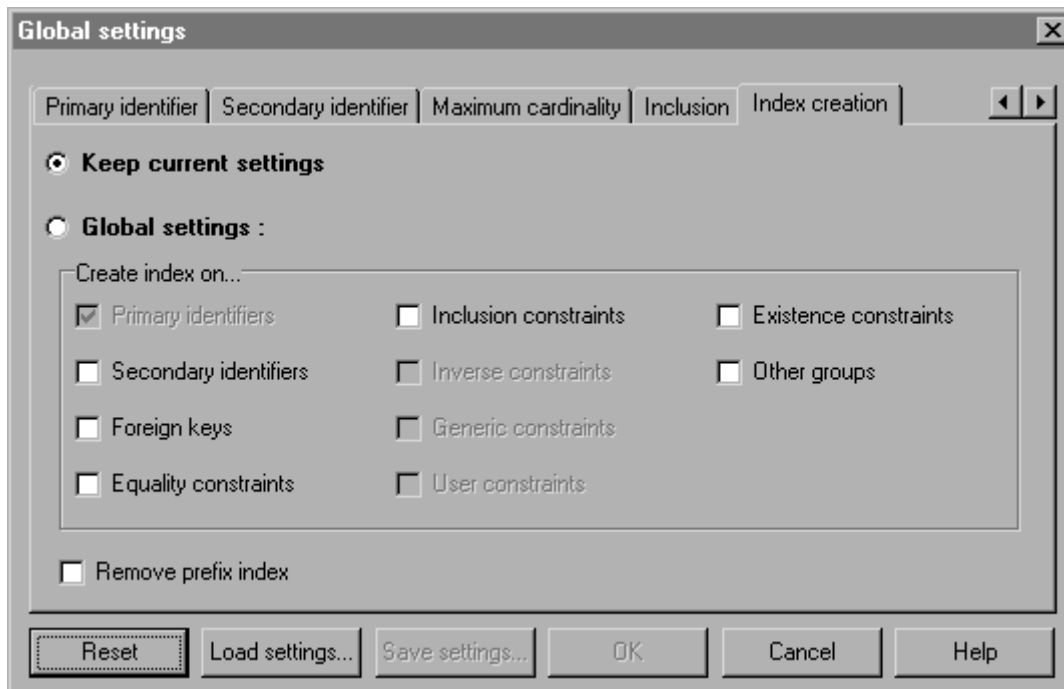
When the user chooses this action, a new dialog box is open, that allows the user to modify options of generation for each type of constraint.

Each constraint has a dedicated tabsheet, in which the user can choose the level of modifications (corporate, project or schema) and the technique used for the generation of all the constraints of that type (best settings, comment, native, check, trigger, view, procedure). Some techniques are unavailable, because they can't be used for the processing of the current constraint.



**Figure 4.3** - Modifying global physical settings for a constraint

A special tabsheet is dedicated to the index creation, and is available only if a schema is open. The user can choose the groups on which create index (primary identifiers, secondary identifiers, foreign keys, equality constraints, inclusion constraints, inverse constraints, generic constraints, user constraints, existence constraints, or other groups, i.e. groups with no special role). Some types of groups are unavailable, if they don't exist in the current schema. The user can also choose to remove prefix index.



**Figure 4.4** - Choosing the index to generate

The exit buttons are :

- *Reset* : undoes all modifications done in the dialog box
- *Load settings...* : loads previously saved settings
- *Save settings...* : saves current settings for further use
- *OK* : closes the dialog box and displays the requested modifications in the main window
- *Cancel* : closes the dialog box without saving
- *Help* : displays the help file

### 4.3.3 Default settings (current schema)

With this option, the user doesn't make any choice. The physical design assistant chooses the best settings for the current schema. The modifications are applied at the schema level. If local settings already exist, they have priority on the schema settings.

#### 4.3.3.1 Index creation

Index will be created on :

- primary identifiers (a primary identifier always supports an index in Oracle 8);
- secondary identifiers;
- foreign keys;
- equality constraints.

