第7章 高级 SQL

本章包括 DB2 通用数据库的几个功能，这些功能允许您在定制查询以满足您的要求的同时更有效地设计查询。本章中的主题假设您对这之前的材料已理解透彻。

本章包括:

- 用约束和触发器实施商业规则
- 连接
- ROLLUP 和 CUBE 查询和递归查询
- OLAP 函数

用约束和触发器实施商业规则

在商界，需要确保始终实施某些规则。例如，参与某个项目的雇员必须在工资单上。或者，可能我们想要某些事件自动地发生。例如，如果销售人员售出一些商品，则应增加其佣金。

为此，DB2 通用数据库提供了一套有用的方法:

- *唯一约束*禁止在表的一列或多列中出现重复值。
- *参考完整性约束*确保在各个指定的表中的数据的一致性。
- *表检查约束*是限制列所允许的值的规则。如果正在对列指定的值不满足该列的检查约束，则插入和更新失败。
- *触发器*定义一组操作，这组操作是由对指定的表进行删除、插入或更新操作来执行或触发的。触发器可用于写入其他表、修改输入值以及发布警报信息。

第一节提供关键字的概念性概述。稍后，通过示例和图解来研究参考完整性、约束和触发器。

关键字

*关键字*是可用来标识或存取特定行的一组列。

由不止一列组成的关键字称为*组合关键字*。在具有组合关键字的表中，组合关键字中各列的排序不一定与这些列在表中的排序相对应。

唯一关键字

*唯一关键字*定义为其中没有相同的值的列（或一组列）。唯一关键字的列不能包含空值。在执行 INSERT 和 UPDATE 语句期间，数据库管理程序强制执行该约束。一个表可有多个唯一关键字。唯一关键字是可选的，并且可在 CREATE TABLE 或 ALTER TABLE 语句中定义。

主关键字

*主关键字*是一种唯一关键字，表定义的一部分。一个表不能有多个主关键字，并且主关键字的列不能包含空值。主关键字是可选的，并且可在 CREATE TABLE 或 ALTER TABLE 语句中定义。

外部关键字

*外部关键字*在参考约束的定义中指定。一个表可以有零个或多个外部关键字。如果组合外部关键字的值的任何部分为空，则该值为空。外部关键字是可选的，并且可在 CREATE TABLE 语句或 ALTER TABLE 语句中定义。

唯一约束

唯一约束确保关键字的值在表中是唯一的。唯一约束是可选的，并且可以通过使用指定 PRIMARY KEY 或 UNIQUE 子句的 CREATE TABLE 或 ALTER TABLE 语句来定义唯一约束。例如，可在一个表的雇员号列上定义一个唯一约束，以确保每个雇员有唯一的号码。

参考完整性约束

通过定义唯一约束和外部关键字，可以定义表与表之间的关系，从而实施某些商业规则。唯一关键字和外部关键字约束的组合通常称为参考完整性约束。外部关键字所引用的唯一约束称为*父关键字*。外部关键字表示特定的父关键字，或与特定的父关键字相关。例如，某规则可能规定每个雇员（EMPLOYEE 表）必须属于某现存的部门 （DEPARTMENT 表）。因此，将 EMPLOYEE 表中的“部门号”定义为外部关键字，而将 DEPARTMENT 表中的“部门号”定义为主关键字。下列图表提供参考完整性约束的直观说明。

