

Rocket Model 204 SQL Server

User's Guide

Version 7 Release 4.0

May 2012
204-0704-SQLUG-01



Notices

Edition

Publication date: May 2012

Book number: 204-0704-SQLUG-01

Product version: Rocket Model 204 SQL Server User's Guide Version 7 Release 4.0

Copyright

© Computer Corporation of America 1989-2012. All Rights Reserved.

Computer Corporation of America is a wholly-owned subsidiary of Rocket Software, Inc.

Trademarks

Rocket is a registered trademark of Rocket Software, Inc. For a list of Rocket registered trademarks go to: www.rocketsoftware.com/about/legal. All other products or services mentioned in this document may be covered by the trademarks, service marks, or product names of their respective owners.

License agreement

This software and the associated documentation are proprietary and confidential to Rocket Software, Inc., are furnished under license, and may be used and copied only in accordance with the terms of such license.

Note

This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulation should be followed when exporting this product.

Contact information

Web Site: www.rocketsoftware.com

Rocket Software, Inc. Headquarters
77 4th Avenue, Suite 100
Waltham, MA 02451-1468
USA
Tel: +1.617.614.4321
Fax: +1.617.630.7100

Contacting Technical Support

If you have current support and maintenance agreements with Rocket Software and CCA, contact Rocket Software Technical support by email or by telephone:

Email: m204support@rocketsoftware.com

Telephone :

North America +1.800.755.4222

United Kingdom/Europe +44 (0) 20 8867 6153

Alternatively, you can access the Rocket Customer Portal and report a problem, download an update, or read answers to FAQs. You will be prompted to log in with the credentials supplied as part of your product maintenance agreement.

To log in to the Rocket Customer Portal:

Go to <http://www.rocketsoftware.com/support> and click **Rocket M204**.

Contents

About this Guide

Audience	xiii
Model 204 documentation set	xiii
Documentation conventions	xiv

1 Introduction to the Model 204 SQL Server

In this chapter.....	1
Overview	1
Model 204 SQL processing configurations	1
How the SQL Server works within Model 204	2
SQL Server provides seamless operation.....	2
SQL Server operates concurrently	3
SQL DML and SQL DDL may be executed simultaneously	3
SQL DDL and Model 204 DDL are independent.....	3
File preparation is minimal	3
Field attribute functionality is available.....	4
SQL processing adds to Model 204 Online requirements.....	4
SQL processing relies on SQL security.....	4
Model 204 SQL processing components	5
SQL Server components	7
SQL Server associated software.....	7
SQL Server supporting tools	7
SQL intersystem processing interfaces.....	8
Model 204 SQL standards	8
Model 204 SQL clients	9
SQL processing from the PC client	9

2 Model 204 SQL Catalog

In this chapter.....	11
Overview	11
Surveying the SQL catalog	11
Bridge to Model 204 data	11
Model 204 SQL catalog characteristics.....	13
Using the SQL catalog	13
Populating the catalog	14
Reporting catalog contents	15
Monitoring catalog consistency	15
Maintaining the SQL catalog	16
Creating the CCACAT file	17
Including CCACAT in an Online.....	17
Ongoing CCACAT maintenance	18
CCACAT implementation for BLOB and CLOB data.....	19

3 Mapping Model 204 Data to SQL

In this chapter.....	21
Overview	21
Representing Model 204 data in SQL	21
Changing existing Model 204 files	21
SQL pattern search guidelines	23
Using Model 204 file data features.....	23
Using PRIMARY KEY table columns	25
Model 204 and SQL data extraction mismatches	26
Mapping multiply occurring fields to nested tables.....	27
Understanding nested tables	27
Translating multiply occurring fields	28
Simulating normalization of Model 204 record data	28
Handling foreign keys.....	30
Handling primary keys.....	31
Matching Model 204 and SQL data formats.....	31
Compatibility of Model 204 and SQL data formats	31
Optimizing Model 204 data retrieval.....	32
Optimizing Model 204 data conversion	35
How Model 204 SQL processes dirty data	37
Handling NOT NULL, UNIQUE, and multiply occurring data	38
Handling mixed numeric and nonnumeric data	39
Observing data precision limits	39
Converting SQL data types for display.....	41
LOB fields in SQL statements	42

4 Model 204 SQL Data Definition Language

In this chapter.....	45
Overview	46
Model 204 SQL DDL statements	46
Model 204 SQL DDL extensions	48
Creating SQL objects.....	48
Authorization ID is equivalent to Model 204 user ID	49
Model 204 SQL table types	49
Statement ordering is important	49
Naming SQL objects	50
Creating schemas	50
CREATE SCHEMA statement.....	50
Indicating schema name and owner	51
Creating tables	52
CREATE TABLE statement.....	52
Mapping table names to file names	53
Using CLOB or BLOB data	55
Defining columns.....	55
Column definition statement.....	55
Mapping columns to Model 204 fields	57
Column naming and the SYSNAME extension	58
Specifying a multicolumn UNIQUE key	58
Creating nested tables	62

Nested table statements	62
Mapping multiply occurring groups	65
Nested tables require a foreign key	65
Nested tables require a referential constraint definition	65
CASCADE is the only referential triggered action	66
Using system-generated keys	67
Creating views	69
CREATE VIEW statement.....	70
Rules for updating views	70
Guideline for view definitions	71
Using SQL views in Model 204 SQL DDL	71
Simulating file groups	72
Mapping files with mixed record types	73
Maintaining views.....	74
Querying views	74
Setting the schema and user context	75
Determining the default schema context.....	75
Prefixing the schema name to an SQL object	75
Using SET SCHEMA	76
Using SET USER	76
Altering SQL objects	77
ALTER TABLE statement	77
Using ADD column	78
Using DROP column	78
Using MODIFY column	79
Dropping SQL objects	79
Dropping tables	80
Dropping views	80
Dropping schemas	80
Granting privileges for SQL objects	81
GRANTs are for adding privileges	81
GRANT and REVOKE handle nearly all SQL security	81
GRANT statement.....	81
REVOKE statement	82
Granting and altering column UPDATE privileges	83
Column UPDATE examples.....	84
DDL statement-level security	86
Model 204 SQL view privileges.....	89
SQL statement security example	89
Statement security example comments	93
SQL DDL processing	95

5 Creating DDL with the Table Specification Facility

In this chapter.....	97
Overview	97
Introduction to the Table Specification facility (TSF)	98
DDL processing.....	98
TSF processing sequence	99
DDL statements generated by the TSF.....	99
Model 204 SQL DDL extensions generated by the TSF	100

SQL and Model 204 data consistency	101
Using TSF panels.....	101
Panel conventions	101
ENTER and PF key conventions	103
Logging in.....	103
Creating or modifying a base table (Main Menu panel)	104
Creating SQL objects in the context of a schema	104
Schema Authorization	105
SQL Table Name	105
Schema Name	106
Model 204 File Name and Password	106
Table Type	106
Primary Key.....	107
Parent Table	107
Keeping or deleting the pending definition	107
Defining nested tables	108
Defining column names (Column List panel)	108
Model 204 field names	111
Changing the order of field names	112
Defining SQL column names.....	112
Defining column attributes (Column Attributes panel).....	113
Specifying attributes.....	115
Nulls	115
Format	116
Len (length)	117
Prec (precision)	117
Scale	117
Usage note	117
Nonstandard PF key functions	118
Completing table definitions (Completion panel).....	118
Completion panel functions	119
Selection	120
“USE” Cmd Arg	120
Error condition	120
Defining multicolumn unique keys (Multi-Column Unique panel).....	120
Specifying a multicolumn unique key	121
Model 204 Field.....	122
Column Name(s)	123
Usage note	123
Specifying GRANT authority (Grant Authority panel).....	123
Authority	124
Column(s).....	124
User.....	124
Grant option	125
Usage note	125
Viewing DDL at the terminal (Completion panel)	125
Generating DDL to an output file (Completion panel)	127

6 Getting Information from the SQL Catalog

In this chapter.....	129
----------------------	-----

Overview	129
SQL catalog reporting with CCACATREPT	129
Logging in.....	130
CCACATREPT Main Menu	131
Using the CCACATREPT panel	131
Selection field.....	132
Schema Name field	132
Authorization ID field	133
Table/View Name field	133
DDL Statement Type(s) field	133
Grantee field.....	133
“USE” Command Arg	134
Report Selection 1: Generate DDL	134
Specifying report input parameters	134
Report input parameter examples	135
Ordering the DDL output	136
Sample of generated DDL.....	136
Report Selection 2: Formatted Table/View report	137
Report input parameters	138
Contents of the report	138
Sample report.....	138
Report Selection 3: Privilege report by table/view	139
Report input parameters	139
Report display fields.....	140
Sample Privilege Report by table and view.....	140
Report Selection 4: Privilege report by grantee	142
Report input parameter	142
Report display fields.....	142
Sample report.....	142
Querying the SQL catalog.....	143
Querying a CATALOG view	143
Rules for CATALOG queries	145
SCHEMAS view	145
TABLES view	146
TABLE_COLUMNS view	147
COLUMNS view	148
VIEWS view	149
TABLE_CONSTRAINTS view	150
KEY_COLUMN_USAGE view.....	150
TABLE_PRIVILEGES view	151
COLUMN_PRIVILEGES view	152
ODBC_TYPES view.....	153
ODBC_SCALES view.....	154
CONST view	154
ODBC_COLUMNS view.....	155
ODBC_TABLES view.....	156
ODBC_SPECIAL_COLS view.....	156
ODBC_TABLE_STATS view.....	157
ODBC_KEY_STATS view	158
ODBC_STATISTICS view	159

7 Model 204 SQL Data Manipulation Language

In this chapter.....	161
Overview	161
Using Model 204 SQL DML	162
Maintaining data definition consistency	162
DML statement privileges.....	162
Setting SQL isolation level	162
Executing SQL DML and DDL simultaneously	163
Mixing SQL DML and DDL.....	163
Using SET SCHEMA and SET USER.....	164
Using SQL DML against INVISIBLE fields	164
Using SQL columns mapped to INVISIBLE fields.....	164
Using SQL DML against nested tables	168
Sample file and SQL mapping	168
DML example series.....	170
Retrieving a particular occurrence of a multiply occurring group	170
Retrieving a range or series of occurrences.....	171
Retrieving any or all occurrences based on a condition.....	172
Retrieving at least n occurrences based on a condition.....	172
Correlating a table and a nested table	173
Working with nested table constraints	173
Porting nested table applications	174
Options in the SELECT LIST statement.....	174
Correlation name feature	174
Wildcard asterisk (*) for an individual table feature.....	174
CURRENT_TIME keyword.....	175
CURRENT_DATE keyword.....	175
CURRENT_TIMESTAMP keyword	175
USER keyword.....	175
SQLVERSION keyword.....	175
SQLERROR keyword.....	176
SQLSTATE keyword	176
SQL INNER JOIN features.....	176
CROSS JOIN feature	176
NATURAL JOIN feature	176
ON clause feature	177
USING clause feature	177
SQL OUTER JOIN features	177
SQL outer join features expanded	178

A Model 204 SQL DDL Syntax

In this appendix	181
Overview	181
DDL syntax.....	182
Notes for syntax display	185

B Model 204 SQL Reserved Words

In this appendix	187
Overview	187

Reserved words	188
----------------------	-----

C SQL DDL Mapping of the Demonstration Database

In this appendix	189
Overview	189
DDL stream	190
CLIENTS table	190
VEHICLES table.....	191
CLAIMS03 table	192
VIEWS against the CLIENTS table	193

Index

About this Guide

The *Model 204 SQL Server User's Guide* contains file and system management information for accessing a Model 204 database with Structured Query Language (SQL) statements. The manual describes how to use the Model 204 SQL catalog and the proprietary features of Model 204 SQL DDL and DML.

Audience

This manual is primarily for users of Connect★ who are responsible for:

- Model 204 file and system management
- SQL DDL definition of Model 204 files

SQL DML application programmers might also find this manual useful.

Except where noted, ANSI SQL 1989 standard and some SQL 1992 standard functionality and user knowledge of that functionality are assumed throughout the manual. In addition, knowledge of standard Model 204 terminology and functionality is assumed.

Model 204 documentation set

The complete commercially released documentation for the latest version of Model 204 is available for download from the Rocket M204 customer portal.

To access the Rocket Model 204 documentation:

1. Navigate to:
<http://www.rocketsoftware.com/m204>
2. From the drop-down menu, select **Products > Model 204 > Documentation**.
3. Click the link to the current release and select the document you want from the list.
4. Click the .zip file containing the document.
5. Choose whether to open or save the document:
 - Select **Open** and double-click the pdf file to open the document.
 - Select **Save as** and select a location to save the zip file to.