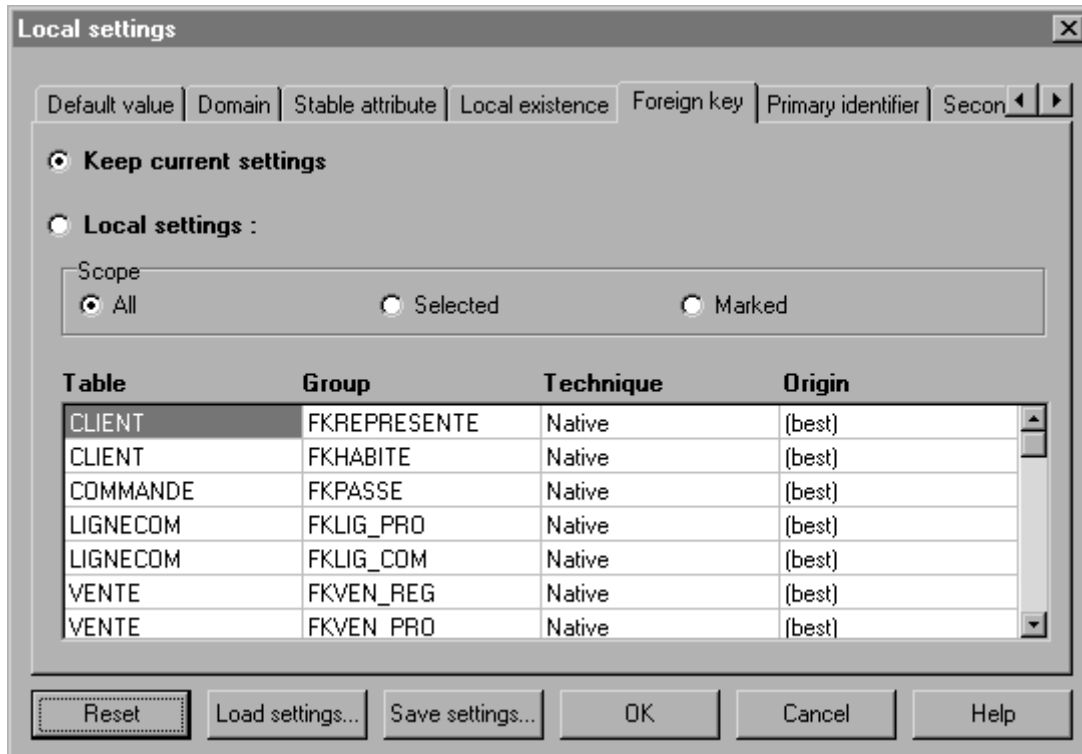
Prefix index will be removed.

#### 4.3.3.2 Techniques of generation

- default values will be coded by triggers ;
- constraints on domains will be coded by checks ;
- stable attributes will be coded by triggers ;
- local existence constraints will be coded by checks ;
- foreign keys will be coded by native techniques ;
- primary identifiers will be coded by native techniques ;
- secondary identifiers will be coded by native techniques ;
- maximum cardinality constraints will be coded by triggers ;
- inclusion constraints will be coded by triggers.