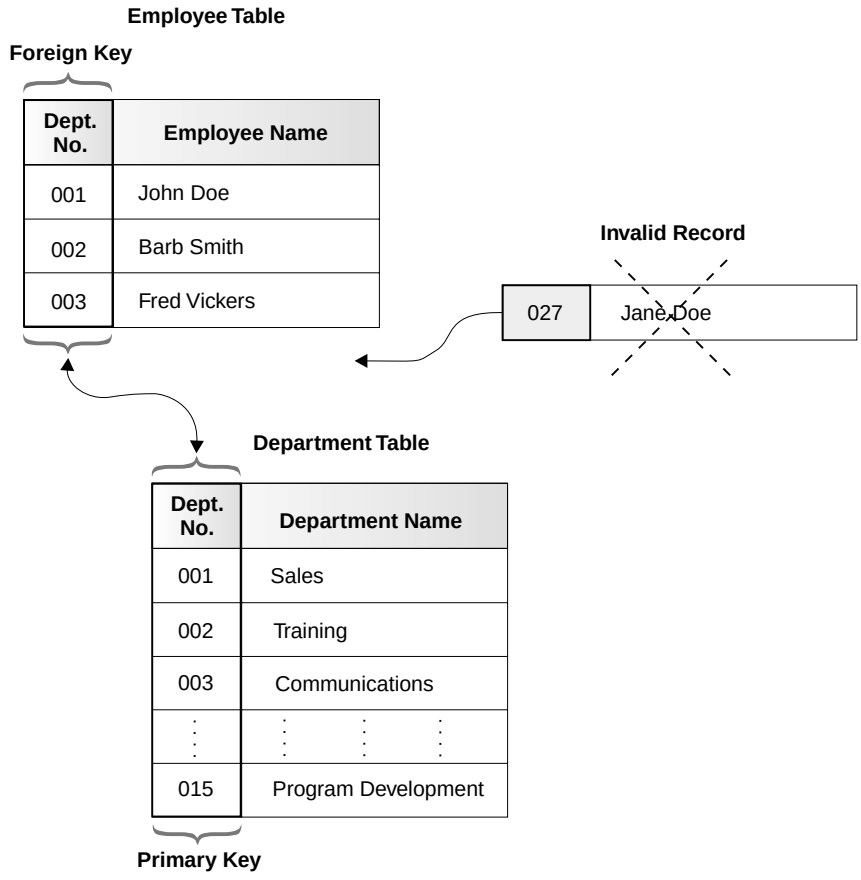

图 4. 外部约束和主约束定义关系并保护数据

表检查约束

*表检查约束*指定对于表的每行都要进行判定的条件。可对个别列指定检查约束。可使用 CREATE 或 ALTER TABLE 语句添加检查约束。

下列语句创建具有下列约束的表:

- 部门号的值必须在范围 10 至 100 内
- 雇员的职务只能为下列之一: “Sales”、“Mgr” 或 “Clerk”
- 1986 年之前雇用的每个雇员的工资必须超过 $40,500。

```
CREATE TABLE EMP
       (ID           SMALLINT NOT NULL,
        NAME         VARCHAR(9),
        DEPT         SMALLINT CHECK (DEPT BETWEEN 10 AND 100),
        JOB          CHAR(5)   CHECK (JOB IN ('Sales', 'Mgr', 'Clerk')),
```

```
HIREDATE     DATE,
SALARY       DECIMAL(7,2),
COMM         DECIMAL(7,2),
PRIMARY KEY (ID),
CONSTRAINT YEARSAL CHECK
                   (YEAR(HIREDATE) >= 1986 OR SALARY > 40500) )
```

仅当条件判定为假时才会违反约束。例如，如果插入行的 DEPT 为空值，则插入继续进行而不出错，尽管 DEPT 的值应该象约束中定义的那样在 10 和 100 之间。

下列语句将一个约束添加至名为 COMP 的 EMPLOYEE 表中，该约束为雇员的总报酬必须超过 $15,000:

```
ALTER TABLE EMP
   ADD CONSTRAINT COMP CHECK (SALARY + COMM > 15000)
```

将检查表中现存的行以确保这些行不违反新约束。可通过使用如下的 SET CONSTRAINTS 语句将此检查延期:

```
SET CONSTRAINTS FOR EMP OFF
ALTER TABLE EMP ADD CONSTRAINT COMP CHECK (SALARY + COMM > 15000)
SET CONSTRAINTS FOR EMP IMMEDIATE CHECKED
```

首先使用 SET CONSTRAINTS 语句以延期对表的约束检查。然后可将一个或多个约束添加至表而不检查这些约束。接着再次发出 SET CONSTRAINTS 语句，反过来将约束检查打开并执行任何延期的约束检查。