Documentation conventions

This manual uses the following standard notation conventions in statement syntax and examples:

Convention	Description
TABLE	Uppercase represents a keyword that you must enter exactly as shown.
TABLE <i>tablename</i>	In text, italics are used for variables and for emphasis. In examples, italics denote a variable value that you must supply. In this example, you must supply a value for <i>tablename</i> .
READ [SCREEN]	Square brackets ([]) enclose an optional argument or portion of an argument. In this case, specify READ or READ SCREEN.
UNIQUE PRIMARY KEY	A vertical bar () separates alternative options. In this example, specify either UNIQUE or PRIMARY KEY.
TRUST <u>NOTRUST</u>	Underlining indicates the default. In this example, NOTRUST is the default.
IS {NOT LIKE}	Braces ({ }) indicate that one of the enclosed alternatives is required. In this example, you must specify either IS NOT or IS LIKE.
item ...	An ellipsis (. . .) indicates that you can repeat the preceding item.
item ,...	An ellipsis preceded by a comma indicates that a comma is required to separate repeated items.
All other symbols	In syntax, all other symbols (such as parentheses) are literal syntactic elements and must appear as shown.
<i>nested-key</i> ::= <i>column_name</i>	A double colon followed by an equal sign indicates an equivalence. In this case, <i>nested-key</i> is equivalent to <i>column_name</i> .
Enter your account: sales11	In examples that include both system-supplied and user-entered text, or system prompts and user commands, boldface indicates what you enter. In this example, the system prompts for an account and the user enters sales11 .
File > Save As	A right angle bracket (>) identifies the sequence of actions that you perform to select a command from a pull-down menu. In this example, select the Save As command from the File menu.
EDIT	Partial bolding indicates a usable abbreviation, such as E for EDIT in this example.

1

Introduction to the Model 204 SQL Server

In this chapter

- Overview
- How the SQL Server works within Model 204
- Model 204 SQL processing components
- Model 204 SQL standards
- Model 204 SQL clients

Overview

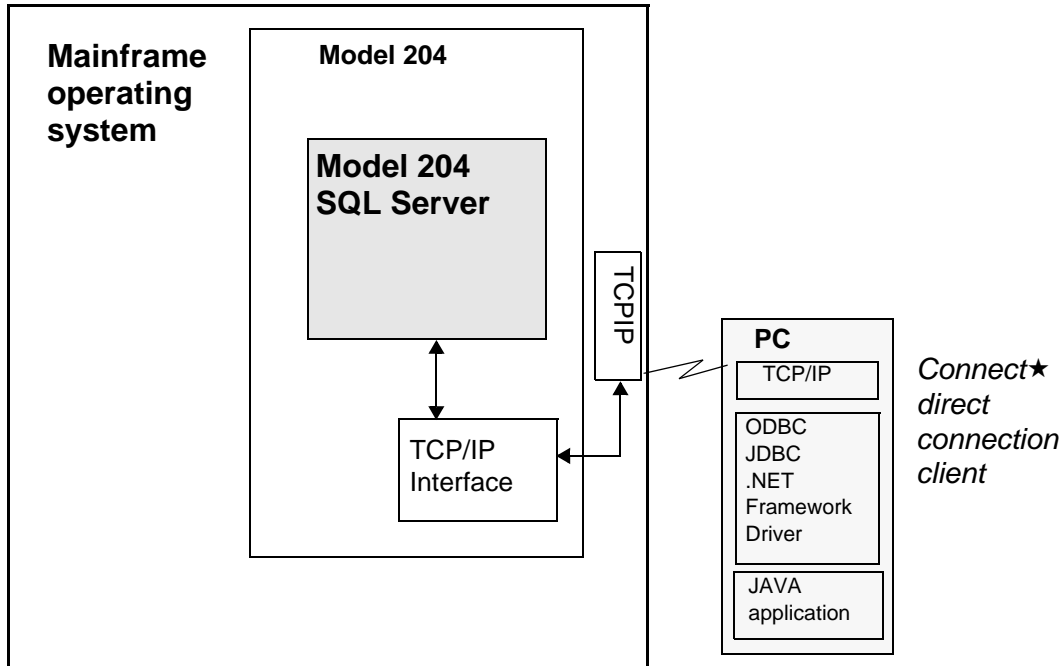
Model 204 now provides industry standard SQL access to Model 204 data through client-server technology. The Model 204 SQL Server provides full SQL processing in the basic Model 204 address space or virtual machine (ONLINE module) in the z/OS, z/VSE, and VM operating systems. The client-server architecture allows the Model 204 SQL Server to service networked PC clients.

Model 204 SQL processing configurations

SQL processing for PC clients is provided by the Model 204 Connect★ Suite. Figure 1-1 on page 2 illustrates the PC clients that can be configured with the Connect★ 32-bit ODBC Model 204 driver, using a TCP/IP connection to PC clients. SQL processing for PC clients is also

provided by JDBC for Model 204 or .NET Framework. See *Connect★ Suite Installation and Programming*.

Figure 1-1. Model 204 SQL Server and clients



How the SQL Server works within Model 204

Model 204 SQL processing is a Model 204 access method such as User Language or the Host Language Interface (HLI). The SQL Server invokes Model 204 DBMS operations and provides a combination of Model 204 and SQL database functionality. This section discusses some of the characteristics of the SQL Server and Model 204 interaction.

SQL Server provides seamless operation

After installing Model 204 and bringing up the Model 204 Online, preparation for SQL access requires only that you define a mapping of your Model 204 files to the SQL catalog. SQL Server processing is activated automatically and functions transparently to provide responses to client requests.

The typical sequence of tasks required to access the Model 204 SQL Server is the following:

- Install Model 204, running the appropriate installation jobs for the type of SQL client you are and the SQL utilities you require.
- Include in your Model 204 Online job the Model 204 SQL IODEV, RCL IODEV, and CCAIN parameters, sizing requirements, catalog file, and subsystems, and at least one TCP/IP thread definition.

- For SQL processing, populate the SQL catalog with SQL DDL definitions for your Model 204 files.
- Issue SQL DML for SQL processing; issue User Language commands, programs, and procedures for RCL processing.

SQL Server operates concurrently

The Model 204 SQL Server may operate concurrently and may share data files with User Language and the Host Language Interface. Model 204 coordinates record and resource locking among the three interfaces. The server takes advantage of Model 204 indexing and file organization efficiencies.

SQL DML and SQL DDL may be executed simultaneously

While Model 204 SQL Data Manipulation Language (DML) is being issued against Model 204 files, you can simultaneously run Model 204 SQL Data Definition Language (DDL) against the SQL catalog. However, SQL DDL may update only SQL objects that are not currently being accessed by SQL DML statements.

SQL DDL and Model 204 DDL are independent

Model 204 SQL DDL maps Model 204 databases to SQL tables by creating records in the SQL catalog. However, the SQL catalog is *not* active like a dictionary: SQL DDL execution does not cause Model 204 DDL to execute. Conversely, Model 204 DDL changes to files that are cataloged in the SQL catalog are not automatically reflected in the SQL catalog.

File preparation is minimal

The task of preparing your Model 204 files for SQL processing is largely that of defining your Model 204 files to the SQL catalog. Model 204 SQL processing uses the standard Model 204 file system, and supports all Model 204 file types.

The lone requirement for files used in SQL processing is that they be transaction backout (TBO) files. Model 204 automatically backs out incomplete transactions for TBO files when problems prevent the completion of a request. The TBO requirement guarantees that all uncommitted SQL updates can be backed out and prevents confusing transaction restrictions.

SQL supports Model 204 file groups with limitations. An SQL table can map to (and be as large as) at most a single Model 204 file. However, you can simulate file groups with SQL views and retrieve (but not update) data through the simulation. See “Simulating file groups” on page 72.

Field attribute functionality is available

Model 204 fields are mapped to SQL columns in the SQL catalog. The functionality of most Model 204 field attributes is available for Model 204 SQL processing. You must properly define the SQL columns corresponding to the Model 204 fields to make use of the attribute functionality. You can make use of the following field attributes:

KEY	STRING	INVISIBLE
ORDERED	FLOAT	OCCURS
NUMERIC RANGE	BINARY	
DEFERRABLE	CODED	
UNIQUE	LEN	

INVISIBLE fields are translated to SQL columns that can be used in certain circumstances to select data but cannot themselves be updated by SQL applications.

Only the first occurrences of multiply occurring OCCURS fields are accessible in SQL if mapped normally to an SQL column. For complete access, however, you can translate these fields into *nested tables*, a Model 204 SQL processing extension to standard SQL DDL.

SQL processing adds to Model 204 Online requirements

To support Model 204 SQL processing, you must make changes to the job or EXEC that brings up your Model 204 Online. These changes include the following:

- Additional file definition statements for the SQL catalog and for utilities that support the Model 204 SQL Server
- SQL and RCL IODEV thread definitions
- Adjustments to, and additional, CCAIN parameters
- SQL connection DEFINE commands for TCP/IP
- Model 204 SQL Server area size increase

SQL processing relies on SQL security

To access the Model 204 SQL Server, SQL processing clients must pass Model 204 login security and any external security software that is in effect. Two of the Model 204 SQL supporting utilities, CCATSF and CCACATREPT, are protected by Model 204 Application Subsystem security.

Model 204 field and record security are not represented and not enforced through the Model 204 SQL Server interface.

The primary security protection for issuing SQL DDL and DML is provided by SQL GRANT and REVOKE statements and SQL views. GRANT and REVOKE define who is allowed to perform a given operation on a given SQL object. Views allow you to define subsets of the database to which you can selectively grant access.

You can also replace Model 204 SQL security with privilege checking by an external security package. You provide user exits to the security package in a Model 204-defined format, as described in the *Model 204 Security Interfaces Manual*.

Model 204 SQL processing components

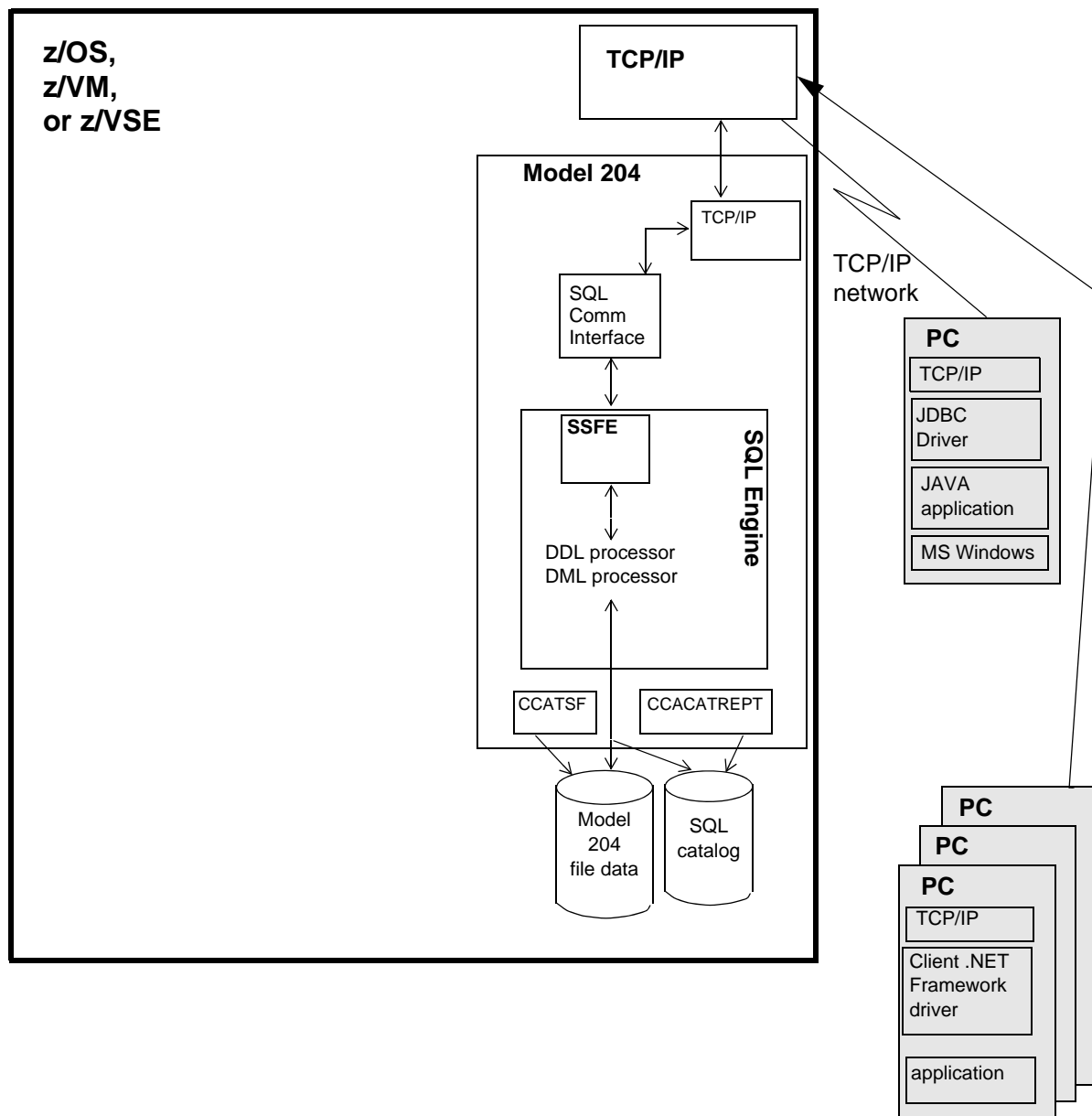
The Model 204 SQL Server integrates SQL processing into the core of Model 204. The SQL Server processes SQL DDL and DML from workstation interfaces.