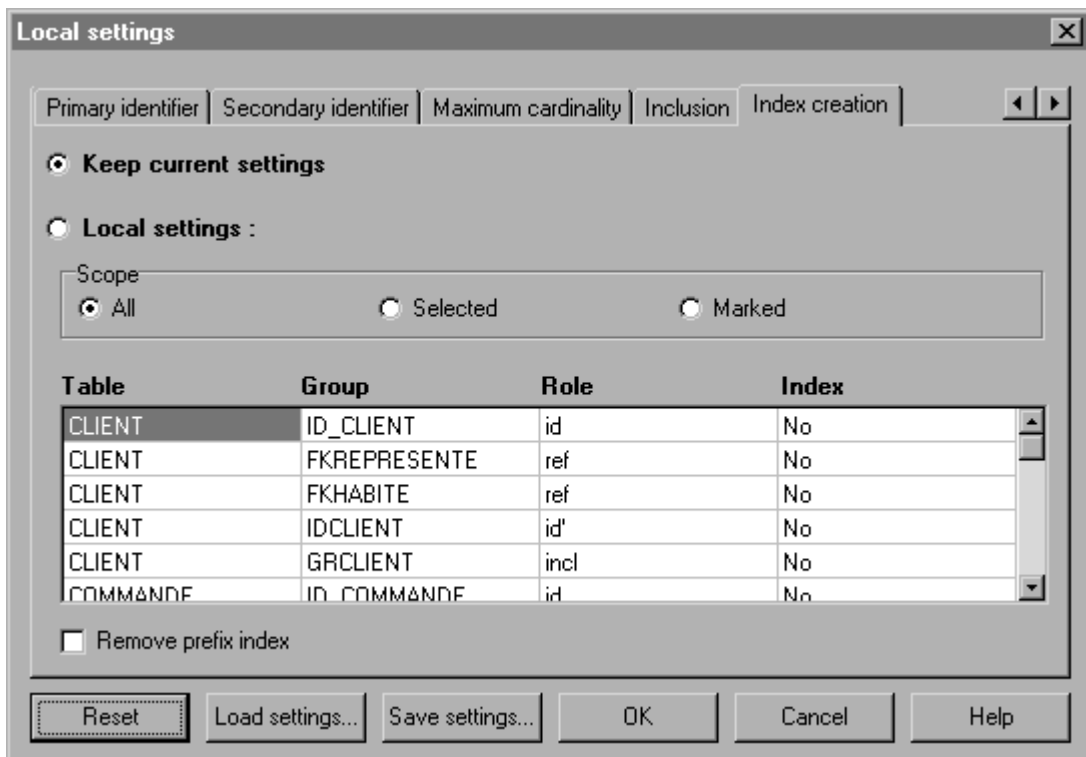
### 4.3.4 Local settings (current schema)

The user can modify the settings for a specific group or column in the current schema. Each constraint has a dedicated tabsheet, where a grid displays all the groups or attributes supporting the current constraint. In this grid, the user can change the technique of generation by double-clicking on the corresponding cell. For his/her convenience, the user can change the scope to display in the grid only the selected items, or the marked items, or all items. Some techniques are unavailable, because they can't be used to process the current constraint.



**Figure 4.5** - Modifying local physical settings for a constraint

A special tabsheet is dedicated to the index creation, and is available only if a schema is open. The user can choose the groups on which create index. The user can also choose to remove prefix index. If this checkbox is checked, and if the user gives the value "Yes" to a group prefix of another group which already has an index, the value "No (Prefix)" is displayed, and no index will be created for that group.



**Figure 4.6** - Choosing the index to generate

The exit buttons are :

- *Reset* : undoes all modifications done in the dialog box
- *Load settings...* : loads previously saved settings
- *Save settings...* : saves current settings for further use
- *OK* : closes the dialog box and displays the requested modifications in the main window
- *Cancel* : closes the dialog box without saving
- *Help* : displays the help file

#### 4.3.4.1 Scope

- *All* : displays all groups/attributes
- *Selected* : displays groups/columns selected in the schema
- *Marked* : displays groups/columns marked in the schema

#### 4.3.4.2 Grid

- *Table* : displays the name of the table containing the group/column specified in the second column
- *Group/Column* : displays the name of the group/column
- *Technique* : displays the technique of generation of the constraint hold by the group/column
- *Origin* : displays the origin of the technique specified in the third column (corporate, project, schema, local or (best), i.e. there is no setting chosen by the user, it's the program that chooses the best technique for that constraint)



## Chapter 5

# Parametric SQL generator for Oracle 8

The parametric SQL generator, accessible in DB-Main via the menu "File", allows the user to choose the options for a code generation : output files, which will contain the code, and constraints to generate. This generator must be used on an open schema, conform to the relational model, but the generator doesn't validate the structures. The files created during the generation are available in DB-Main in the form of documents in the project. The documents are grouped in a product set (a product set by generation).

