

DB2 UDB V7.2

SQL Cookbook

Graeme Birchall

11-Mar-2002

Preface

IBM's DB2 UDB V7.2 comes with a very advanced SQL language that has a lot of powerful features. This book contains numerous examples illustrating how this SQL dialect can be used to answer a variety of every-day business questions.

TRADEMARKS: AIX, DB2, and DB2 MVS, are registered trademarks of the IBM Corporation. Windows, Windows/NT, Windows/2000, Windows/ME, and Word for Windows are registered trademarks of the Microsoft Corporation. Adobe Acrobat is a registered trademark of the Adobe Corporation.

Disclaimer & Copyright

DISCLAIMER: This document is a best effort on my part. However, given the nature of the subject matter covered, it would be extremely unwise to trust the contents in its entirety. I certainly don't. If you do something silly based on what I say, life is tough.

COPYRIGHT: This document may not be copied in whole, or in part, without the explicit written permission of the distributor. Distribution does not imply publication. This copy was printed on 11 March, 2002. It is intended for personal use only. Secondary distribution for gain is not allowed. File name: BOOK08.DOC.

Tools Used

HARDWARE: This book was written on a Dell 400 XPS with 128 MEG of RAM. For printing, I use an HP 4000 double-sided laser printer.

SOFTWARE: DB2 UDB V7.2 Personal Edition, service level WR21270 (i.e. FixPak 4), was used for all testing. Word for Windows was used to write the document. Adobe Acrobat was used to make the PDF file. One thing that all of this software (except DB2 of course) has in common is that it is all bug-ridden junk. I could have written this book in half the time if any of it was even mildly reliable.

Book Binding

A bound book is a lot easier to work with than loose pages, or the same in a three-ring binder. Because few people have access to professional binding machinery, I did some experiments a couple of years ago - binding books using commonly available materials. I came up with what I consider to be a very satisfactory solution that is fully document on page 271.

PRINTING: This book looks best when printed on a double-sided printer.

Author / Book

Author: Graeme Birchall ©
Address: 1 River Court, Apt 1706
Jersey City NJ 07310-2007
Ph/Fax: (201)-963-0071
Email: Graeme_Birchall@compuserve.com
Web: http://ourworld.compuserve.com/homepages/Graeme_Birchall

Title: DB2 UDB V7.2 SQL Cookbook ©
Print: 11 March, 2002
Fname: BOOK08.DOC
#pages: 277

Author Notes

Book History

This book originally began a series of notes for my own use. After a while, friends began to ask for copies, so I decided to tidy everything up and give it away. Over the years, new chapters have been added as DB2 has evolved, and as I have figured out ways to solve new problems. Hopefully, this process will continue for the foreseeable future.

Why Free

This book is free because I want people to use it. The more people that use it, and the more that it helps them, then the more inclined I am to keep it up to date. For these reasons, if you find this book to be useful, please share it with others.

There are several other reasons why the book is free, as opposed to formally published. I want whatever I put into the public domain to be the best that I can write, and I feel that my current distribution setup results (in this case) in a better quality product than that for a comparable published book. What I lack is an editor and a graphic designer to fix my many errors. But I have a very fast time to print - new editions of this book usually come out within two months of new versions of DB2 becoming available. And corrections and/or enhancements can be included almost immediately. Lastly, I am under no pressure to make the book marketable. I simply include whatever I think might be useful.

Other Free Documents

The following documents are also available for free from my web site:

- **SAMPLE SQL:** The complete text of the SQL statements in this Cookbook are available in an HTML file. Only the first and last few lines of the file have HTML tags, the rest is raw text, so it can easily be cut and paste into other files.
- **CLASS OVERHEADS:** Selected SQL examples from this book have been rewritten as class overheads. This enables one to use this material to teach DB2 SQL to others. Use this cookbook as the student notes.
- **OLDER EDITIONS:** This book is rewritten, and usually much improved, with each new version of DB2. The version 6.1 and 5.2 editions of this book are available from my web site. The version 5.0 and V2.1 are also available, via email, upon request.

Answering Questions

As a rule, I do not answer technical questions because I need to have a life. But I'm interested in hearing about **interesting** SQL problems. However do not expect to get a prompt response, nor necessarily any response. And if you are obviously an idiot, don't be surprised if I point out (for free, remember) that you are idiot.