The SQL Server works along with supporting software that provides data definition, packaging, and transfer, and has optional tools that simplify client use of the SQL Server.

The configuration of the SQL processing components when Model 204 is running under z/OS, z/VM, or z/VSE is shown in Figure 1-2 on page 6.

Note: Figure 1-2 illustrates the PC clients that can be configured with the Connect★ Suite JDBC driver and the .NET Framework driver. Only a TCP/IP connection to a PC client can be configured.

Figure 1-2. Model 204 SQL processing components



SQL Server components

The following are the principal components of the Model 204 SQL Server. These components are typically transparent to a Connect★ client.

SQL Engine	Includes the SQL Compiler, Optimizer, and Evaluator. The Engine is responsible for compiling SQL syntax strings, checking SQL semantics, optimizing database access, generating code to accomplish SQL requests, and executing the generated code. Each SQL statement generates data, status information, or both for the requesting application.
SQL Server Front End (SSFE)	The presentation level, or access layer to the SQL Engine. Primarily, SSFE accepts client request packets, processes the requests in each packet for the SQL Engine, and returns the SQL processing results in the form of a result packet. Performing corresponding functions to the SSFE is the SQL Server Client Front End.

SQL Server associated software

The SQL Server works in conjunction with the following Model 204 software. Except for the SQL catalog, this software is transparent to the client:

SQL catalog	Houses the SQL catalog information for Model 204 files defined with SQL DDL. The CCACAT system file is created during installation of Model 204.
SQL communications interface	Receives SQL requests from TCP/IP and passes them to the SQL Server Front End, and the interface receives result packets from the SQL Server Front End and routes them back to the client. This interface is a Model 204 module used only for SQL processing.
SQL Client Front End (SCFE)	Is the counterpart to the SSFE, described above. SCFE groups and sends client requests and receives and distributes results to the client. SCFE is platform independent: it is available as a Model 204 module on the mainframe or included as part of the Connect★ workstation installation software.

SQL Server supporting tools

The following Model 204 tools support SQL Server processing:

Table Specification facility (TSF)	Provides an interactive, menu-driven facility for mapping existing Model 204 files to SQL tables and columns. The TSF generates a stream of DDL statements you can edit, transmit to a workstation and submit to the Connect★ Visual Interface. The TSF is available on the mainframe only.
------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Catalog Reporting facility (CCACATREPT)	Provides a menu-driven facility for generating DDL from and reports of the SQL catalog contents. With the CCACATREPT, you can review your SQL object definitions, names, and privileges or use DDL it generates to repopulate the SQL catalog. CCACATREPT is available on the mainframe only.
-----------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The following PC tool supports SQL server processing.

Connect★ Visual Interface	Submits SQL DDL streams to the SQL catalog and populates the catalog with your valid DDL definitions. The Connect★ Visual Interface is a component of the Connect Star Suite for Model 204 and is available only on a PC workstation. Note: Each DDL statement submitted to the CVI must terminate in with a semicolon (;).
---------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

SQL intersystem processing interfaces

Transport of the data between the SQL Server and its clients relies on TCP/IP. Model 204 TCP/IP, along with IBM's TCP/IP software on the mainframe, enables a TCP/IP network connection from Model 204 to PC clients.

Model 204 SQL standards

This section describes the SQL standards adhered to in this Model 204 SQL support.

ANSI SQL 1989 and 1992

Model 204 SQL provides all clients ANSI SQL 1989 support and some ANSI SQL 1992 support for SQL DDL and DML. Unless otherwise specified, references in this manual to standard SQL are to the ANSI SQL 1989 and standards.

SQL Server DDL and DML support includes some extensions to and omissions from the standard. These deviations from the syntax or functionality of the standard are described in Chapters 4 and 7.

Federal Information Processing Standards

SQL DDL and DML supported by the Model 204 SQL Server are compliant with the Federal Information Processing Standards (FIPS Pub 127-1).

Open Database Connectivity

Connect★ provides Level 1 compliance with Microsoft's Open Database Connectivity (ODBC) Interface. ODBC is a de facto industry standard, based on the SAG standard for an SQL Call Level Interface, for inter-application substitutability. The Connect★ ODBC feature lets Model 204 access data from

an ODBC-compliant Windows spreadsheet, application development, and word processing packages.

The Model 204 JDBC driver incorporates JDBC 2.5 with no extended functionality.

The Model 204 .NET Framework driver is .NET 2.0 compliant.

Model 204 SQL clients

This section provides brief overviews of the Model 204 SQL client configurations.

For the specific current client hardware and software configuration requirements for Model 204 SQL processing, see the Connect★ Suite and SQL documentation.

SQL processing from the PC client

Connect★ Suite gives PC clients SQL and Remote Command Line (RCL) access to Model 204 data.

PC clients

PC clients for Connect★ Suite must be using a workstation with a LAN connection to the mainframe that supports TCP/IP communications software.

Connect★ Suite application program interface

For Connect★ application program interfaces, PC clients can use the Model 204 32-bit ODBC driver, J204 JDBC driver, or Model204Client .NET Framework data provider to support SQL application programs.

2

Model 204 SQL Catalog

In this chapter

- Overview
- Surveying the SQL catalog
- Using the SQL catalog
- Maintaining the SQL catalog

Overview

The Model 204 SQL catalog is a facility for storage, retrieval, and modification of the data definition information for SQL objects and SQL users. The SQL catalog contains the SQL schema, table, column, and view definitions and privileges that map to the Model 204 file data accessed with SQL applications.

Surveying the SQL catalog

Bridge to Model 204 data

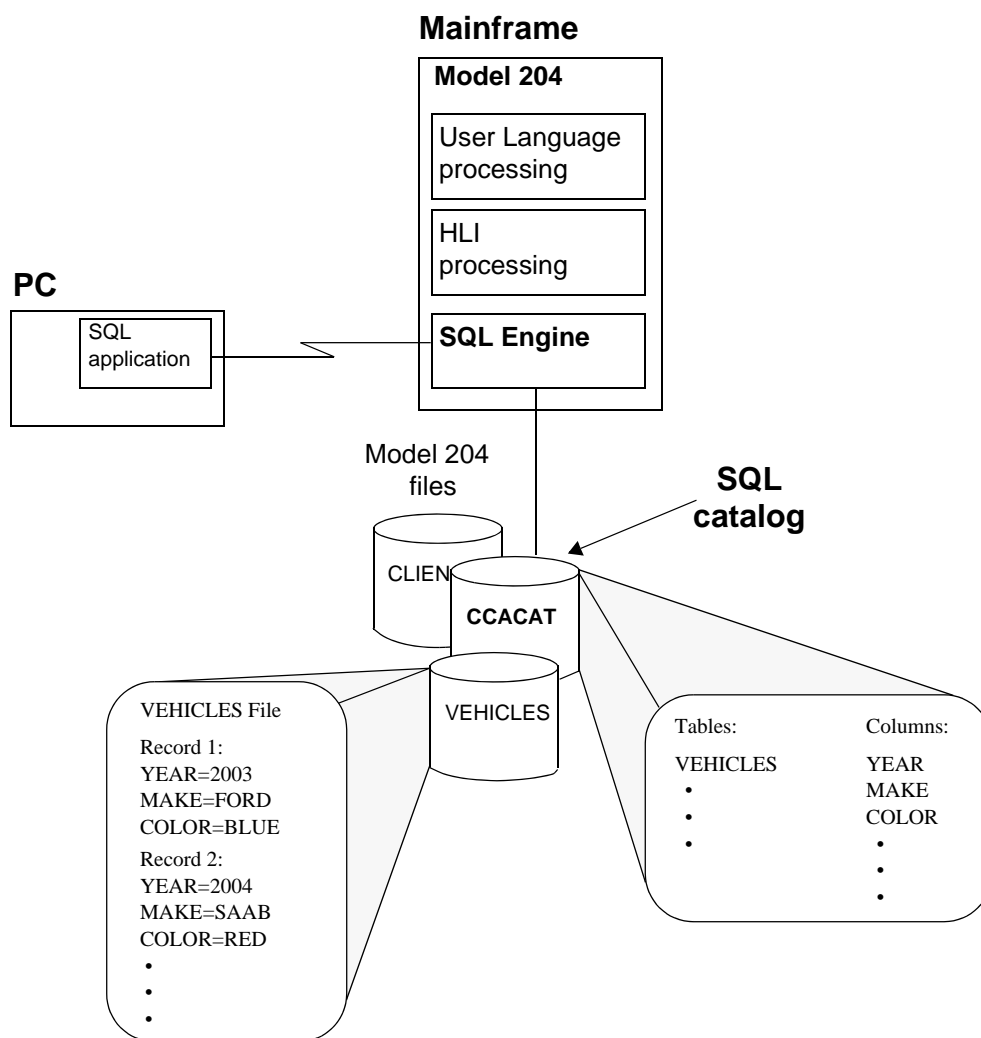
The SQL catalog is an essential bridge between your SQL application and Model 204 data. Figure 2-1 on page 12 shows the position and function of the SQL catalog in Model 204 SQL processing. The catalog is a Model 204 file that contains an SQL mapping of other Model 204 files.

Before you can run an SQL application against an existing Model 204 file, you must define the file's field descriptions to the Model 204 SQL

catalog. The SQL catalog maps Model 204 files to SQL tables and Model 204 fields to SQL columns. The catalog also stores SQL user privileges.

When you issue an SQL query against the Model 204 file data, the Model 204 SQL Server compiler and optimizer read the catalog and translate the query into a physical data request against the Model 204 files. The retrieved records are returned as SQL rows.

Figure 2-1. The SQL catalog is the CCACAT Model 204 file



Model 204 SQL catalog characteristics

The Model 204 SQL catalog has the following characteristics:

Model 204 SQL catalog is...	Because...
Model 204 file (called CCACAT)	As a Model 204 file, CCACAT is managed by Model 204 system and file managers and governed by Model 204 concurrency control and recovery.
Singular	Only one SQL system catalog is allowed per Model 204 Online. You cannot designate an alternate catalog or use multiple catalogs.
Queryable	SQL applications can access but not manipulate the catalog data. Eighteen nonupdateable views of the catalog data are available to all valid SQL users.
Not an active dictionary-type file	Changes you make to a cataloged Model 204 file are not automatically reflected in the SQL catalog. You must make corresponding changes to the SQL catalog to ensure that it is synchronized with the Model 204 file.
Populated and maintained by the Connect★ Visual Interface	Input into the CVI, a Connect★ workstation utility is SQL DDL (data definition language) you create or have generated. Note: All DDL statements input to the CVI must terminate with a semicolon(;).
Independent of the Model 204 Dictionary	You cannot create the SQL catalog with Dictionary facilities; CCATSF helps do this. Nor is SQL catalog information reflected in Dictionary metadata. However, you can use the Dictionary to operate on a Model 204 file that has been cataloged for SQL applications.
Repository of SQL security information.	You provide security definitions to the catalog through SQL GRANT and REVOKE statements. The SQL compiler reads the catalog security information to check the privileges for use of SQL objects.
Repository of SQL view definitions.	A view is an SQL table that is defined in terms of other SQL tables, columns, or views. These view definitions are stored in the SQL catalog and accessed like any other SQL object.

Using the SQL catalog

The Model 204 SQL catalog contains the SQL definition of the Model 204 files that can be accessed by SQL applications. The SQL catalog is loaded, or populated, with this SQL description of Model 204 files through the Connect★

Visual Interface utility (CVI). Other Model 204 SQL utilities are available to help you create the DDL for the catalog and to help you monitor the catalog contents.

Populating the catalog

You populate the SQL catalog by using the CVI utility to do the following:

- Define or update SQL objects in the catalog that map to Model 204 file data.
- Define views of, and the SQL security associated with, these objects.

The CVI utility updates the SQL catalog through DDL statements. The DDL used by CVI can come from the following sources:

- DDL that you generate with mainframe SQL utilities (the Table Specification facility, TSF, or the catalog reporting utility, CCACATREPT)
- DDL that you prepare manually

CVI is available to PC clients as part of the Connect★ Suite.

Generating DDL with SQL utilities

The TSF is an interactive Model 204 subsystem that generates DDL statements you can input to the CVI utility. Available to mainframe users, the TSF displays the current Model 204 file definitions and simplifies your specification of corresponding SQL characteristics. From these specifications, the TSF generates DDL statements defining SQL objects that map to the Model 204 files.