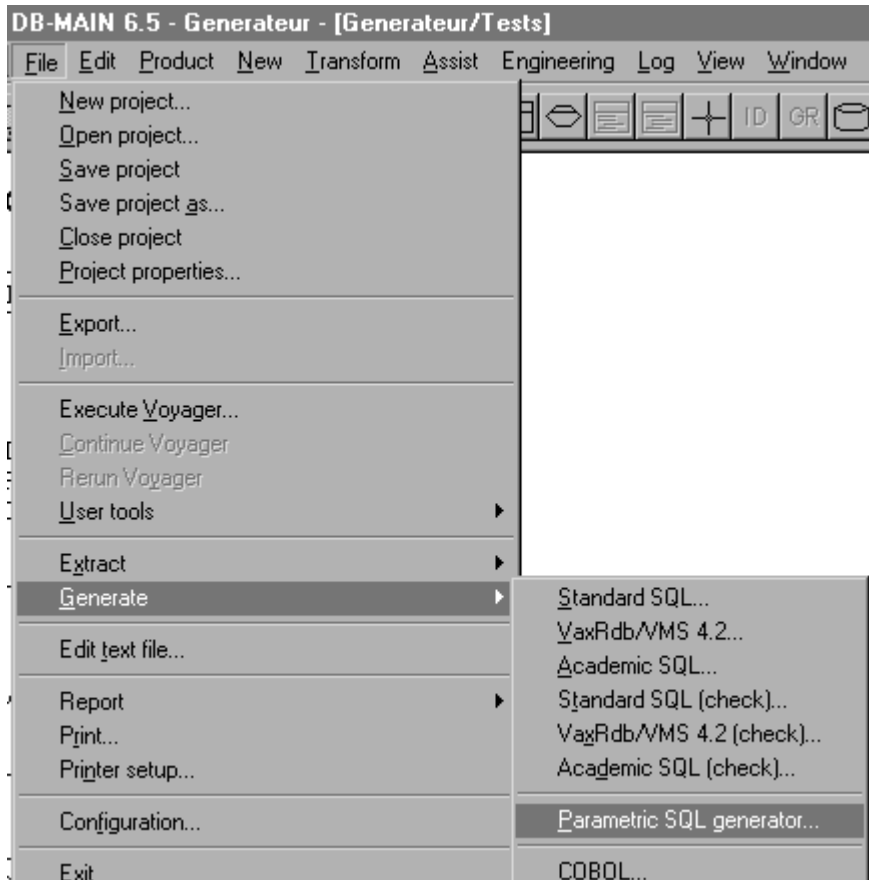
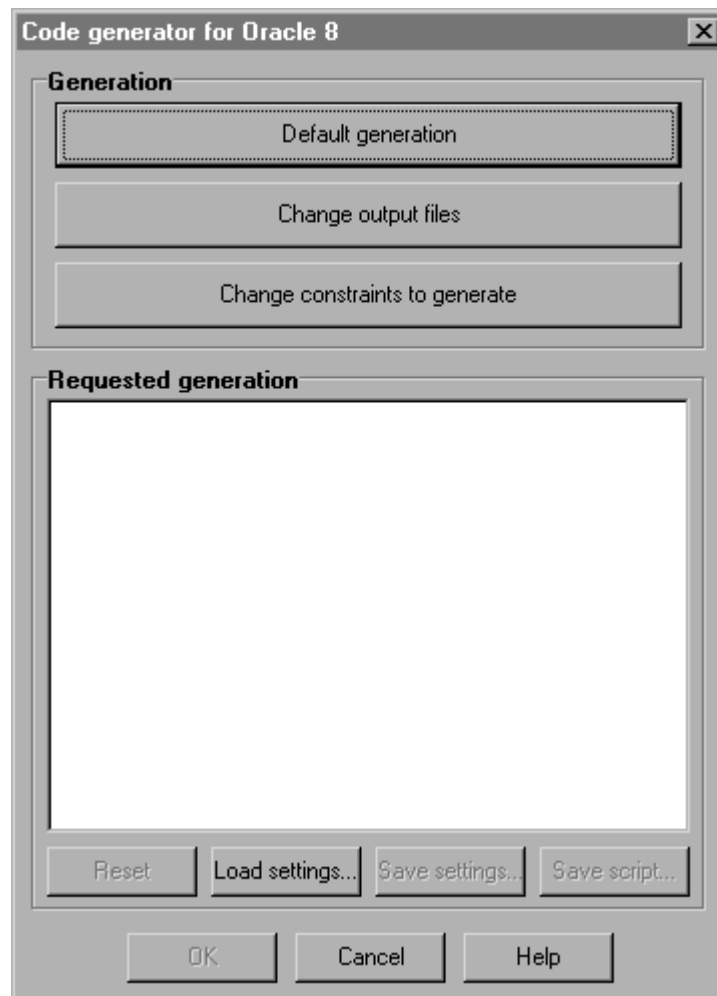


Figure 5.1 - The menu for parametric SQL generator

## 5.1 The code generator assistant



**Figure 5.2** - The code generator assistant

The user can choose one of the three actions : default generation, change output files, change constraints to generate. The requested options for the generation are displayed in the main window, so that the user may verify them before launching or cancelling the generation.