Graeme

Book Editions

Upload Dates

1996-05-08	First edition of the DB2 V2.1.1 SQL Cookbook was posted to my web site. This version was in Postscript Print File format.
1998-02-26	The DB2 V2.1.1 SQL Cookbook was converted to an Adobe Acrobat file and posted to my web site. Some minor cosmetic changes were made.
1998-08-19	First edition of the DB2 UDB V5 SQL Cookbook was posted. Every SQL statement was checked for V5, and there were new chapters on OUTER JOIN and GROUP BY.
1998-08-26	About 20 minor cosmetic defects were corrected in the V5 Cookbook.
1998-09-03	Another 30 or so minor defects were corrected in the V5 Cookbook.
1998-10-24	The Cookbook was updated for DB2 UDB V5.2.
1998-10-25	About twenty minor typos and sundry cosmetic defects were fixed.
1998-12-03	IBM published two versions of the V5.2 upgrade. The initial edition, which I had used, evidently had a lot of errors. It was replaced within a week with a more complete upgrade. This book was based on the later upgrade.
1999-01-25	A chapter on Summary Tables (new in the Dec/98 fixpack) was added and all the SQL was checked for changes.
1999-01-28	Some more SQL was added to the new chapter on Summary Tables.
1999-02-15	The section of stopping recursive SQL statements was completely rewritten, and a new section was added on denormalizing hierarchical data structures.
1999-02-16	Minor editorial changes were made.
1999-03-16	Some bright spark at IBM pointed out that my new and improved section on stopping recursive SQL was all wrong. Damn. I undid everything.
1999-05-12	Minor editorial changes were made, and one new example (on getting multiple counts from one value) was added.
1999-09-16	DB2 V6.1 edition. All SQL was rechecked, and there were some minor additions - especially to summary tables, plus a chapter on "DB2 Dislikes".
1999-09-23	Some minor layout changes were made.
1999-10-06	Some errors fixed, plus new section on index usage in summary tables.
2000-04-12	Some typos fixed, and a couple of new SQL tricks were added.
2000-09-19	DB2 V7.1 edition. All SQL was rechecked. The new areas covered are: OLAP functions (whole chapter), ISO functions, and identity columns.
2000-09-25	Some minor layout changes were made.
2000-10-26	More minor layout changes.
2001-01-03	Minor layout changes (to match class notes).
2001-02-06	Minor changes, mostly involving the RAND function.
2001-04-11	Document new features in latest fixpack. Also add a new chapter on Identity Columns and completely rewrite sub-query chapter.
2001-10-24	DB2 V7.2 fixpack 4 edition. Tested all SQL and added more examples, plus a new section on the aggregation function.
2002-03-11	Minor changes, mostly to section on precedence rules.

Table of Contents

PREFACE.....	3
Disclaimer & Copyright.....	3
Tools Used	3
Book Binding	3
Author / Book.....	3
Author Notes	4
Book History	4
Why Free	4
Other Free Documents.....	4
Answering Questions	4
Book Editions	5
Upload Dates.....	5
TABLE OF CONTENTS.....	7
INTRODUCTION TO SQL	13
Syntax Diagram Conventions.....	13
SQL Components	13
DB2 Objects	13
SELECT Statement.....	15
FETCH FIRST Clause.....	17
Correlation Name	18
Renaming Fields	19
Working with Nulls.....	19
SQL Predicates.....	20
Basic Predicate	20
Quantified Predicate.....	21
BETWEEN Predicate	22
EXISTS Predicate	22
IN Predicate.....	22
LIKE Predicate	23
NULL Predicate	24
Precedence Rules.....	24
Temporary Tables (in Statement).....	25
Common Table Expression.....	26
Full-Select.....	29
CAST Expression	32
VALUES Clause	33
CASE Expression	35
COLUMN FUNCTIONS	39
Introduction.....	39
Column Functions, Definitions.....	39
AVG	39
CORRELATION	41
COUNT	41
COUNT_BIG.....	42
COVARIANCE.....	42
GROUPING	43
MAX.....	43
MIN	44
REGRESSION	44
STDDEV	45
SUM.....	45
VAR or VARIANCE	46
OLAP FUNCTIONS.....	47
Introduction.....	47
The Bad Old Days	47
OLAP Functions, Definitions.....	50
Ranking Functions.....	50
Row Numbering Function.....	56
Aggregation Function	62
SCALAR FUNCTIONS	71
Introduction.....	71
Sample Data.....	71

Scalar Functions, Definitions	71
ABS or ABSVAL	71
ACOS	72
ASCII	72
ASIN	72
ATAN	72
ATAN2	72
BIGINT	72
BLOB	73
CEIL or CEILING	73
CHAR	74
CHR	75
CLOB	76
COALESCE	76
CONCAT	77
COS	78
COT	78
DATE	78
DAY	79
DAYNAME	79
DAYOFWEEK	79
DAYOFWEEK_ISO	80
DAYOFYEAR	80
DAYS	80
DBCLOB	81
DEC or DECIMAL	81
DEGREES	81
DEREF	81
DECRYPT_BIN and DECRYPT_CHAR	81
DIFFERENCE	82
DIGITS	82
DLCOMMENT	83
DLINKTYPE	83
DLURLCOMPLETE	83
DLURLPATH	83
DLURLPATHONLY	83
DLURLSCHEME	83
DLURLSERVER	83
DLVALUE	83
DOUBLE or DOUBLE_PRECISION	83
ENCRYPT	83
EVENT_MON_STATE	84
EXP	84
FLOAT	85
FLOOR	85
GENERATE_UNIQUE	85
GETHINT	86
GRAPHIC	86
HEX	86
HOURL	87
IDENTITY_VAL_LOCAL	87
INSERT	88
INT or INTEGER	88
JULIAN_DAY	88
LCASE or LOWER	90
LEFT	91
LENGTH	91
LN or LOG	92
LOCATE	92
LOG or LN	92
LOG10	92
LONG_VARCHAR	93
LONG_VARGRAPHIC	93
LOWER	93
LTRIM	93
MICROSECOND	93
MIDNIGHT_SECONDS	93
MINUTE	94
MOD	94
MONTH	95
MONTHNAME	95
MULTIPLY_ALT	95
NODENUMBER	96
NULLIF	96
PARTITION	96