The TSF is described in detail in Chapter 5.

The CCACATREPT generates DDL statements from the SQL catalog that you can submit to CVI. The CCATREPT is:

- Discussed in “Reporting catalog contents” on page 15
- Described in Chapter 6.

Preparing DDL manually

You do not have to rely entirely on SQL utilities for the DDL stream you submit to the CVI utility. You can also prepare a DDL stream by:

- Modifying an SQL utility-generated DDL stream. For example, the Model 204 Table Specification facility (TSF) does not produce all DDL statements. You must manually add DDL statements to the TSF-generated DDL stream for certain operations.
- Writing your own stream of DDL statements.

Creating the DDL manually requires that you verify that the Model 204 file definition is consistent with the DDL. You must make sure, for example, that the file definition is current. In addition, you must ensure the proper mapping of Model 204 field attributes to SQL column data types, and you must be aware of proper Model 204 SQL DDL statement syntax and usage rules.

The TSF, however, reads the current data dictionary (Table A) information from the Model 204 file and displays it to you so you can build the SQL definition. Consequently, using the TSF is less likely to introduce inconsistencies between the SQL catalog definition and the actual Model 204 file data.

Reporting catalog contents

You can examine the contents of the Model 204 SQL catalog by:

- Querying the catalog directly
- Using CCACATREPT, the Model 204 SQL catalog reporting utility

The contents of the SQL catalog are represented in user accessible views that are defined in the schema CATALOG in the SQL catalog. Eighteen non-updateable views summarize the SQL catalog objects, naming the schemas, tables, views, columns, constraints, and privileges. Any authorized SQL user can access the views with an SQL query. For a description of each of the views and their contents, see “Querying the SQL catalog” on page 143.

With CCACATREPT, you can produce an online or printed report that provides information (including attributes and corresponding Model 204 file and field names) for any SQL tables or views defined in the SQL catalog. CCACATREPT also has reports for privilege information from the catalog. In addition, CCACATREPT can display DDL generated from the SQL catalog data and in the form of valid DDL syntax.

You can compare the CCACATREPT output to the Model 204 file definition (DISPLAY command output) to determine the updates you need to make to the SQL catalog to obtain consistency with the file. You can modify the report’s generated DDL and use it to repopulate your SQL catalog.

For further description of CCACATREPT and how to use it, see Chapter 6.

Monitoring catalog consistency

SQL data definitions in the catalog must correspond to the Model 204 file data they describe. Because the SQL catalog is not constructed to change automatically when Model 204 files are changed, it might not be in sync with the Model 204 file at the time an application is run. SQL DDL updates to the SQL catalog may be required to maintain correspondence between the file and the catalog definition.

The Model 204 TSF subsystem uses the Model 204 file definitions that are current at the time you invoke it. If no changes are made to the file contents

before you submit the TSF DDL to the SQL catalog, you can be sure that the catalog and the Model 204 file are consistent. The DDL that you create manually, however, is not checked for consistency with the Model 204 file. The person creating DDL manually must be aware of the current Model 204 file definition.

When you run the CVI utility, it provides error messages for any syntax errors, but it does not report data definitions that are inconsistent with the Model 204 file.

You are responsible for ensuring that if the Model 204 file definition changes, the corresponding SQL definitions are updated. To monitor SQL catalog and Model 204 file consistency, you can use direct SQL catalog queries or the Model 204 SQL catalog reporting utility, CCACATREPT. Both resources provide a display of the contents of the SQL catalog. Querying the catalog and using CCACATREPT are both described in Chapter 6.

Maintaining the SQL catalog

Creation and maintenance of the SQL catalog (the CCACAT file) implies responsibilities for both SQL-specific data management and Model 204-specific data management.

As the SQL catalog, CCACAT must be populated and updated through SQL.

As a Model 204 file, CCACAT is subject to Model 204 file and system management. CCACAT must be installed and created as part of Model 204 installation and must be maintained as part of the Model 204 system (including recovery and security).

As both SQL catalog and Model 204 file, CCACAT is protected both by SQL security and Model 204 security. SQL access to CCACAT is protected by Model 204 login security and by SQL GRANT and REVOKE security. User Language and Host Language Interface access to CCACAT is protected by normal Model 204 file security and by allowing only system managers to open CCACAT.

Because CCACAT is a system file with the prefix CCA, you need system manager privileges to create, open, or initialize it. Access to CCACAT for purposes other than normal SQL installation, operation, and reporting, and other than normal Model 204 operations like sizing, reorganizing, and recovery is highly discouraged.

Note: User Language statements or Model 204 file management commands must not be issued against CCACAT when SQL processing is taking place. CCACAT is a special Model 204 file not unlike CCASYS, for example. The integrity of your SQL processing depends on the integrity of CCACAT.

The rest of this section describes what the Model 204 system manager and file manager need to do to use and maintain the Model 204 file CCACAT.

Creating the CCACAT file

You create the CCACAT file as part of the mainframe installation of Model 204. For information about installing CCACAT, see the *Rocket Model 204 Installation Guide* for your operating system.

The procedure used in the CCACAT installation contains all the file parameter settings and DEFINE statements necessary to create a working version of CCACAT. The installation procedure does the following:

- Creates the Model 204 file CCACAT
- Defines fields in CCACAT
- Includes User Language procedures that load into CCACAT the SQL catalog definitions (schema, tables, views, columns) that are accessible to user queries

Rocket Software recommends that you use the CCACAT file parameters as installed. The reasons for some of the default parameter settings are discussed on the following pages.

Including CCACAT in an Online

This section summarizes CCACAT file considerations that affect the setting up of your Model 204 Online. For more complete information about Online job requirements for Model 204 SQL processing, see the *Model 204 SQL Connectivity Guide*.

Every Model 204 Online with Model 204 SQL applications must have exactly one CCACAT file allocated to it. Two Onlines can share a CCACAT file, but only in read-only mode. If CCACAT is shared, no user can update it with the SQL catalog populating utility, CVI.

When an SQL thread is initialized, CCACAT is opened automatically for the user. When the thread is terminated, CCACAT is closed. An explicit OPEN command for CCACAT can be issued only by a user with system manager privileges.

Job control requirements

Online job control must contain a DD statement or FILEDEF for the CCACAT file. CCACAT cannot be allocated or freed dynamically.

File access and security

The CVI utility is the usual way to update CCACAT. CVI has its own privilege mechanism, controlled by runtime parameters such as login ID.

The system manager can provide additional file security by resetting the OPENCTL parameter and by entering passwords for the CCACAT file entries in the password table (CCASTAT). The default value of the OPENCTL

parameter is X'80'. CCASTAT maintenance is described in the *Model 204 System Manager's Guide*.

The default setting for PRIVDEF is X'BFFF', which includes all privileges. The recommended minimum settings for the PRIVDEF file parameter are X'8441' for the system administrator and X'0441' for the SQL user.

Consistency and recovery

CCACAT, like other Model 204 files, can participate in recovery and transaction backout processing. If CCACAT is used in a run, it can be recovered by a RESTART command.

The FRCVOPT (file recovery options) parameter for CCACAT can ensure that:

- File cannot be updated if ROLL FORWARD logging is not active.
- File cannot be updated if checkpoint logging is not active.

The parameter settings are distributed with the CREATE procedure. For more file creation information, refer to the *Model 204 File Manager's Guide*. For more about setting FRCVOPT see *Model 204 Parameter and Command Reference*.

Ongoing CCACAT maintenance

This section describes CCACAT file maintenance issues that are most likely to be post installation, ongoing considerations.

Backup and restore

Back up the CCACAT file frequently, certainly before making any major changes to the catalog. Use the same backup, restore, and reorganization procedures that you use for any Model 204 file.

File organization and sizing

CCACAT is distributed as an entry order file (FILEORG = X'00'). With this setting the CCACAT file might eventually become full even though there is unused space due to deleted catalog entries. In this case, expand CCACAT as you do any Model 204 file. Do *not* change the FILEORG to Reuse Record Number, however, because the order of entries in the catalog is important for the regeneration of DDL from an existing catalog.

The installation procedure also provides default settings for file parameters such as BSIZE, CSIZE, DSIZE, BRECPPG and BRESERVE. The file manager tunes these settings to conform with application requirements. The optimum settings for file sizing parameters vary depending on factors such as the number of tables and columns within tables, length of table names, complexity of view definitions, the number of privilege records, and the number of users (affects number of GRANTS).

In general, it is important to use the normal mechanisms provided for maintaining CCACAT. Use CCACATREPT to view catalog contents, CVI to update, and standard commands such as SQL DROP to delete catalog definitions.

Large DDL updates

If a CVI user submits a CREATE SCHEMA statement that is several pages long, the system manager might have to increase the amount of space allocated to CCACAT. It might also be necessary to increase the Model 204 SQLIQBSZ, SQLBUFSZ and LHEAP parameters so that there is a large enough buffer available to process the update.

You might be able to avoid increasing these parameters by breaking up the DDL into smaller statements (see Chapter 4 for further details). The need to increase the amount of space allocated to CCACAT still remains.

CCACAT implementation for BLOB and CLOB data

Any new SQL Catalog that you create by following the SQL Installation instructions in your Rocket Model 204 Installation Guide will be completely compatible with the Binary Large Object (BLOB) and (CLOB) data types.

If you have an older, pre-v7.4 SQL Catalog, you can perform the procedure below to use the catalog without recreating it. This saves the time required to rerun all DDL to recreate the current tables.

To use your SQL catalog from a Model 204 release before Version 7 Release 4:

1. Back up the SQL Catalog.
2. OPEN CATPROC.
3. INCLUDE ODBCTABLES.INSTALL

This procedure updates the CCACAT file and installs the necessary SQL data types, BLOB/CLOB, for use in your DDL processing.

4. To test this installation:
 - a) CREATE a table that includes BLOB or CLOB data types.
 - b) Run the following SQL SELECT statement:

```
SELECT type_name, data_type FROM CATALOG.ODBC_TYPES
```

This returns the following results showing the new BLOB/CLOB data types.

```
TYPE_NAME DATA_TYPE
CHAR 1
DEC 3
DOUBLE PRECISION 8
FLOAT 6
INT 4
```

Maintaining the SQL catalog

NUMERIC 2
REAL 7
SMALLINT 5
BLOB 30
CLOB 40

3

Mapping Model 204 Data to SQL

In this chapter

- Overview
- Representing Model 204 data in SQL
- Mapping multiply occurring fields to nested tables
- Matching Model 204 and SQL data formats

Overview

Successful SQL processing of the data in Model 204 files requires an accurate SQL catalog definition of the files. Essential to the catalog definition is knowing how to take advantage of Model 204 database features in SQL. This chapter discusses how to work with Model 204 files in SQL: from preliminary file preparation to SQL accommodation of Model 204 file features, data formats, and data handling.

Representing Model 204 data in SQL

This section describes what you need to do to your Model 204 files to prepare for SQL processing and how certain Model 204 file features are handled in SQL processing.

Changing existing Model 204 files

Model 204 SQL requires little, if any, manipulation of the Model 204 file data before you catalog the data. This section describes the Model 204

SQL processing requirements that may necessitate changes to your Model 204 files.

You might have to change file parameter settings, add or redefine fields or field attributes, or update field values as follows:

- Files used in SQL processing must be transaction backout (TBO) files.
A file manager can change non-TBO files to TBO files by issuing the Model 204 RESET command. In addition, logging for TBO files makes demands on CCATEMP file space. For more information about transaction backout files, see the *Model 204 File Manager's Guide*.
- SQL multicolumn unique constraints require addition of a new field.
To enforce an SQL multicolumn unique constraint, you must first define and populate a special Model 204 UNIQUE field, which provides the constraint index. See "Specifying a multicolumn UNIQUE key" on page 58.
- The following operations or SQL column definitions might require you to add or redefine field attributes:
 - INVISIBLE field values cannot be updated by SQL DML. To use SQL DML to update columns mapped to Model 204 INVISIBLE fields, you must redefine the fields to VISIBLE.
 - Model 204 fields mapped to columns designated as primary keys must have the UNIQUE and ORDERED field attributes.
 - Multiply occurring groups of fields mapped to nested tables must have the same number of occurrences.
 - Model 204 field attributes must map to compatible SQL column data types. Your choice of SQL data type determines the type of Model 204 storage index that is used to retrieve data. You might need to change a Model 204 field definition to better fit the data type you select for the corresponding SQL column.
- The following operations or SQL column definitions might require you to check or update Model 204 data values:
 - SQL operations against columns mapped to STRING fields containing hexadecimal zeros are not reliable. Avoid storing hexadecimal zeros in STRING fields.
 - Empty character strings mapped to SQL character columns are findable in SQL with the WHERE clause:

```
WHERE columnname= "
```


In SQL numeric operations, an empty string is interpreted as zero.
 - SQL operations against CHARACTER columns mapped to fields containing characters other than EBCDIC X '40' through X 'FE' are not reliable. Model 204 SQL processing supports only printable characters that sort higher than the blank character, that is, those equivalent to EBCDIC X '40' through X 'FE'.