触发器

*触发器*定义一组操作。可由修改指定基表中的数据的操作来激活触发器。

触发器的一些用途:

- 执行输入数据的验证
- 自动生成新插入的行的值
- 为交叉引用而从其它表读取
- 为审查记录而写入其它表
- 通过电子邮件信息支持警报

使用触发器将导致应用程序开发和商业规则的全面实施更快速，以及使得应用程序和数据的维护更容易。

DB2 通用数据库支持几种类型的触发器。可定义触发器在 DELETE、INSERT 或 UPDATE 操作之前或之后激活。每个触发器包括一组称为*触发操作*的 SQL 语句，这组语句可包括一个可选的搜索条件。

可进一步定义*后触发器*以对每一行都执行触发操作，或对语句执行一次触发操作。*前触发器*总是对每一行都执行触发操作。

在 INSERT、UPDATE 或 DELETE 语句之前使用触发器，以便在执行触发操作之前检查某些条件，或在将输入值存储在表中之前更改输入值。

使用后触发器，以便在必要时传播值或执行其他任务，如发送信息等，这些任务可能是触发器操作所要求的。

以下示例说明了前触发器和后触发器的一个用途。考虑一个记录并跟踪股票价格波动的应用程序。该数据库包含两个表，CURRENTQUOTE 和 QUOTEHISTORY，定义如下:

```
CREATE TABLE CURRENTQUOTE
(SYMBOL VARCHAR(10),
 QUOTE DECIMAL(5,2),
 STATUS VARCHAR(9))

CREATE TABLE  QUOTEHISTORY
(SYMBOL VARCHAR(10),
 QUOTE DECIMAL(5,2),
 TIMESTAMP TIMESTAMP)
```

当使用如下语句更新 CURRENTQUOTE 的 QUOTE 列时:

```
UPDATE CURRENTQUOTE
   SET QUOTE = 68.5
   WHERE SYMBOL = 'IBM'
```

应更新 CURRENTQUOTE 的 STATUS 列以反映股票是否:

- 在升值
- 处于本年度的新高
- 在下跌
- 处于本年度的新低
- 价位稳定

这通过使用下列前触发器来实现:

1

```
CREATE TRIGGER STOCK_STATUS
   NO CASCADE BEFORE UPDATE OF QUOTE ON CURRENTQUOTE
   REFERENCING NEW AS NEWQUOTE OLD AS OLDQUOTE
   FOR EACH ROW MODE DB2SQL
```

2

```
SET NEWQUOTE.STATUS =

3

   CASE

4

     WHEN NEWQUOTE.QUOTE >=
               (SELECT MAX(QUOTE)
                   FROM QUOTEHISTORY
                   WHERE SYMBOL = NEWQUOTE.SYMBOL
                     AND YEAR(TIMESTAMP) = YEAR(CURRENT DATE) )
     THEN 'High'

5

   WHEN NEWQUOTE.QUOTE <=
               (SELECT MIN(QUOTE)
                   FROM QUOTEHISTORY
                   WHERE SYMBOL = NEWQUOTE.SYMBOL
                   AND YEAR(TIMESTAMP) = YEAR(CURRENT DATE) )
   THEN 'Low'

6

   WHEN NEWQUOTE.QUOTE > OLDQUOTE.QUOTE
      THEN 'Rising'
   WHEN NEWQUOTE.QUOTE < OLDQUOTE.QUOTE
      THEN 'Dropping'
   WHEN NEWQUOTE.QUOTE = OLDQUOTE.QUOTE
      THEN 'Steady'
END
```

1 此代码块将 STOCK_STATUS 定义为一个应该在更新 CURRENTQUOTE 表的 QUOTE 列之前激活的触发器。第二行指定，在将 CURRENTQUOTE 表的实际更新所引起的任何更改应用于数据库之前，要应用触发操作。NO CASCADE 子句意味着触发操作将不会导致激活任何其他触发器。第三行指定一些名称，必须将这些名称作为列名的限定符用于新值 (NEWQUOTE) 和旧值 (OLDQUOTE)。用这些相关名（NEWQUOTE 和 OLDQUOTE）限定的列名称为*转换变量*。第四行表示应对每一行都执行触发操作。