POSSTR.....	97
POWER.....	97
QUARTER.....	97
RADIANS.....	97
RAISE_ERROR.....	98
RAND.....	98
REAL.....	101
REC2XML.....	102
REPEAT.....	102
REPLACE.....	102
RIGHT.....	103
ROUND.....	103
RTRIM.....	103
SECOND.....	103
SIGN.....	104
SIN.....	104
SMALLINT.....	104
SOUNDEX.....	104
SPACE.....	105
SQLCACHE_SNAPSHOT.....	106
SQRT.....	106
SUBSTR.....	107
TABLE.....	108
TABLE_NAME.....	108
TABLE_SCHEMA.....	108
TAN.....	109
TIME.....	109
TIMESTAMP.....	109
TIMESTAMP_ISO.....	109
TIMESTAMPDIFF.....	110
TRANSLATE.....	111
TRUNC or TRUNCATE.....	112
TYPE_ID.....	112
TYPE_NAME.....	112
TYPE_SECHEMA.....	112
UCASE or UPPER.....	112
VALUE.....	113
VARCHAR.....	113
VARGRAPHIC.....	113
VEBLOB_CP_LARGE.....	113
VEBLOB_CP_LARGE.....	113
WEEK.....	113
WEEK_ISO.....	113
YEAR.....	114
ORDER BY, GROUP BY, AND HAVING.....	115
Introduction.....	115
Order By.....	115
Sample Data.....	115
Order by Examples.....	115
Notes.....	116
Group By and Having.....	117
GROUP BY Sample Data.....	117
Simple GROUP BY Statements.....	117
GROUPING SETS Statement.....	119
ROLLUP Statement.....	123
CUBE Statement.....	127
Complex Grouping Sets - Done Easy.....	129
Group By and Order By.....	131
Group By in Join.....	132
COUNT and No Rows.....	132
JOINS.....	133
Why Joins Matter.....	133
Sample Views.....	133
Join Syntax.....	133
Join Types.....	135
Inner Join.....	135
Cartesian Product.....	135
Left Outer Join.....	137
Right Outer Join.....	137
Outer Joins.....	138
Outer Join Notes.....	139

Using the COALESCE Function.....	139
ON Predicate Processing.....	139
WHERE Predicate Processing.....	140
Listing non-matching rows only.....	142
Outer Join in SELECT Phrase.....	144
Predicates and Joins, a lesson.....	145
Summary.....	146
SUB-QUERY	147
Sample Tables.....	147
Sub-query Flavours.....	147
Sub-query Syntax.....	147
Correlated vs. Uncorrelated Sub-Queries.....	154
Multi-Field Sub-Queries.....	155
Nested Sub-Queries.....	155
Usage Examples.....	156
True if NONE Match.....	156
True if ANY Match.....	157
True if TEN Match.....	158
True if ALL match.....	159
UNION, INTERSECT, AND EXCEPT	161
Syntax Diagram.....	161
Sample Views.....	161
Usage Notes	162
Union & Union All.....	162
Intersect & Intersect All.....	162
Except & Except All.....	162
Precedence Rules.....	163
View Usage.....	164
Outer Join Usage.....	164
SUMMARY TABLES.....	165
Summary Table Types.....	165
IBM Implementation	166
DDL Restrictions.....	166
Definition Only Summary Tables.....	167
Refresh Deferred Summary Tables.....	167
Refresh Immediate Summary Tables.....	168
Usage Notes and Restrictions.....	169
Multi-table Summary Tables.....	170
Indexes on Summary Tables.....	172
Roll Your Own.....	173
Inefficient Triggers.....	173
Efficient Triggers.....	176
IDENTITY COLUMN	183
Usage Notes	183
Rules and Restrictions.....	184
Restart the Identity Start Value.....	184
Cache Usage.....	185
Gaps in the Sequence.....	186
Roll Your Own - no Gaps in Sequence.....	186
IDENTITY_VAL_LOCAL Function.....	187
RECURSIVE SQL	191
Use Recursion To.....	191
When (Not) to Use Recursion.....	191
How Recursion Works	191
List Dependents of AAA.....	192
Notes & Restrictions.....	193
Sample Table DDL & DML.....	193
Introductory Recursion.....	194
List all Children #1.....	194
List all Children #2.....	194
List Distinct Children.....	195
Show Item Level.....	195
Select Certain Levels.....	196
Select Explicit Level.....	197
Trace a Path - Use Multiple Recursions.....	197
Extraneous Warning Message.....	198