Characters outside this defined range cannot be entered using SQL. If other than these values exist in a Model 204 file, SQL operations involving character comparisons, sorting, or pattern matching might have unexpected results.

- For joint User Language and SQL access to a file: store data fields with no nonblank characters as *one blank*; do not define the Model 204 PAD character as a blank; and note that SQL removes trailing blanks for fixed-length CHARACTER data while User Language does not.

SQL pattern search guidelines

For single character SQL pattern searches to return correct results, the ORDERED CHARACTER attribute is recommended.

Do not end an SQL search pattern with an underscore, the single character substitution symbol. Instead, use %, for multiple character substitution.

Using Model 204 file data features

This section describes what you have to do to use the following Model 204 file features in SQL processing.

File security

Model 204 files accessed for SQL processing are also available for User Language and Host Language Interface (HLI) use. These files are subject to both Model 204 security and SQL security.

The usual Model 204 security features are in effect for User Language and HLI access. SQL access to these files is protected by Model 204 login security and SQL security from GRANT, REVOKE, and CREATE VIEW statements.

Model 204 field and record security are not directly applicable in the SQL environment.

File groups

Model 204 file groups cannot be represented directly in SQL. However, you can simulate file groups with a view, and you can retrieve (but not update) data through the view. See “Simulating file groups” on page 72.

If there is a group file with *filename1* and a file with *filename1*, Model 204 will open the group file and try to process the SQL request against the group file.

Model 204 precedence algorithm tries to open a Group-file *filename1* first. If Group-file *filename1* does not exist, Model 204 tries to open File *filename1*. Take care when naming a group-file and a file, using the same name; the file cannot be accessed directly through SQL because of the precedence algorithm. See the OPEN command in the *Model 204 Command Reference Manual* for a more detailed explanation.

Sorted or hash key files

If a Model 204 file is a sorted or hash key file, specify the sort or hash key as a column in the table for the file. If the sort or hash key is required in the file (Model 204 FILEORG parameter X'02' option is set), define the column NOT NULL. For more information about mapping SQL columns to sort or hash keys, see “Defining columns” on page 55.

Files with multiply occurring fields

For full use of Model 204 multiply occurring fields, you must translate them into SQL nested tables. The nested table feature is a Model 204 SQL extension described in “Mapping multiply occurring fields to nested tables” on page 27.

Files with INVISIBLE fields

In Model 204, INVISIBLE fields are stored in the index portion of the file and not in the data portion (Table B). They are typically used to assist in Table B data retrieval operations but are not themselves printed or sorted. They generally require special treatment for Model 204 file and field update operations.

In Model 204 SQL, columns mapped to INVISIBLE fields are available to qualify searches but are not themselves retrievable. They also require special treatment for the files they map to and have restrictions on their use in SQL DML specifications.

In general, Model 204 files that contain INVISIBLE fields should be maintained by User Language or Host Language Interface applications and not by SQL applications. This recommendation applies regardless of whether the fields are mapped to SQL columns. It does not apply to files that only have INVISIBLE fields that are mapped to multi-column unique constraints.

For example, you can remove an SQL row containing a column mapped to an INVISIBLE field with SQL DELETE, but there is no way with SQL DML to remove the INVISIBLE field from the Model 204 file index. Similarly, SQL UPDATE can modify a visible SQL column but cannot affect any INVISIBLE field values.

You can use an INVISIBLE field as a concatenation of the individual fields that map to the SQL columns that comprise a multicolumn unique key. However, the individual fields that map to these SQL columns might not be INVISIBLE.

Files with multiple record types

You must map all the data in each Model 204 file to a single non-nested SQL table (plus optional nested tables). Consequently, SQL column attributes you assign must apply to every record in a file. This might present a problem for files that have multiple record types. A record type is a set of records having the same collection of field names or formats and connected by the same value of a record type field.

For example, a single file may have a set of driver records (with field RECTYPE equal to DRIVERS) and a set of policyholder records (with field RECTYPE equal to POLICYHOLDER). Many of the fields on the driver records are not common to the policyholder records, and vice versa.

To avoid retrieving unwanted driver information when you need only policyholder information, your SQL queries against the file must list each of the policyholder column names. Your queries grow lengthy and you have to keep track of the fields associated with each record type.

In addition, files with multiple record types limit the use of the SQL NOT NULL attribute. SQL column attributes must apply to all the records in the file, regardless of record type. Only fields common to both driver and policyholder records (with no empty-string values) can validly be mapped to a column defined with NOT NULL.

The most efficient way to map multiple record types within the same file is to use SQL views. After defining all the fields in each record type to the SQL base table you are creating, you define a view for each record type. The view definition contains a listing of the columns associated with the record type.

You can then execute simpler queries against the view, taking advantage of the preselection of required columns in the view definition. For an example of views defined for mixed record type files, see “Mapping files with mixed record types” on page 73.

File data indexes

Model 204 data retrieval indexes (for example, NUMERIC RANGE and ORDERED) might be ignored for SQL queries against the file, if you are not careful about your SQL catalog mapping of SQL and Model 204 data types.

Model 204 data values inconsistent with the SQL attributes for a column might not be usable by the SQL application or might be truncated upon conversion to the specified SQL format. Compatibility between SQL and Model 204 data formats is discussed in “Matching Model 204 and SQL data formats” on page 31.

Using PRIMARY KEY table columns

Successful SQL processing of the data in Model 204 files requires an accurate SQL catalog definition of the files. For example, mapping a Model 204 field to a column designated as a PRIMARY KEY requires the use of UNIQUE and ORDERED field attributes.

Some third-party, SQL compliant packages require a UNIQUE key in any table or view that is to be updated.

To ensure that such products work with Model 204 data, the following restrictions apply:

- The base table in question, either the table to be queried or the table underlying the view to be queried, must be defined with a PRIMARY KEY. This PRIMARY KEY can be either a system-generated key (that is, the Model 204 record number) or a unique ORDERED INDEX. It must consist of a single column.
- When you define a view that you want to update, it must be defined on one of the following:
 - Table with a PRIMARY KEY, with the PRIMARY KEY visible in the view definition
 - View conforming to a table with a PRIMARY KEY visible in the definition of the new view as well

Because a Model 204 nested table cannot have a PRIMARY KEY, any third-party product that requires a PRIMARY KEY for updating cannot update nested tables.

Model 204 and SQL data extraction mismatches

When you extract values in a Model 204 field that do not conform to a defined SQL data type, you can get unexpected results. For example, if a Model 204 stored value 0.05 (which is not an integer) is mapped to an SQL integer data type and you try to fetch that row with:

```
WHERE colname=0
```

There are no returned values. However, if you rewrite the request to:

```
WHERE colname>0 AND colname<1
```

SQL fetches the row and shows the column value as 0.

Model 204 extracts from the file all records qualifying for the selection criteria. From that found set, the SQL engine flags as Dirty Data only those rows where the data cannot be converted to the SQL data type.

You might see the following messages in the audit trail, which may affect the outcome of your query:

```
M204.0554: DIVIDE BY ZERO
```

```
M204.0563: ARITHMETIC OVERFLOW
```

Message 0554 indicates that an arithmetic expression attempted to divide by zero.

The SQL error message will be like the following:

```
SQL Error -802 DIVIDE BY ZERO exception has occurred  
during INTEGER DIVISION processing.
```


or

SQL Error -802 DECIMAL OVERFLOW exception has occurred during DECIMAL MULTIPLICATION processing.

Mapping multiply occurring fields to nested tables

A nested table is a Model 204 SQL extension that makes Model 204 multiply occurring fields available to SQL access. This section introduces the basic design of the feature and includes explanation of a simple example.

For information about using Model 204 SQL DDL to define nested tables to the SQL catalog, see “Rules for nested table columns” on page 63. For information about creating nested tables with the Table Specification facility, see “Defining nested tables” on page 108. For information about using SQL DML to access nested tables, see “Retrieving a particular occurrence of a multiply occurring group” on page 170.

Understanding nested tables

In the relational model, SQL tables must be normalized: at each row-column position in a table, there can be only one value (or a null). Repeating, or multiple, values are not allowed. This is not a requirement in Model 204 databases. In a single Model 204 record an individual field can have a set of, or multiple, values. Such a field is called a multiply occurring field. A group of multiply occurring fields is also allowed.

The Model 204 SQL nested table extension allows you to take advantage of Model 204 multiply occurring fields. This extension permits the mapping of a file with multiply occurring fields to a main (*parent*) SQL table plus one or more subsidiary (*nested*) tables related to the parent. The nested tables contain only columns that map to multiply occurring fields and a single key column that joins the nested table to its parent.

The values of the nested table joining key column (*nesting key*, or *foreign key*) are the same as those of the *primary key* in the parent table. These values must be unique. If the nesting key is a composite of two or more columns, you can have the Model 204 SQL Server generate and maintain a SYSTEM primary key.

A nested table can have only one parent table and must belong to the same SQL schema as the parent. A parent table, which must not be nested, can have multiple nested tables.

The Model 204 SQL nested table extension is a DDL extension only. Once you have defined repeating fields as nested tables in the SQL catalog, you can issue standard SQL DML against these tables.

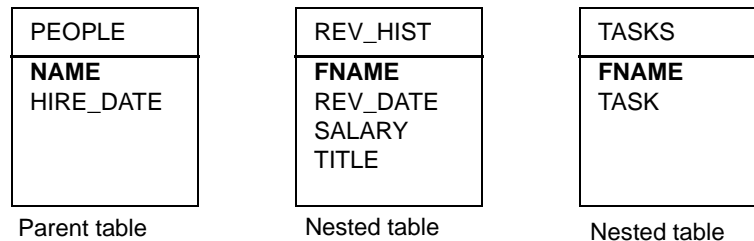
Translating multiply occurring fields

A simple example follows of a Model 204 SQL translation of Model 204 multiply occurring fields. A Model 204 file has the following fields:

Field	Frequency of occurrence
NAME	Once per record
HIRE_DATE	Once per record
REV_DATE	Multiply occurring group member
SALARY	Multiply occurring group member
TITLE	Multiply occurring group member
TASK	Multiple times per record

REV_DATE, SALARY, and TITLE are repeating groups that occur once each salary review.

You can translate this Model 204 situation into one SQL parent table (PEOPLE) with two columns (NAME and HIRE_DATE) and two nested tables (REV_HIST and TASKS):



The parent table is linked to the nested tables by the common values of the primary key NAME in the parent and the foreign key FNAME in the nested tables. The values of NAME must be unique (NAME must have the Model 204 UNIQUE attribute). Each value of NAME or FNAME is the unique identifier the SQL Server uses to locate the Model 204 record that contains the multiply occurring values.

SQL updates to the primary key are propagated to the foreign key. You may not directly update the foreign key.

Notice that nested tables may include only a foreign key and the columns that map to Model 204 multiply occurring fields or multiply occurring groups.

This example is expanded on the next page.

Simulating normalization of Model 204 record data

The file in the preceding example contains the following two records:

Record 1

NAME	HIRE_DATE	REV_DATE	SALARY	TITLE	TASK
n1	n1HD	n1RD1	n1S1	n1TI1	n1T1
		n1RD2	n1S2	n1TI2	n1T2
					n1T3

Record 2

NAME	HIRE_DATE	REV_DATE	SALARY	TITLE	TASK
n2	n2HD	n2RD1	n2S1	n2TI1	n2T1
		n2RD2	n2S2	n2TI2	n2T2
		n2RD3	n2S3	n2TI3	

Using the Model 204 SQL DDL for nested tables discussed in “Creating nested tables” on page 62, you map this file to the SQL parent table PEOPLE and the nested tables REV_HIST and TASKS. Logically the data appears to the SQL Server as follows:

Table	Columns	
PEOPLE	NAME	HIRE_DATE
	n1	n1HD
	n2	n2HD

Table	Columns			
REV_HIST	FNAME	REV_DATE	SALARY	TITLE
	n1	n1RD1	n1S1	n1TI1
	n1	n1RD2	n1S2	n1TI2
	n2	n2RD1	n2S1	n2TI1
	n2	n2RD2	n2S2	n2TI2
	n2	n2RD3	n2S3	n2TI3

Table	Columns	
TASKS	FNAME	TASK
	n1	n1T1
	n1	n1T2
	n1	n1T3
	n2	n2T1
	n2	n2T2

The nested table rows are ordered and retrieved by matching occurrence: the first row is mapped to the first occurrences of the repeating field values, the second row is mapped to the second occurrences of the repeating field values, and so on. The rank of occurrence (first, second, and so on) is determined by physical storage order in the file.

When you issue a query against a nested table, the query must specify either the primary key of the parent table or the foreign key of the nested table. The Model 204 SQL Server uses the unique value of the primary or foreign key in the nested table row to locate the physical record with the repeating field values.