The exit buttons are :

- *Reset* : undoes all requested modifications
- *Load settings* : loads previously saved settings
- *Save settings* : saves current settings for further use
- *Save script...* : saves the text displayed under the title "requested generation"
- *OK* : closes the dialog box and launches the generation
- *Cancel* : closes the dialog box without action
- *Help* : displays the help file

### 5.1.1 Default generation

With this option, the user doesn't make any choice. The code generator assistant chooses the best settings for the current schema : generation of all constraints, one file by technique. These settings may be modified with the buttons "Change output files" and "Change constraints to generate".

### 5.1.1.1 Database creation

Generates the code for the creation of tables and indices.

### 5.1.1.2 Constraints on columns

If such constraints exist in the current schema, they will be generated :

- default values;
- constraints on domains;
- stable attributes.

### 5.1.1.3 Constraints on groups

If such constraints exist in the current schema, they will be generated :

- local existence constraints;
- foreign keys;
- primary identifiers;
- secondary identifiers;
- maximum cardinality;
- inclusion constraints.

### 5.1.1.4 Output files

If such techniques are specified in the current schema, the following files will be created in the current directory :

- for the report : report.txt;
- for the database creation : ddl.sql;
- for the comments : comment.sql;
- for the native techniques : native.sql;
- for the checks : check.sql;
- for the views with check options : view.sql;
- for the triggers : trigger.sql;
- for the procedures : stored.sql.

## 5.1.2 Change output files

In this dialog box, the user may choose the output files that will contain the generated code. Each edit box can be edited manually or via the browse and the delete buttons next to the edit box.

The exit buttons are :

- *OK* : closes the dialog box and saves the changes
- *Cancel* : closes the dialog box without saving
- *Help* : displays the help file