Logical Hierarchy Flavours	199
Divergent Hierarchy.....	199
Convergent Hierarchy	200
Recursive Hierarchy.....	200
Balanced & Unbalanced Hierarchies	201
Data & Pointer Hierarchies.....	201
Halting Recursive Processing	202
Sample Database.....	202
Stop After "n" Levels	203
Stop After "n" Levels - Remove Duplicates.....	203
Stop After "n" Levels - Show Data Paths.....	204
Stop After "n" Rows.....	205
Find all Children, Ignore Data Loops	205
Find all Children, Mark Data Loops.....	206
Find all Data Loops - Only.....	206
Stop if Data Loops.....	207
Working with Other Key Types.....	208
Stopping Simple Recursive Statements Using FETCH FIRST code.....	208
Clean Hierarchies and Efficient Joins.....	209
Introduction.....	209
Limited Update Solution	209
Full Update Solution	211
FUN WITH SQL	215
Creating Sample Data	215
Create a Row of Data.....	215
Create "n" Rows & Columns of Data	215
Linear Data Generation.....	216
Tabular Data Generation.....	216
Cosine vs. Degree - Table of Values	217
Make Reproducible Random Data.....	217
Make Random Data - Different Ranges.....	218
Make Random Data - Different Flavours	218
Make Random Data - Varying Distribution	219
Make Test Table & Data	219
Statistical Analysis using SQL	222
Sample Table DDL & DML.....	222
Mean.....	222
Median.....	223
Mode.....	223
Average Deviation	223
Variance	223
Standard Deviation.....	224
Data Distribution.....	224
Standard Error.....	225
Pearson Product-Moment Coefficient	225
Time-Series Data Usage.....	226
Find Overlapping Rows	226
Find Gaps in Time-Series	227
Show Each Day in Gap	229
Other Fun Things	229
Convert Character to Numeric	229
Convert Timestamp to Numeric	231
Selective Column Output	231
Making Charts Using SQL.....	231
Multiple Counts in One Pass.....	232
Multiple Counts from the Same Row	233
Find Missing Rows in Series / Count all Values	234
Normalize Denormalized Data	236
Denormalize Normalized Data	237
Reversing Field Contents.....	238
Stripping Characters.....	240
Query Runs for "n" Seconds	241
QUIRKS IN SQL	243
Trouble with Timestamps	243
No Rows Match	243
Dumb Date Usage.....	244
RAND in Predicate	245
Date/Time Manipulation	247
Use of LIKE on VARCHAR	248
Comparing Weeks.....	249
DB2 Truncates, not Rounds.....	249

CASE Checks in Wrong Sequence.....	250
Division and Average	250
Date Output Order.....	250
Ambiguous Cursors.....	251
Floating Point Numbers.....	252
Legally Incorrect SQL.....	254
DB2 DISLIKES.....	257
Performance Analysis.....	257
The DB2 Monitor	257
Win/NT Problems	258
Visual Explain	259
Other Whines.....	260
Statement Terminator.....	260
Stopping Recursion	260
RAND Function	261
SYSFUN Character Functions	262
STRIP Function	262
DB2BATCH Reliability.....	262
APPENDIX.....	263
DB2 Sample Tables	263
Class Schedule	263
Department.....	263
Employee.....	263
Employee Activity	264
Employee Photo.....	266
Employee Resume	266
In Tray	266
Organization.....	267
Project	267
Sales.....	268
Staff	268
BOOK BINDING	271
INDEX	271
READER COMMENTS.....	277

Introduction to SQL

This chapter contains a basic introduction to DB2 UDB SQL. It also has numerous examples illustrating how to use this language to answer particular business problems. However, it is not meant to be a definitive guide to the language. Please refer to the relevant IBM manuals for a more detailed description.

Syntax Diagram Conventions

This book uses railroad diagrams to describe the DB2 UDB SQL statements. The following diagram shows the conventions used.