2 这标记此触发器的触发操作中第一个也是唯一的一个 SQL 语句的开始。SET 转换变量语句在一个触发器中用来将值赋给表的行中的列，该表正在由激活该触发器的语句进行更新。此语句正在给 CURRENTQUOTE 表的 STATUS 列赋值。

3 该赋值语句右边使用的表达式为 CASE 表达式。 CASE 表达式扩充为 END 关键字。

4 第一种情况检查新报价 (NEWQUOTE.QUOTE) 是否超过当前日历年度中股票符号的最高价。子查询正在使用由跟在后面的后触发器更新的 QUOTEHISTORY 表。

5 第二种情况检查新报价 (NEWQUOTE.QUOTE) 是否小于当前日历年度中股票符号的最低价。子查询正在使用由跟在后面的后触发器更新的 QUOTEHISTORY 表。

6 最后三种情况将新报价 (NEWQUOTE.QUOTE) 与表 (OLDQUOTE.QUOTE) 中的报价比较，以确定新报价是大于、小于还是等于旧报价。 SET 转换变量语句在此处结束。

除了更新 CURRENTQUOTE 表中的项之外，还需要通过将新报价连同时间戳记一起复制到 QUOTEHISTORY 表中来创建一个审查记录。这通过使用下列后触发器来实现:

1

```
CREATE TRIGGER RECORD_HISTORY
AFTER UPDATE OF QUOTE ON CURRENTQUOTE
REFERENCING NEW AS NEWQUOTE
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
```

2

```
INSERT INTO QUOTEHISTORY
   VALUES (NEWQUOTE.SYMBOL, NEWQUOTE.QUOTE, CURRENT TIMESTAMP);
END
```

1 此代码块将命名为 RECORD_HISTORY 的触发器定义为应该在更新 CURRENTQUOTE 表的 QUOTE 列之后激活的触发器。第三行指定应作为列名的限定符用于新值 (NEWQUOTE) 的名称。第四行表示应对每一行都执行触发操作。

2 此触发器的触发操作包括单个 SQL 语句，该语句使用已更新的行中的数据（ NEWQUOTE.SYMBOL 和 NEWQUOTE.QUOTE ）和当前的时间戳记将该行插入 QUOTEHISTORY 表。

CURRENT TIMESTAMP 是包含时间戳记的专用寄存器。第68页的『专用寄存器』中提供了列表和解释。

连接

从两个或更多个表中组合数据的过程称为连接表。数据库管理程序从指定的表中形成行的所有组合。对于每个组合，它都测试*连接条件*。连接条件是带有一些约束的搜索条件。有关约束的列表，参考 *SQL Reference*。

注意，连接条件涉及的列的数据类型不必相同；然而，这些数据类型必须相容。计算连接条件的方式与计算其他搜索条件的方式相同，并且使用相同的比较规则。

如果未指定连接条件，则返回在 FROM 子句中列出的表中行的所有组合，即使这些行可能完全不相关。该结果称为这两个表的*交叉积*。

本节中的示例基于下面两个表。这两个表只是样本数据库中表的简化形式，在样本数据库中并不存在。这两个表一般用来概述关于连接的重点。 SAMP_STAFF 列出未作为合同工雇用的雇员的姓名以及这些雇员的职务说明，而 SAMP_PROJECT 则列出雇员（合同工和全职人员）的姓名以及这些雇员所参与的项目。

这些表如下:

NAME	PROJ
Haas	AD3100
Thompson	PL2100
Walker	MA2112
Lutz	MA2111

图 5. SAMP_PROJECT 表

NAME	JOB
Haas	PRES
Thompson	MANAGER
Lucchessi	SALESREP
Nicholls	ANALYST

图 6. SAMP_STAFF 表

以下示例产生两个表的交叉积。未指定连接条件，因而存在每一个行组合:

```
SELECT SAMP_PROJECT.NAME,
       SAMP_PROJECT.PROJ, SAMP_STAFF.NAME, SAMP_STAFF.JOB
  FROM SAMP_PROJECT, SAMP_STAFF
```