Handling foreign keys

If a foreign key is defined twice, first with a REFERENCES clause and then with a FOREIGN KEY clause, the statement is accepted only if the two clauses are identical and reference the same column.

SQL error message -4703 is generated if a FOREIGN key is defined twice for a nested table and the two keys do not reference the same column. For example:

Acceptable:

```
CREATE TABLE NESTEDINVENTOR2 NESTED USING PART_NO
(PART_NO
  DECIMAL(8) NOT NULL
  REFERENCES INVENTORY,
ON_HAND
  SYSNAME 'ON HAND'
  FLOAT(4) NOT NULL,
LOCATION
  CHAR(255) NOT NULL,
FOREIGN KEY (PART_NO) REFERENCES INVENTORY)
```

Returns SQL error -4703:

```
CREATE TABLE NESTEDINVENTOR2 NESTED USING PART_NO
(PART_NO
  DECIMAL(8) NOT NULL
```

```

REFERENCES INVENTORY,
ON_HAND
SYSNAME 'ON HAND'
FLOAT(4) NOT NULL,
LOCATION
CHAR(255) NOT NULL,
FOREIGN KEY (ON_HAND) REFERENCES INVENTORY)

```

Handling primary keys

SQL error message -6315 is now displayed if a column in a table is defined with PRIMARY KEY SYSTEM and the SYSNAME clause. The following example returns SQL error message -6315:

```

CREATE TABLE XYZ
(COL1          INT,
 COL2          CHAR (25) ,
 F_KEY  INT
                PRIMARY KEY SYSTEM
                SYSNAME 'INDEX')

```

Matching Model 204 and SQL data formats

The DDL you create to map SQL tables to Model 204 files specifies the data types of SQL columns. Model 204 field attributes and SQL column attributes place different restrictions on the type of data values that can be stored in a field or column. This section discusses aspects of Model 204 data mapping you need to consider when you are preparing SQL DDL to map the Model 204 file data.

Compatibility of Model 204 and SQL data formats

Model 204 data typing is more informal than SQL data typing. Therefore, Model 204 files can include data that meets the Model 204 data typing definitions, but which might cause unexpected results when a Model 204 SQL program is run that uses Model 204 data.

In general, Model 204 SQL returns data to an SQL application in the SQL format requested, and stores data in the Model 204 format after conversion from the defined SQL format. It is your responsibility to ensure that the SQL format you define for a column is compatible with the format of the data stored in the Model 204 file.

Data format compatibility between Model 204 and SQL affects the accuracy and efficiency of data retrievals and data conversions. The broad levels of compatibility between Model 204 and SQL data formats are:

Data retrieval performance	This first level of compatibility is more general and requires that you pair SQL numeric column data types with numeric Model 204 data and nonnumeric types with nonnumeric data. Failure to do so means that you cannot use Model 204 indexes for that field to retrieve data, or certain data is not retrievable, or both.
Data conversion	This second level of compatibility extends the first and requires Model 204 format and SQL data type combinations that result in data conversions without loss of accuracy or precision.

Optimizing Model 204 data retrieval

Data retrieval is more efficient in Model 204 when you can use a variety of numeric and nonnumeric indexes. Model 204 data indexes are defined with Model 204 field attributes such as KEY, NUMERIC RANGE, and ORDERED. SQL application programmers are responsible for understanding the effects of these attributes and for tailoring their applications accordingly. Model 204 field attributes are described in the *Model 204 File Manager's Guide*.

Optimizing retrieval for SQL selections

When using a SELECT DISTINCT statement on an SQL column, define the underlying Model 204 field as ORDERED or FRV to eliminate NULL column values from the result.

The Model 204 SQL engine is specifically designed to handle Model 204 field types. Rocket Software recommends that you define all fields that are used in SQL DISTINCT or WHERE processing with the ORDERED attribute. Performance improvements are noticeable.

Figure 3-1 compares the paths taken when fields are *not* ORDERED or *not* FRV with the path taken when fields are defined ORDERED or FRV.

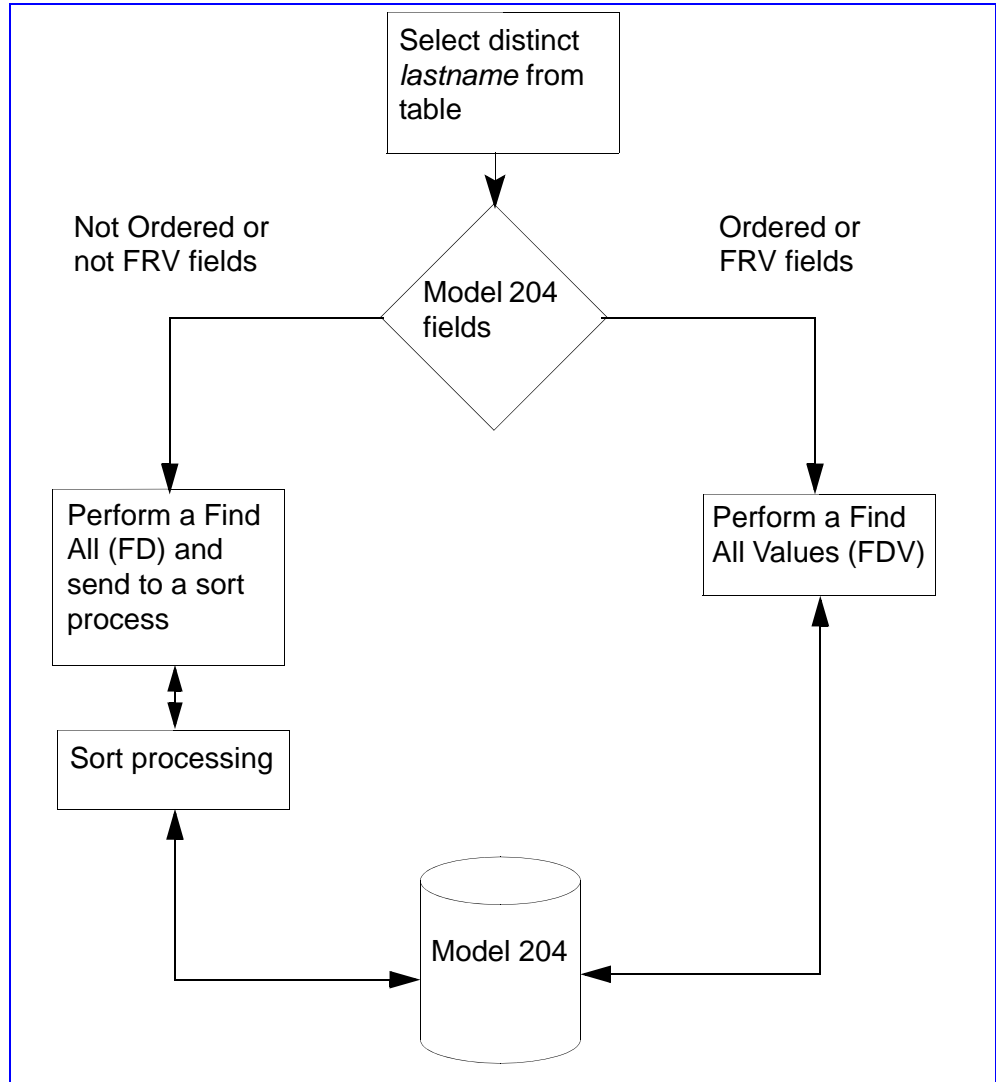


Figure 3-1. SQL SELECT DISTINCT processing paths

You can preserve these Model 204 retrieval efficiencies in SQL by making sure that you assign compatible SQL data types to the Model 204 fields you are using. Usually you define an SQL numeric data type for numeric Model 204 data, and you define the SQL nonnumeric data type for nonnumeric data, although not all cases are this straightforward.

Table 3-1 and Table 3-2 list compatible mappings. Table 3-1 shows permissible mappings of Model 204 data format attributes and the Model 204 indexes that are preserved or lost by such mappings. Table 3-2 on page 35 shows mappings

that are not permitted, because they present serious data conversion problems.

Table 3-1. Model 204 indexes and data format mappings

Model 204 field attribute	SQL data type	Indexes preserved	Indexes not usable
STRING	CHARACTER	Ordered character Key** Hash key	Numeric Range Ordered numeric
STRING	numeric*	Numeric Range Ordered numeric	Ordered character Key Hash key
BINARY (except with OCCURS NON-CODED)	CHARACTER	Ordered character Key**	Numeric Range Ordered numeric
BINARY	numeric*	Numeric Range Ordered numeric	Ordered character Key
FLOAT	numeric*	Ordered numeric Key**	none
CLOB	CLOB	N/A	N/A
BLOB	BLOB	N/A	N/A

* Numeric types supported are:

INTEGER	
SMALLINT	
DECIMAL (precision, scale)	with decimal precision and scale
NUMERIC (precision, scale)	with decimal precision and scale
FLOAT (precision)	with binary precision
REAL	
DOUBLE PRECISION	

** Key Index is used only for direct searches in SQL as in:

WHERE fieldname=value

Avoiding Table B searches

Not using Model 204 data indexes results in data retrievals that are much less efficient and may require searches of the entire Table B of the file. For example, you assign an SQL numeric column attribute, say INTEGER, to a Model 204 STRING field. If the field is also defined with the Model 204 KEY attribute, which indexes the field's values, your pairing of SQL and Model 204 attributes prevents use of the KEY index. Retrievals involving this field search the file's Table B instead of using the KEY index.

The KEY index is used only for equality data retrievals. Range retrievals against a field having only a KEY index results in a Table B search of the entire data file.

A more ambiguous example involves the ANNIV_DATE column, mapped to a Model 204 field that has the STRING and ORDERED NUMERIC field attributes. A sample field value is 0917. If you map the column to SQL DECIMAL(4,0), you can use the Model 204 numeric ordered index in SQL queries (as Table 3-1 on page 34 shows), but the essential leading zeros in the data are not preserved. If you use SQL CHAR(4) to preserve the leading zeros, you lose the benefit of the Model 204 numeric ordered index.

If the data type you assign to an SQL column would prevent the use of a Model 204 index, the SQL Server issues a warning message when you issue *DML* against that data. Successful DDL statement execution does *not* imply a compatible mapping of Model 204 and SQL data types.

If a Model 204 field is defined with more than one index, the data type you assign to your SQL column might prevent the use of one index but preserve another.

Optimizing Model 204 data conversion

The previous section defined the combinations of Model 204 and SQL data formats that allow use of the Model 204 indexes. Compatible combinations of Model 204 and SQL data formats are also necessary to maximize the accuracy of the conversions of the data from Model 204 format to SQL format.

Data is converted when you use SQL to insert values into the Model 204 fields and when you use SQL to retrieve values from Model 204 fields. The SQL Server ensures that data you insert is never truncated to fit Model 204 field format requirements; you cannot execute requests with columns whose SQL data types the Model 204 fields cannot fully accommodate. These incompatible combinations of column-to-field mappings are listed in Table 3-2.

Table 3-2. Model 204/SQL data format incompatibilities

Model 204 attribute	Incompatible SQL data type(s)
STRING (non-preallocated)	None

Table 3-2. Model 204/SQL data format incompatibilities (Continued)

Model 204 attribute	Incompatible SQL data type(s)	
STRING (preallocated with LENGTH n)	CHARACTER(L)	(If $L > n$, the preallocated length)
	DEC(p, s)	(If $s=0$ and $p > n$, or
	or	$s=p$ and $p > n-2$, or
	NUMERIC(p, s)	$0 < s < p$ and $p > n-1$)
	INTEGER	(If $n < 11$)
	SMALLINT	(If $n < 11$)
	FLOAT	
	REAL	
	DOUBLE PRECISION	
BINARY (preallocated with NON-CODED)	CHARACTER	
BINARY (non- preallocated)	DEC(p, s)	(If $s > 0$, or $p > 9$ and $s=0$)
	NUMERIC(p, s)	(If $s > 0$, or $p > 9$ and $s=0$)
	FLOAT	
	REAL	
	DOUBLE PRECISION	
BLOB	all but CLOB	
CLOB	all but BLOB	
FLOAT 4	CHARACTER	
	DEC(p, s)	(If $p > 6$)
	NUMERIC(p, s)	(If $p > 6$)
	INTEGER	
	SMALLINT	
	FLOAT(p)	(If $p > 21$)
	DOUBLE PRECISION	
FLOAT 8 FLOAT 16	CHARACTER	

Note: The following mappings are not prohibited by the rules shown in Table 3-2, but they result in a Model 204 error (soft restart) if you attempt to insert *negative-valued* data with *p* significant digits into these field and column combinations.

For example, you *cannot* insert the value -0.1234 (4 significant digits) into a DEC(4,0) column (s=0, p=4) mapped to a preallocated STRING LEN 4 field (n=4). This is the first case shown below: s=0 and p=n.