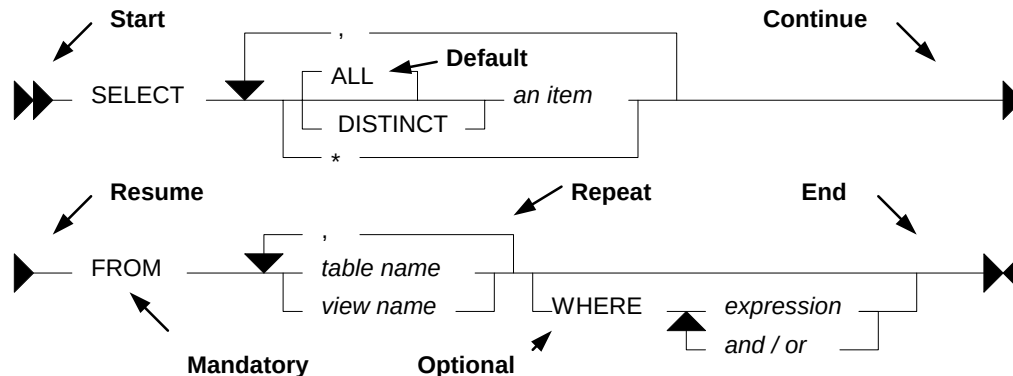


Figure 1, Syntax Diagram Conventions

Rules

- **Upper Case** text is a SQL keyword.
- **Italic** text is either a placeholder, or explained elsewhere.
- **Backward** arrows enable one to repeat parts of the text.
- A branch line going **above** the main line is the default.
- A branch line going **below** the main line is an optional item.

SQL Components

DB2 Objects

DB2 is a relational database that supports a variety of object types. In this section we shall overview those items which one can obtain data from using SQL.

Table

A table is an organized set of columns and rows. The number, type, and relative position, of the various columns in the table is recorded in the DB2 catalogue. The number of rows in the table will fluctuate as data is inserted and deleted.

The CREATE TABLE statement is used to define a table. The following example will define the EMPLOYEE table, which is found in the DB2 sample database.

```

CREATE TABLE EMPLOYEE
(EMPNO    CHARACTER (00006)    NOT NULL
, FIRSTNME VARCHAR (00012)    NOT NULL
, MIDINIT  CHARACTER (00001)    NOT NULL
, LASTNAME VARCHAR (00015)    NOT NULL
, WORKDEPT CHARACTER (00003)
, PHONENO  CHARACTER (00004)
, HIREDATE DATE
, JOB      CHARACTER (00008)
, EDLEVEL  SMALLINT            NOT NULL
, SEX      CHARACTER (00001)
, BIRTHDATE DATE
, SALARY   DECIMAL (00009,02)
, BONUS    DECIMAL (00009,02)
, COMM     DECIMAL (00009,02)
)
DATA CAPTURE NONE;

```

Figure 2, DB2 sample table - EMPLOYEE

View

A view is another way to look at the data in one or more tables (or other views). For example, a user of the following view will only see those rows (and certain columns) in the EMPLOYEE table where the salary of a particular employee is greater than or equal to the average salary for their particular department.

```

CREATE VIEW EMPLOYEE_VIEW AS
SELECT  A.EMPNO, A.FIRSTNME, A.SALARY, A.WORKDEPT
FROM    EMPLOYEE A
WHERE   A.SALARY >=
        (SELECT AVG(B.SALARY)
         FROM   EMPLOYEE B
         WHERE  A.WORKDEPT = B.WORKDEPT);

```

Figure 3, DB2 sample view - EMPLOYEE_VIEW

A view need not always refer to an actual table. It may instead contain a list of values:

```

CREATE VIEW SILLY (C1, C2, C3)
AS VALUES (11, 'AAA', SMALLINT(22))
, (12, 'BBB', SMALLINT(33))
, (13, 'CCC', NULL);

```

Figure 4, Define a view using a VALUES clause

Selecting from the above view works the same as selecting from a table:

```

SELECT  C1, C2, C3
FROM    SILLY
ORDER BY C1 ASC;

```

ANSWER		
=====		
C1	C2	C3
--	--	--
13	AAA	22
12	BBB	33
11	CCC	-

Figure 5, SELECT from a view that has its own data

We can go one step further and define a view that begins with a single value that is then manipulated using SQL to make many other values. For example, the following view, when selected from, will return 10,000 rows. Note however that these rows are not stored anywhere in the database - they are instead created on the fly when the view is queried.

```
CREATE VIEW TEST_DATA AS
WITH TEMP1 (NUM1) AS
  (VALUES (1)
   UNION ALL
   SELECT NUM1 + 1
   FROM   TEMP1
   WHERE  NUM1 < 10000)
SELECT *
FROM   TEMP1;
```

Figure 6, Define a view that creates data on the fly

Alias

An alias is an alternate name for a table or a view. Unlike a view, an alias can not contain any processing logic. No authorization is required to use an alias other than that needed to access to the underlying table or view.

```
CREATE ALIAS  EMPLOYEE_AL1 FOR EMPLOYEE;
COMMIT;

CREATE ALIAS  EMPLOYEE_AL2 FOR EMPLOYEE_AL1;
COMMIT;

CREATE ALIAS  EMPLOYEE_AL3 FOR EMPLOYEE_AL2;
COMMIT;
```