此语句产生下列结果:

```
NAME       PROJ   NAME       JOB
---------- ------ ---------- --------
Haas       AD3100 Haas       PRES
Thompson   PL2100 Haas       PRES
Walker     MA2112 Haas       PRES
Lutz       MA2111 Haas       PRES
Haas       AD3100 Thompson   MANAGER
Thompson   PL2100 Thompson   MANAGER
Walker     MA2112 Thompson   MANAGER
Lutz       MA2111 Thompson   MANAGER
Haas       AD3100 Lucchessi  SALESREP
Thompson   PL2100 Lucchessi  SALESREP
Walker     MA2112 Lucchessi  SALESREP
Lutz       MA2111 Lucchessi  SALESREP
Haas       AD3100 Nicholls   ANALYST
Thompson   PL2100 Nicholls   ANALYST
Walker     MA2112 Nicholls   ANALYST
Lutz       MA2111 Nicholls   ANALYST
```

两个主要的连接类型是*内连接*和*外连接*。到目前为止，所有示例中使用的都是内连接。内连接只保留交叉积中满足连接条件的那些行。如果某行在一个表中存在，但在另一个表中不存在，则结果表中不包括该信息。

下列示例产生两个表的内连接。该内连接列出分配给某个项目的全职雇员:

```
SELECT SAMP_PROJECT.NAME,
       SAMP_PROJECT.PROJ, SAMP_STAFF.NAME, SAMP_STAFF.JOB
  FROM SAMP_PROJECT, SAMP_STAFF
  WHERE SAMP_STAFF.NAME = SAMP_PROJECT.NAME
```

或者，也可以指定如下内连接:

```
SELECT SAMP_PROJECT.NAME,
       SAMP_PROJECT.PROJ, SAMP_STAFF.NAME, SAMP_STAFF.JOB
  FROM SAMP_PROJECT INNER JOIN SAMP_STAFF
    ON SAMP_STAFF.NAME = SAMP_PROJECT.NAME
```

结果是:

```
NAME       PROJ                NAME       JOB
---------- ------ ---------- --------
Haas       AD3100              Haas       PRES
Thompson   PL2100              Thompson   MANAGER
```

注意，该内连接的结果由与右表和左表中 NAME 列的值匹配的行组成 – 'Haas' 和 'Thompson' 都包括在 SAMP_STAFF 表（该表列出所有全职雇员）中和 SAMP_PROJECT 表（该表列出分配给某个项目的全职雇员和合同雇员）中。

外连接是内连接和左表和 / 或右表中不在内连接中的那些行的并置。当对两个表执行外连接时，可任意将一个表指定为左表而将另一个表指定为右表。外连接有三种类型:

1. ***左外连接***包括内连接和左表中未包括在内连接中的那些行。
2. ***右外连接***包括内连接和右表中未包括在内连接中的那些行。
3. ***全外连接***包括内连接以及左表和右表中未包括在内连接中的行。

使用 SELECT 语句来指定要显示的列。在 FROM 子句中，列出后跟关键字 LEFT OUTER JOIN、RIGHT OUTER JOIN 或 FULL OUTER JOIN 的第一个表的名称。接着需要指定后跟 ON 关键字的第二个表。在 ON 关键字后面，指定表示要连接的表之间关系的连接条件。

在下列示例中，将 SAMP_STAFF 指定为右表，而 SAMP_PROJECT 则被指定为左表。通过使用 LEFT OUTER JOIN，列出所有全职雇员和合同雇员（在 SAMP_PROJECT 中列出）的姓名和项目号，如果是全职雇员（在 SAMP_STAFF 中列出），还列出这些雇员的职位:

```
SELECT SAMP_PROJECT.NAME, SAMP_PROJECT.PROJ,
       SAMP_STAFF.NAME, SAMP_STAFF.JOB
  FROM SAMP_PROJECT LEFT OUTER JOIN SAMP_STAFF
    ON SAMP_STAFF.NAME = SAMP_PROJECT.NAME
```