Model 204 attribute	SQL data type	
STRING (preallocated with LENGTH n)	DEC(p, s)	(If s=0 and p= n, or s=p and p= n-2, or
	NUMERIC(p, s)	0<s<p and p= n-1)

Data mapping incompatibilities are validated for an SQL table at runtime when a DML statement involves a reference to the catalog. Such incompatibilities result in an SQL statement validation error, and the statement is not processed.

Even with compatible mappings of data formats, the SQL Server does not guarantee that the actual Model 204 data you retrieve with SQL is convertible to the defined SQL format without some modification or truncation. When necessary, the SQL Server observes the following rules for data conversion:

- Whenever possible, Model 204 data is converted (with or without modification) to your SQL specification. Data that the Model 204 SQL Server cannot convert to the format you specify is called *dirty data*. See the following “How Model 204 SQL processes dirty data”.
- Model 204 fields with character strings longer than the defined length for SQL CHARACTER columns are accepted, but the values are truncated. Trailing blanks are removed and leading blanks are preserved if such fields are updated by SQL.
- Numeric data not matching the definition of the SQL data type is truncated, rounded, or converted to match the SQL data type.

How Model 204 SQL processes dirty data

Model 204 data that cannot be converted to fit the defined SQL data type is dirty data. What data is dirty data is influenced by the SQL data type:

- SQL CHARACTER specifications encounter no dirty data. All string and non-preallocated binary data is convertible to this format. Empty strings, for example, are converted to blanks. Empty strings in preallocated fields are converted to blanks.
- For SQL numeric data types, any nonnumeric data is not convertible (dirty data). Empty fields are interpreted as (numeric) zero.

For both SQL character and numeric data types, including nulls, a missing field is interpreted as an SQL null and is not dirty data.

Handling dirty data at runtime

At runtime, each time the Model 204 SQL Server gets data from a Model 204 record, the data type characteristics of each field being returned are validated: if invalid (dirty) data is found in any field referenced in the query, the SQL Server takes one of the following actions:

- Processing of the SQL statement stops, and a negative SQL code is returned.
- The record is bypassed, and an SQL warning message is issued. If many records are bypassed, you might still receive only one warning message. Also, whether the record is included in a COUNT of selected data is unpredictable. Processing of the SQL statement continues.

You might receive a dirty data warning even when your query results are correct, that is, not affected by dirty data. Such a warning indicates that the SQL Engine encountered dirty data while deriving the resultant set of records.

- You will not receive a dirty data warning if the dirty data was detected in the last record processed prior to EOF.

The system manager determines which of these actions the SQL Server takes by the setting of the Model 204 SQLCNVER parameter, described in the *Model 204 SQL Connectivity Guide* and for greater detail, *Model 204 Parameter and Command Reference*. PC clients can override SQLCNVER by selecting alternative action for the Dirty Data Treatment option.

In addition, whenever a conversion error due to dirty data occurs in the SQL Engine, Model 204 error message 1296 is logged to the audit trail. This message identifies the file, field, and record number of the field that experienced the conversion error.

Handling NOT NULL, UNIQUE, and multiply occurring data

If an SQL column is defined as NOT NULL, the Model 204 SQL Server does not allow SQL updates that result in a null value in the corresponding Model 204 field.

Note: User Language operations can introduce a null into a Model 204 field that is mapped to a NON NULL column. These operations circumvent Model 204 SQL NON NULL checking, which is enforced only for SQL operations. An SQL SELECT against such a column would return the null value.

Except for nested table columns, the SQL Server does not inform you when null values are found for a NOT NULL column. If a row of null values is found for a nested table, the SQL Server issues a warning message, bypasses the record, and continues processing. If a row with some, but not all, null values is found for a nested table, the SQL Server ends the processing of the statement with a negative SQL code.

No retrievals or updates are allowed against an SQL UNIQUE column not mapped to a Model 204 ORDERED UNIQUE field. The uniqueness of the data in such mappings is guaranteed by Model 204 UNIQUE attribute checking. If you redefine the Model 204 field to be not-UNIQUE, the SQL Server prevents retrievals or updates against the corresponding UNIQUE column.

If the Model 204 multiply occurring fields mapped to the columns of a nested table do not have the same number of occurrences, attempts to query, fetch, or update rows with mismatched occurrences have unexpected retrieval results and produce no error return code. Inserts of data into such fields are permitted; they do not result in detection of the mismatches.

Handling mixed numeric and nonnumeric data

SQL data typing is not well suited for handling mixed numeric and nonnumeric data, for example a Model 204 CODED BIN field. If this were the Model 204 field format, the best choice for SQL column type would be CHAR. Although specifying CHAR allows only character operations with the data, at least the data is convertible.

Rocket Software recommends that you examine the actual data stored in a field before assigning an SQL data type. Match the physical data characteristics and not simply the designated Model 204 data attribute.

Observing data precision limits

This section discusses discrepancies between the precision of the data stored in Model 204 fields and the precision available to the data types of the SQL columns mapped to those fields. Data precision considerations are discussed for decimal integer data and then for floating point format.

Decimal integer

The Model 204 SQL Server treats data mapped to SQL DECIMAL and NUMERIC the same. The SQL Server supports user-defined scale and precision for DECIMAL and NUMERIC (where $0 \leq \text{scale} \leq \text{precision}$) and maintains both formats with a maximum precision of 15 decimal digits.

The SQL Server treats data mapped to SQL INTEGER and SMALLINT the same. The SQL Server maintains *both* SQL INTEGER and SMALLINT column formats with a maximum precision of four bytes (31 bits plus one bit for the sign: the range from decimal -2147483648 to 2147483647). The 4-byte integer precision limit is in effect for all operations involving this data, and any operations that exceed this limit result in an error message.

Note: Remember that the Model 204 BINARY field attribute has at most 30 bits of precision available. Mapping Model 204 fields to SQL DECIMAL, NUMERIC, INTEGER, and SMALLINT columns is subject to the restrictions listed in Table 3-2 on page 35.

Floating point

Model 204 FLOAT (floating point) fields can be one of the following lengths, reflecting the possible precision:

This precision...	Is equivalent to...
4-byte floating point (FLOAT LEN 4)	6 decimal digits of precision
8-byte floating point (FLOAT LEN 8)	15 digits of precision
16-byte floating point (FLOAT LEN 16)	31 digits of precision

Model 204 SQL also defines SQL column precision available to the SQL floating point data types. Data you attempt to store or extract that is longer than the precision limit is truncated either before storage or before extraction:

This precision...	Is limited to...
REAL and FLOAT (binary precision ≤ 21)	6 decimal digits of precision
DOUBLE PRECISION and FLOAT (binary precision > 21)	15 decimal digits of precision

As in Model 204 User Language, 15 significant digits is the Model 204 SQL maximum. You can still retrieve data stored in a FLOAT LEN 16 field, for example, but the precision of the retrieved value will be no more than 15.

SQL processing floating point numbers

SQL processing of floating point values greater than the largest valid value in:

- An INSERT statement results in:

```
SQL error -103. '7.237E75 is an invalid numeric literal.'
```

- A SELECT statement returns an SQL warning with:

```
'Invalid data was skipped by the SQL Engine.'
```

- A SELECT DISTINCT statement retrieves no data for such a value.

Mapping recommendations

The precision limits for Model 204 fields and SQL columns dictate the recommended mappings shown in Table 3-3. These mappings are most efficient in terms of space usage and precision preservation. For example, if you map a FLOAT LEN 8 field to an SQL REAL column, you will lose some of

the precision of your stored data, or you will waste space by storing 6-significant digit data in 15-significant digit fields, or both.

Table 3-3. Mapping floating point fields

Model 204 field attribute	Most compatible SQL data type	Maximum precision (decimal digits)
FLOAT LEN 4	REAL FLOAT ≤ 21	6
FLOAT LEN 8	DOUBLE PRECISION FLOAT > 21	15

Remember, the actual precision of the data returned to an SQL application is never greater than the precision of the stored data, which is always the precision of the defined format of the Model 204 fields. To return the data according to your SQL data type specification, the SQL Server converts (rounds, truncates, expands) the field values.

You can't specify a precision on the column that's greater than that of the Model 204 field, or at least it won't work. If you have a Model 204 field defined as FLOAT LEN 4, and you define the precision of the mapped SQL column as FLOAT(53), then you will get an SQL error code (5518), or the row will be skipped, depending upon the Dirty Data options.

If the SQL data type precision you specify does not match the precision of the field data, the actual precision of the data returned to you has the lower precision of the two. If you specify a lower precision than the stored data, the data is truncated to give the lower precision you specify; if you specify a higher precision than the stored data, the data is expanded to meet your specification, but it retains the lower precision of its storage format.

To achieve the precision you specify for an SQL column, make sure the column is mapped to the Model 204 FLOAT LEN that matches your specification. Also, remember that your mappings are subject to the incompatibility restrictions listed in Table 3-2 on page 35.

You can also map a Model 204 FLOAT field to an SQL DECIMAL or NUMERIC column. The precision considerations are the same as above. For example, mapping a DECIMAL 4 to FLOAT LEN 8 loses precision, wastes space, or both.

In general, Model 204 SQL precision and conversion rules match those for Model 204 User Language. These rules are discussed in the *Model 204 User Language Manual*. For more information about Model 204 floating point fields, see the *Model 204 File Manager's Guide*.

Converting SQL data types for display

The preceding discussion of precision limits does not address the conversion of retrieved data to the final display or print format for your application. These conversions are from one SQL data type to another.

Model 204 SQL processing of client result packets converts data in SQL columns to CHARACTER data.

These conversions primarily affect numeric data and might result in unexpected truncation or rounding of data or loss of least significant digits. You need to be aware of the following:

- In Model 204 SQL conversions to CHARACTER:
 - If from FLOAT, results might be in scientific notation, and the number of significant digits displayed is not under user control.
 - Truncation and/or rounding rules might vary with the client platform. For example, where 2 and 5 are the sixth and seventh digits, respectively, of a retrieved value, Model 204 SQL might round up and display 3 for a Connect★ client.
 - Your data might be truncated (with a warning message) if the display format is not long enough.
- In Model 204 SQL conversions from FLOAT:
 - Conversions to any data type risk truncation and/or rounding.
 - Truncation or rounding affects the least significant digits or characters *on the right*. The leading digit and magnitude are always preserved.
 - Truncation or rounding occurs without a warning message or notice.
 - Conversions of 8-byte float to 4-byte float risk loss of least significant digits, because 8-byte float holds more significant digits than 4-byte float.
- Model 204 4-byte (and 8-byte) floating point numbers use IBM mainframe floating point representation and can accommodate exponents as large as 75. IBM PC clients use IEEE floating point representation and are limited to exponents as large as 38 for 4-byte floating point numbers. If you are an IBM PC client and want to retrieve the full Model 204 4-byte floating point range, you must request data type conversion to 8-byte float (DOUBLE or FLOAT 22 or greater) or CHARACTER.

LOB fields in SQL statements

The SQL Server supports the definition, update and retrieval of columns containing CLOB (Character Large Object) and BLOB (Binary Large Object) data.

CLOB/BLOB sizes

The maximum size LOB (Large Object) that may be transferred is dependent upon the amount of storage available to allocate the SQLBUF and the Universal Buffer. The SQLBUF and the Universal Buffer must be large enough to hold the entire LOB. Model 204 v7.4 SQL supports a maximum size CLOB/BLOB of up to 1 GB for UPDATE or INSERT and up to 2 GB for SELECT depending on the amount of server memory available.

The CHAR_MAX_LENGTH in the SQL catalog is set to a non-zero value for CLOB and BLOB fields. This value is for internal use only.

SQL statements supporting CLOB/BLOB data types

The CREATE TABLE, SELECT, INSERT, UPDATE, and DELETE SQL statements have been enhanced to support the CLOB and BLOB data types. See “Using CLOB or BLOB data” on page 55 for details on using large object data in CREATE TABLE.

SELECT

To download a CLOB or BLOB column, specify the column name in the SELECT statement as you would for any other column data type. However, you cannot perform a search on a CLOB/BLOB column. In other words a CLOB or BLOB column name cannot be specified in the WHERE clause.

For example, you can use the following SELECT statement to download the User Language Manual from the MANUALS table defined above:

```
SELECT MANUAL FROM MANUALS
WHERE NAME = 'User Language Manual'
```

INSERT

To insert a CLOB or BLOB column into a row, the INSERT statement specifies the CLOB or BLOB column name as it would any other type of column. The value of the CLOB or BLOB data must be specified as a parameter. INSERT and UPDATE of LOB data through SQL requests can be CPU intensive.

Therefore, Rocket Software recommends that when inserting or updating very large LOBs (over 100MB) or many small/medium size LOBs (0-100MB), Model 204 USER priority should be set to low for users doing such SQL requests, to allow other users in the Online to process normally. If possible, such updating should be done during non-peak hours. Normal single (or small number) INSERT/UPDATE of small LOBs should not noticeably affect other users. We recommend that each customer determine the CPU usage effects of SQL LOB processing applications for each instance of the application before distributing such applications to users.