Figure 7, Define three aliases, the latter on the earlier

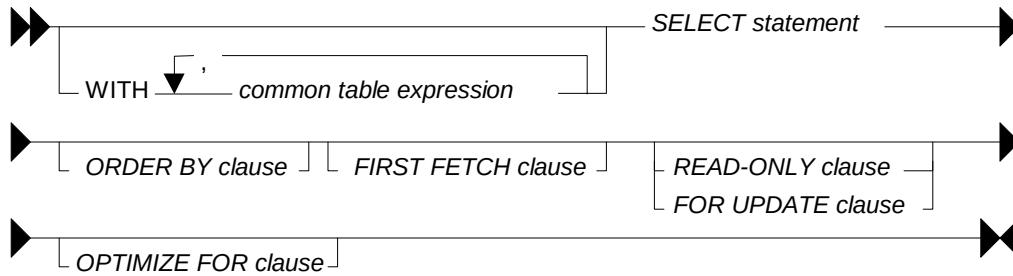
Neither a view, nor an alias, can be linked in a recursive manner (e.g. V1 points to V2, which points back to V1). Also, both views and aliases still exist after a source object (e.g. a table) has been dropped. In such cases, a view, but not an alias, is marked invalid.

SELECT Statement

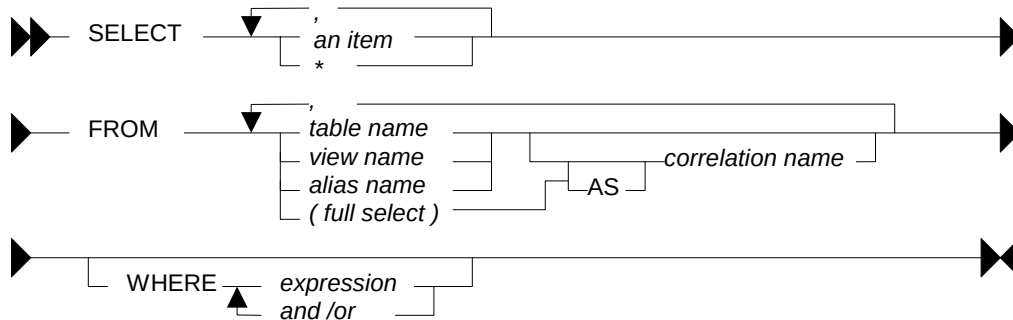
A SELECT statement is used to query the database. It has the following components, not all of which need be used in any particular query:

- **SELECT clause.** One of these is required, and it must return at least one item, be it a column, a literal, the result of a function, or something else. One must also access at least one table, be that a true table, a temporary table, a view, or an alias.
- **WITH clause.** This clause is optional. Use this phrase to include independent SELECT statements that are subsequently accessed in a final SELECT (see page 26).
- **ORDER BY clause.** Optionally, order the final output (see page 115).
- **FETCH FIRST clause.** Optionally, stop the query after "n" rows (see page 17). If an *optimize-for* value is also provided, both values are used independently by the optimizer.
- **READ-ONLY clause.** Optionally, state that the query is read-only. Some queries are inherently read-only, in which case this option has no effect.
- **FOR UPDATE clause.** Optionally, state that the query will be used to update certain columns that are returned during fetch processing.
- **OPTIMIZE FOR n ROWS clause.** Optionally, tell the optimizer to tune the query assuming that not all of the matching rows will be retrieved. If a first-fetch value is also provided, both values are used independently by the optimizer.

Refer to the IBM manuals for a complete description of all of the above. Some of the more interesting options are described below.

Figure 8, *SELECT Statement Syntax (general)***SELECT Clause**

Every query must have at least one SELECT statement, and it must return at least one item, and access at least one object.

Figure 9, *SELECT Statement Syntax***SELECT Items**

- **Column:** A column in one of the table being selected from.
- **Literal:** A literal value (e.g. "ABC"). Use the AS expression to name the literal.
- **Special Register:** A special register (e.g. CURRENT TIME).
- **Expression:** An expression result (e.g. MAX(COL1*10)).
- **Full Select:** An embedded SELECT statement that returns a single row.

FROM Objects

- **Table:** Either a permanent or temporary DB2 table.
- **View:** A standard DB2 view.
- **Alias:** A DB2 alias that points to a table, view, or another alias.
- **Full Select:** An embedded SELECT statement that returns a set of rows.

Sample SQL

```
SELECT  DEPTNO
        ,ADMRDEPT
        , 'ABC' AS ABC
FROM    DEPARTMENT
WHERE   DEPTNAME LIKE '%ING%'
ORDER BY 1;
```

```
ANSWER
=====
DEPTNO ADMRDEPT ABC
-----
B01     A00      ABC
D11     D01      ABC
```

Figure 10, *Sample SELECT statement*

To select all of the columns in a table (or tables) one can use the "*" notation:

<pre>SELECT * FROM DEPARTMENT WHERE DEPTNAME LIKE '%ING%' ORDER BY 1;</pre>	<pre>ANSWER (part of) ===== DEPTNO etc... ----->>> B01 PLANNING D11 MANUFACTU</pre>
---	--