此语句产生下列结果:

```
NAME       PROJ                NAME       JOB
---------- ------------------- ---------- --------------------
Haas       AD3100              Haas       PRES
Lutz       MA2111              -          -
Thompson   PL2100              Thompson   MANAGER
Walker     MA2112              -          -
```

所有列中都具有值的那些行是该内连接的结果。这些都是满足连接条件的行: 'Haas'和'Thompson'既在 SAMP_PROJECT（左表）中列出又在 SAMP_STAFF（右表）中列出。对于不满足连接条件的行，右表的列上出现空值: 'Lutz' 和 'Walker' 都是在 SAMP_PROJECT 表中列出的合同雇员，因而未在 SAMP_STAFF 表中列出。注意，左表中的所有行都包括在结果集中。

在下一个示例中，将 SAMP_STAFF 指定为右表而 SAMP_PROJECT 则被指定为左表。通过使用 RIGHT OUTER JOIN 列出所有全职雇员（在 SAMP_STAFF 中列出）的姓名和工作职位，如果将这些雇员分配给了某个项目（在 SAMP_PROJECT 中列出），还列出他们的项目编号:

```
SELECT SAMP_PROJECT.NAME,
       SAMP_PROJECT.PROJ, SAMP_STAFF.NAME, SAMP_STAFF.JOB
  FROM SAMP_PROJECT RIGHT OUTER JOIN SAMP_STAFF
    ON SAMP_STAFF.NAME = SAMP_PROJECT.NAME
```

结果为:

```
NAME       PROJ                NAME       JOB
---------- ------------------- ---------- --------------------
Haas       AD3100              Haas       PRES
-          -                   Lucchessi  SALESREP
-          -                   Nicholls   ANALYST
Thompson   PL2100              Thompson   MANAGER
```

象在左外连接中一样，所有列中都具有值的那些行是内连接的结果。这些都是满足连接条件的行: 'Haas'和'Thompson'既在 SAMP_PROJECT（左表）中列出又在 SAMP_STAFF（右表）中列出。对于不满足连接条件的行，右表的列上出现空值: 'Lucchessi' 和 'Nicholls' 都是未分配给某项目的全职雇员。虽然他们在 SAMP_STAFF 中列出，但未在 SAMP_PROJECT 中列出。注意，右表中的所有行都包括在结果集中。

下一个示例对 SAMP_PROJECT 表和 SAMP_STAFF 表使用 FULL OUTER JOIN。该示例列出所有全职雇员（包括未分配给某项目的雇员）和合同雇员的姓名:

```
SELECT SAMP_PROJECT.NAME, SAMP_PROJECT.PROJ,
       SAMP_STAFF.NAME, SAMP_STAFF.JOB
  FROM SAMP_PROJECT FULL OUTER JOIN SAMP_STAFF
    ON SAMP_STAFF.NAME = SAMP_PROJECT.NAME
```

结果为:

```
NAME       PROJ                NAME       JOB
---------- ------------------- ---------- --------------------
Haas       AD3100              Haas       PRES
-          -                   Lucchessi  SALESREP
-          -                   Nicholls   ANALYST
Thompson   PL2100              Thompson   MANAGER
Lutz       MA2111              -          -
Walker     MA2112              -          -
```

此结果包括左外连接、右外连接以及内连接。列出所有全职雇员和合同雇员。正如左外连接和右外连接一样，对于不满足连接条件的值，各个列中会出现空值。SAMP_STAFF 和 SAMP_PROJECT 中的每一行都包括在结果集中。

复杂查询

DB2 通用数据库允许您通过使用 ROLLUP 和 CUBE 分组、合并及查看单个结果集中的多列。这种新型而强大的功能增强并简化了基于数据分析的 SQL。

有很多方法可从数据库中抽取有用信息。可执行递归查询从现存数据集中产生结果表。

ROLLUP 和 CUBE 查询

在查询的 GROUP BY 子句中指定 ROLLUP 和 CUBE 运算。