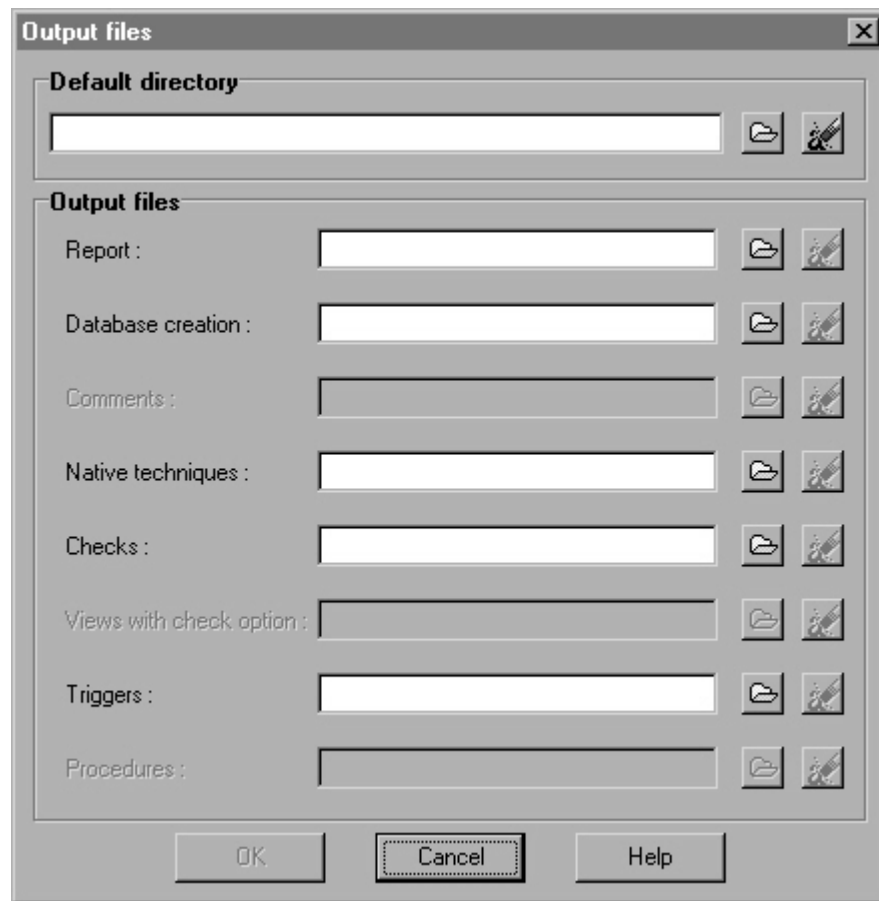


Figure 5.3 - Chosen output files

### 5.1.2.1 Default directory

In this edit box, the user can specify a default directory. The user may write the complete path of the directory, or choose it with the browse button. If the directory doesn't exist, the user will be asked whether he/she wants to create it. If a default directory is chosen, all the output files will receive a default name. The user always can modify these settings and choose another file and/or another directory for a technique.

### 5.1.2.2 Output files

By default, each technique is generated in a specific file. If a user doesn't want a technique to be generated, he/she does not fill the corresponding edit. If a technique is not specified in the physical settings of the schema (in the user settings, via the physical design assistant, or in the default settings), the user can't choose a file for it. A same file can be used for several techniques.

### 5.1.3 Change constraints to generate

In this dialog box, the user chooses the types of constraints that will be generated by checking or unchecking the boxes. Some checkboxes could be unavailable, if the corresponding constraints don't exist in the current schema.

The exit buttons are :

- *OK* : closes the dialog box and saves the changes
- *Cancel* : closes the dialog box without saving
- *Help* : displays the help file

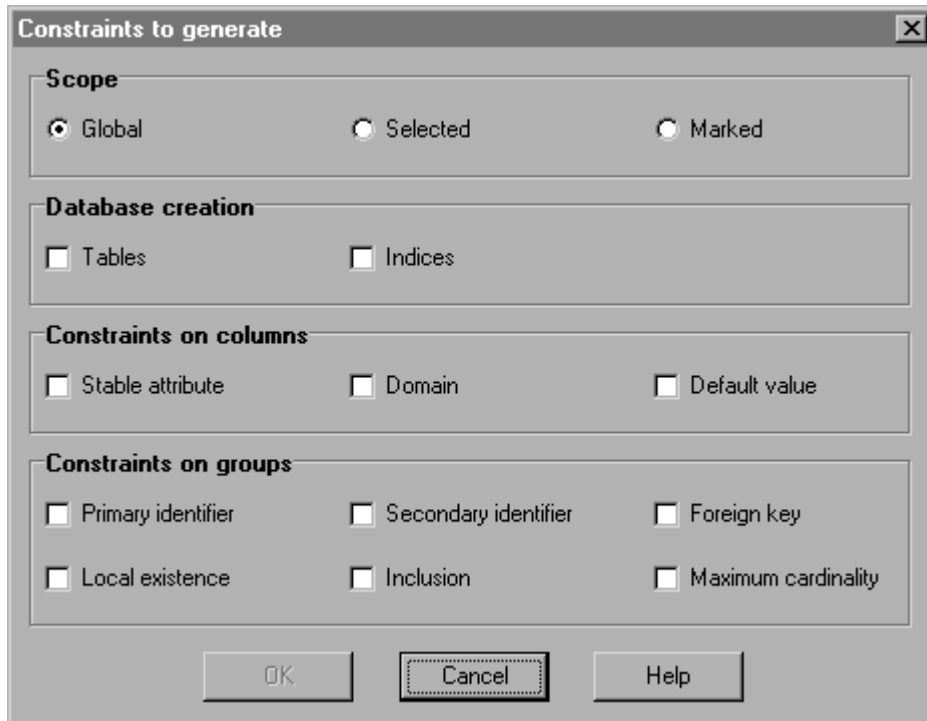


Figure 5.4 - Choosing structures and constraints to generate

### 5.1.3.1 Scope

The user can restrict the generation at some constraints :

- *Global* : all the chosen constraints will be generated;
- *Selected* : the chosen constraints will be generated only if they are selected in the schema;
- *Marked* : the chosen constraints will be generated only if they are marked in the schema.