Figure 11, Use "*" to select all columns in table

To select both individual columns, and all of the columns (using the "*" notation), in a single SELECT statement, one can still use the "*", but it must fully-qualified using either the object name, or a correlation name:

<pre>SELECT DEPTNO ,DEPARTMENT.* FROM DEPARTMENT WHERE DEPTNAME LIKE '%ING%' ORDER BY 1;</pre>	<pre>ANSWER (part of) ===== DEPTNO DEPTNO etc... ----->>> B01 B01 PLANNING D11 D11 MANUFACTU</pre>
--	---

Figure 12, Select an individual column, and all columns

Use the following notation to select all the fields in a table twice:

<pre>SELECT DEPARTMENT.* ,DEPARTMENT.* FROM DEPARTMENT WHERE DEPTNAME LIKE '%NING%' ORDER BY 1;</pre>	<pre>ANSWER (part of) ===== DEPTNO etc... ----->>> B01 PLANNING</pre>
---	---

Figure 13, Select all columns twice

FETCH FIRST Clause

The fetch first clause limits the cursor to retrieving "n" rows. If the clause is specified and no number is provided, the query will stop after the first fetch.

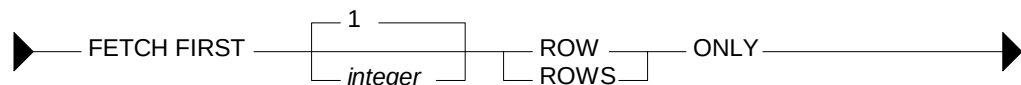


Figure 14, Fetch First clause Syntax

If this clause is used, and there is **no** ORDER BY, then the query will simply return a random set of matching rows, where the randomness is a function of the access path used and/or the physical location of the rows in the table:

<pre>SELECT YEARS ,NAME ,ID FROM STAFF FETCH FIRST 3 ROWS ONLY;</pre>	<pre>ANSWER ===== YEARS NAME ID ----- 7 Sanders 10 8 Pernal 20 5 Marenghi 30</pre>
---	--

Figure 15, FETCH FIRST without ORDER BY, gets random rows

WARNING: Using the FETCH FIRST clause to get the first "n" rows can sometimes return an answer that is not what the user really intended. See below for details.

If an ORDER BY is provided, then the FETCH FIRST clause can be used to stop the query after a certain number of what are, perhaps, the most desirable rows have been returned. However, the phrase should only be used in this manner when the related ORDER BY uniquely identifies each row returned.

To illustrate what can go wrong, imagine that we wanted to query the STAFF table in order to get the names of those three employees that have worked for the firm the longest - in order to give them a little reward (or possibly to fire them). The following query could be run:

SELECT	YEARS	ANSWER
	, NAME	=====
	, ID	YEARS NAME ID
FROM	STAFF	-----
WHERE	YEARS IS NOT NULL	13 Graham 310
ORDER BY	YEARS DESC	12 Jones 260
FETCH FIRST 3 ROWS ONLY;		10 Hanes 50

Figure 16, *FETCH FIRST* with *ORDER BY*, gets wrong answer

The above query answers the question correctly, but the question was wrong, and so the answer is wrong. The problem is that there are **two** employees that have worked for the firm for ten years, but only one of them shows, and the one that does show was picked at random by the query processor. This is almost certainly not what the business user intended.

The next query is similar to the previous, but now the ORDER ID uniquely identifies each row returned (presumably as per the end-user's instructions):

SELECT	YEARS	ANSWER
	, NAME	=====
	, ID	YEARS NAME ID
FROM	STAFF	-----
WHERE	YEARS IS NOT NULL	13 Graham 310
ORDER BY	YEARS DESC	12 Jones 260
	, ID DESC	10 Quill 290
FETCH FIRST 3 ROWS ONLY;		

Figure 17, *FETCH FIRST* with *ORDER BY*, gets right answer

WARNING: Getting the first "n" rows from a query is actually quite a complicated problem. Refer to page 58 for a more complete discussion.

Correlation Name

The **correlation name** is defined in the FROM clause and relates to the preceding object name. In some cases, it is used to provide a short form of the related object name. In other situations, it is required in order to uniquely identify logical tables when a single physical table is referred to twice in the same query. Some sample SQL follows:

SELECT	A.EMPNO	ANSWER
	, A.LASTNAME	=====
FROM	EMPLOYEE A	EMPNO LASTNAME
	, (SELECT MAX(EMPNO) AS EMPNO	-----
	FROM EMPLOYEE) AS B	000340 GOUNOT
WHERE	A.EMPNO = B.EMPNO;	

Figure 18, *Correlation Name usage example*

SELECT	A.EMPNO	ANSWER
	, A.LASTNAME	=====
	, B.DEPTNO AS DEPT	EMPNO LASTNAME DEPT
FROM	EMPLOYEE A	-----
	, DEPARTMENT B	000090 HENDERSON E11
WHERE	A.WORKDEPT = B.DEPTNO	000280 SCHNEIDER E11
AND	A.JOB <> 'SALESREP'	000290 PARKER E11
AND	B.DEPTNAME = 'OPERATIONS'	000300 SMITH E11
AND	A.SEX IN ('M', 'F')	000310 SETRIGHT E11
AND	B.LOCATION IS NULL	
ORDER BY	1;	

Figure 19, *Correlation Name usage example*

Renaming Fields

The AS phrase can be used in a SELECT list to give a field a different name. If the new name is an invalid field name (e.g. contains embedded blanks), then place the name in quotes:

SELECT	EMPNO	AS	E_NUM		ANSWER
	,MIDINIT	AS	"M INT"		=====
	,PHONENO	AS	"..."		E_NUM M INT ...
FROM	EMPLOYEE				-----
WHERE	EMPNO <	'000030'			000010 I 3978
ORDER BY	1;				000020 L 3476

Figure 20, Renaming fields using AS

The new field name must not be qualified (e.g. A.C1), but need not be unique. Subsequent usage of the new name is limited as follows:

- It can be used in an order by clause.
- It cannot be used in other part of the select (where-clause, group-by, or having).
- It cannot be used in an update clause.
- It is known outside of the full-select of nested table expressions, common table expressions, and in a view definition.

CREATE VIEW EMP2 AS	
SELECT EMPNO	AS E_NUM
	,MIDINIT AS "M INT"
	,PHONENO AS "..."
FROM	EMPLOYEE;
SELECT *	
FROM	EMP2
WHERE	"..." = '3978';

ANSWER
=====
E_NUM M INT ...

000010 I 3978

Figure 21, View field names defined using AS

Working with Nulls

In SQL something can be **true**, **false**, or **null**. This three-way logic has to always be considered when accessing data. To illustrate, if we first select all the rows in the STAFF table where the SALARY is < \$10,000, then all the rows where the SALARY is >= \$10,000, we have not necessarily found all the rows in the table because we have yet to select those rows where the SALARY is null.

The presence of null values in a table can also impact the various column functions. For example, the AVG function ignores null values when calculating the average of a set of rows. This means that a user-calculated average may give a different result from a DB2 calculated equivalent:

SELECT	AVG(COMM)	AS	A1		ANSWER
	,SUM(COMM) / COUNT(*)	AS	A2		=====
FROM	STAFF				A1 A2
WHERE	ID <	100;			-----
					796.025 530.68

Figure 22, AVG of data containing null values

Null values can also pop in columns that are defined as NOT NULL. This happens when a field is processed using a column function and there are no rows that match the search criteria:

```

SELECT    COUNT(*)      AS NUM
          , MAX(LASTNAME) AS MAX
FROM      EMPLOYEE
WHERE     FIRSTNAME = 'FRED';

```

ANSWER	
=====	
NUM	MAX
---	---
0	-

Figure 23, Getting a NULL value from a field defined NOT NULL

Why Nulls Exist

Null values can represent two kinds of data. In first case, the value is **unknown** (e.g. we do not know the name of the person's spouse). Alternatively, the value is **not relevant** to the situation (e.g. the person does not have a spouse).

Many people prefer not to have to bother with nulls, so they use instead a special value when necessary (e.g. an unknown employee name is blank). This trick works OK with character data, but it can lead to problems when used on numeric values (e.g. an unknown salary is set to zero).

Locating Null Values

One can not use an equal predicate to locate those values that are null because a null value does not actually equal anything, not even null, it is simply null. The IS NULL or IS NOT NULL phrases are used instead. The following example gets the average commission of only those rows that are not null. Note that the second result differs from the first due to rounding loss.

```

SELECT    AVG(COMM)      AS A1
          , SUM(COMM) / COUNT(*) AS A2
FROM      STAFF
WHERE     ID < 100
AND       COMM IS NOT NULL;

```

ANSWER	
=====	
A1	A2
-----	-----
796.025	796.02

Figure 24, AVG of those rows that are not null

SQL Predicates

A predicate is used in either the WHERE or HAVING clauses of a SQL statement. It specifies a condition that true, false, or unknown about a row or a group.

Basic Predicate

A basic predicate compares two values. If either value is null, the result is unknown. Otherwise the result is either true or false.

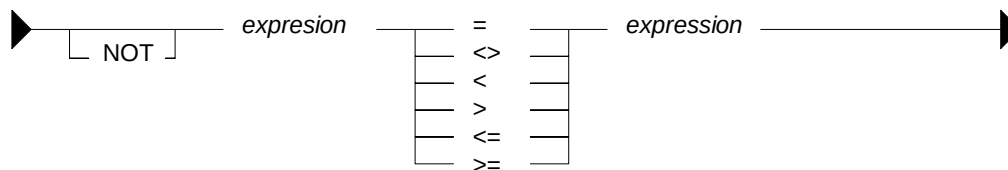


Figure 25, Basic Predicate syntax