UPDATE

To update a CLOB/BLOB field in a row, the UPDATE statement specifies the CLOB or BLOB field name as it would any other type of field. The value of the CLOB or BLOB data must be specified as a parameter.

DELETE

To delete a row containing a CLOB or BLOB column, use the DELETE statement with its usual syntax.

Limitations

Following are the known limitations when using CLOB or BLOB fields in SQL statements.

LOB values cannot:

- be key values
- be compared in predicates
- appear in any clause which will involve a data comparison, including but not limited to:
 - GROUP BY
 - HAVING
 - ORDER BY
 - SELECT DISTINCT
 - WHERE
 - ON
 - IN
 - LIKE

Multiple CLOB/BLOB fields cannot be specified in an INSERT or UPDATE statement. To store multiple CLOB/BLOB columns per row, you must execute a separate UPDATE statement for each CLOB/BLOB. This limitation does not apply to SELECT; you can specify multiple CLOB/BLOB columns in the SELECT statement.

ONLINE Parameter Considerations

The SQLBUFSZ parameter defines the length of the maximum incoming SQL message. For more information on SQLBUFSZ, see the *Rocket Model 204 Parameter and Command Reference*.

4

Model 204 SQL Data Definition Language

In this chapter

- Overview
- Model 204 SQL DDL statements
- Creating SQL objects
- Creating schemas
- Creating tables
- Defining columns
- Creating nested tables
- Creating views
- Setting the schema and user context
- Altering SQL objects
- Dropping SQL objects
- Granting privileges for SQL objects
- DDL statement-level security
- SQL DDL processing

Overview

This chapter describes the characteristics of the Model 204 SQL Data Definition Language (DDL). DDL statements are SQL statements that are used to maintain the SQL catalog by creating and altering table, view, and column definitions that describe Model 204 file data. DDL also includes SQL GRANT and REVOKE statements that define the security associated with SQL tables, views, and columns.

Emphasis in this chapter is on the characteristics of Model 204 SQL DDL that differ from the ANSI SQL 1989 standard DDL. Unless otherwise specified, references to “the standard” in this chapter are to the ANSI SQL 1989 standard.

The description of the Model 204 SQL DDL is organized by function (creating, altering, dropping, and granting), including basic syntax for the statements that perform each function.

A diagram of the entire Model 204 SQL DDL statement syntax is found in Appendix A.

Model 204 SQL DDL statements

Table 4-1 on page 46 identifies the DDL statements that can be processed against CCACAT, the SQL catalog file. Each statement is discussed in this chapter.

The statements in Table 4-1 affect *only* the SQL catalog file. Model 204 database files other than CCACAT are never affected by these statements.

The second column of Table 4-1 explains the effect of each of the statements on the SQL catalog records. The catalog has the following record types:

- SCHEMA (S)
- TABLE
 - TABLE (T)
 - TABLE (V)
- PRIVILEGE (P)
- CONSTRAINT (C)

Note: TABLE has two subtypes: TABLE(T) for tables and TABLE(V) for views.

Table 4-1. Effects of DDL statements

DDL statement	Effect on CCACAT
CREATE SCHEMA	Adds a SCHEMA record if none is already present.
CREATE TABLE	Adds a TABLE(T) record and possibly CONSTRAINT records for multicolumn unique indexes, if present. Adds PRIVILEGE records for the table owner.

Table 4-1. Effects of DDL statements (Continued)

DDL statement	Effect on CCACAT
CREATE VIEW	Adds a TABLE(V) record and adds PRIVILEGE records for the view owner.
GRANT	Adds or updates one or more PRIVILEGE records. If no column list, number of records is: no.-of-grantees * no.-of-privileges If a column list, number of records is: no.-of-grantees * (no.-of-privileges- without-lists + no.-of-columns)
SET SCHEMA	None.
SET USER	None.
DROP SCHEMA	Physically deletes SCHEMA record and all TABLE, VIEW, PRIVILEGE, and CONSTRAINT records for objects associated with this schema.
DROP TABLE	Physically deletes TABLE(T) record and all PRIVILEGE and CONSTRAINT records associated with this table. If a parent table, it cannot be dropped before nested tables associated with it are dropped. No effect on TABLE(V) records that reference this table: view definitions involving this table are left intact.
DROP VIEW	Physically deletes the TABLE(V) record and all PRIVILEGE records for the view.
ALTER TABLE	Cannot be used against views, so no effect on TABLE(V) records.
ADD	Adds COLUMN occurrence group to existing TABLE(T) record.
DROP	Deletes COLUMN occurrence group from existing TABLE(T) record. Deletes PRIVILEGE record(s) that reference this column. No effect on CONSTRAINT records because DROP is not allowed for columns that are part of multi-column unique constraints.
MODIFY	Updates column occurrence group on existing TABLE(T) record. No effect on CONSTRAINT records, because MODIFY is not allowed for constrained columns.
REVOKE	Updates or deletes existing PRIVILEGE records.

Model 204 SQL DDL extensions

In addition to the statements in Table 4-1, Model 204 SQL DDL includes the extensions listed in Table 4-2 on page 48.

Table 4-2. DDL extensions

Extension	Statement	Description
NESTED USING clause	CREATE TABLE	Defines the table as nested within a parent table and specifies the joining column. Used for mapping Model 204 files with repeating fields or repeating groups of fields.
REFERENCES clause	CREATE TABLE	Syntax as part of an optional referential constraint definition is the same as defined in the standard. Its extended functionality in Model 204 SQL DDL is that it is <i>required</i> in any nested table definition, it is ignored for nonnested tables, and it implies a CASCADE action.
SYSNAME clause	CREATE TABLE	Identifies the actual name of the Model 204 file or field associated with the SQL table or column being defined. For more information, see “Mapping table names to file names” on page 53 and “Column naming and the SYSNAME extension” on page 58.
SYSTEM clause	CREATE TABLE	Used in column definition with the PRIMARY KEY option to provide a system-generated primary key. For more information, see “Using system-generated keys” on page 67.
SET USER statement		Sets SQL authorization ID without affecting Model 204 login ID, which allows a system manager to create SQL objects for another user without having to login as that user or give that user system manager privileges.

Creating SQL objects

SQL objects (tables, columns, and views) are created with the CREATE statement in the context of a schema. The Model 204 SQL syntax for CREATE SCHEMA is shown on “Creating schemas” on page 50.

This section introduces the descriptions of the Model 204 SQL DDL statements with which you define SQL objects:

```
CREATE SCHEMA
CREATE TABLE
CREATE VIEW
```

Note: The use and function of the DDL statements in this chapter conform to the standard except where otherwise specified.

Authorization ID is equivalent to Model 204 user ID

You establish yourself as a valid SQL user by logging in to Model 204 with the LOGIN command. You are verified as a valid SQL user by Model 204 login security and any external security package that is in effect. The Model 204 user ID you specify in the LOGIN command becomes the SQL user authorization ID used by various DDL statements in a Model 204 SQL session. This Model 204 login ID is also the value returned when an SQL DML query specifies the SQL keyword USER.

For more information about Model 204 login security, see the *Model 204 System Manager's Guide*.

Model 204 SQL table types

Model 204 SQL DDL defines the following types of tables:

- *Base tables* are SQL schema tables that map directly to Model 204 files and fields. Such tables are not defined in terms of any other tables.
- *Views* are schema tables that map directly to SQL base tables or to other views. Base tables are created by CREATE TABLE; views are created by CREATE VIEW.
- *Nested tables* are base tables that have columns that map to Model 204 multiply occurring fields or groups of fields. A nested table is associated with a single base table *parent* by a unique table column key that joins the nested table to the parent table. Nested tables are a Model 204 SQL extension.

Statement ordering is important

You must create objects before they are referenced by other objects. For example, tables referenced by views must be created before the views that reference them.

Rocket Software recommends that you organize statements in a CREATE SCHEMA statement in the following order:

Order	Statement	Description
1.	CREATE table statements	Parent tables, which cannot be nested, referenced in a NESTED clause must be defined before the nested table being created.
2.	CREATE view statements	Views referenced in other views must be created before the other views.

Order	Statement	Description
3.	GRANT statements (for tables and views)	Creating objects before objects that they reference causes a semantic error during compilation.

Naming SQL objects

The Model 204 SQL rules for naming SQL schemas, tables, views, and columns are the same as the standard SQL identifier rules: SQL object names can contain the characters A-Z, 0-9, or underscore, can have as many as 18 characters, and must begin with a letter. You cannot use embedded blanks or SQL reserved words (see Appendix B).

Creating schemas

Like tables and columns, schemas exist as distinct records in the SQL catalog that are created by CREATE SCHEMA. Unlike tables and columns, a schema does not map directly to a Model 204 file.

This section focuses on the rules governing schema creation.

CREATE SCHEMA statement

Syntax

```
CREATE SCHEMA {schemaname
    | AUTHORIZATION authorization-id
    | schemaname AUTHORIZATION authorization-id}
[schema-element ...]
```

Parameters

where:

- *schemaname* conforms to rules for an SQL identifier (see “Naming SQL objects” on page 50). Does not have to be explicitly specified. Examples in “Indicating schema name and owner” on page 51 show how you can indicate the schema name.
- *authorization-id* conforms to the rules for a Model 204 login user ID (no more than 10 characters and no underscore characters, must begin with an alphabetic character and must not contain certain character combinations). The authorization ID cannot be an SQL reserved word (see Appendix B).
- *schema-element* is one of the following:

```
table-definition | view-definition
| privilege-definition
```


- *table-definition* as described in “Creating tables” on page 52.
- *view-definition* as described in “Creating views” on page 69.
- *privilege-definition* as described in “GRANT statement” on page 81 and “REVOKE statement” on page 82.

Syntax rules

The CREATE SCHEMA syntax rules follow:

- Schema name must be unique within the SQL catalog.
- Default authorization ID is the Model 204 login user ID.
- If the schema name is not specified, the schema name defaults to the authorization ID.
- Issuing CREATE SCHEMA for a schema that already exists is an error.
- Only a Model 204 system manager can issue CREATE SCHEMA.
- You can create schemas that have no tables, views, or privileges. Such an empty schema can be defined first and populated later. For example, the following sequence is valid DDL:

```
CREATE SCHEMA S
CREATE SCHEMA P
CREATE SCHEMA SP
```

Indicating schema name and owner

Examples of schema creation with different CREATE SCHEMA formats follow. These examples show different ways to indicate the schema name and owner. You can choose whether to explicitly specify a schema name or to explicitly specify an authorization ID.

In the following example, schema name is TED, schema owner is TED. If you specify no name, the default name is your authorization ID.

```
CREATE SCHEMA AUTHORIZATION TED
CREATE TABLE S (●●●)
CREATE TABLE P (●●●)
CREATE TABLE SP (●●●)
```

In the following example, schema name is MATERIAL_CONTROL, schema owner is MFTNG.

```
CREATE SCHEMA MATERIAL_CONTROL AUTHORIZATION MFTNG
CREATE TABLE PARTS (●●●)
CREATE TABLE SHIPMENTS (●●●)
```

In the following example, schema name is ACCOUNTS, schema owner defaults to the authorization ID of the issuing user.

```
CREATE SCHEMA ACCOUNTS
```

```
CREATE TABLE ACCOUNTS_PAYABLE (●●●)
CREATE TABLE ACCOUNTS_RECEIVE (●●●)
```

The schema name determined by a CREATE SCHEMA statement remains the default schema name, which is assigned to the SQL objects you define, until the CREATE SCHEMA transaction completes. For information about setting the default schema outside the context of a CREATE SCHEMA using the Model 204 SQL extension SET SCHEMA, see “Using SET SCHEMA” on page 76.

Creating tables

This section describes table creation syntax and considerations that apply generally to both nested and non-nested SQL tables. Information that is nested-table specific is presented in “Creating nested tables” on page 62. The definition of columns for a table is described in “Defining columns” on page 55.

A slightly abridged version of the CREATE TABLE statement syntax that the Model 204 SQL Server supports is shown below. See Appendix A for the complete syntax.

CREATE TABLE statement

Syntax

```
CREATE TABLE <tablename>
[ SYSNAME 'filename' | NESTED USING columnname ]
( <column-definition> | <table-constraint-definition>
[, <column-definition> | <table-constraint-definition> ]
●●●)
```

Parameters

where:

- *tablename* conforms to rules for an SQL identifier (see “Naming SQL objects” on page 50). See also “Prefixing the schema name to an SQL object” on page 75 for prefixing schema name.
- *SYSNAME 'filename'* maps a table to a physical file; see “Mapping table names to file names” on page 53 for more detail.
- *NESTED USING columnname* is an extension that identifies a nested table. See “Creating nested tables” on page 62.
- *column-definition* has the following syntax as described in “Defining columns” on page 55.

```
columnname <datatype> [SYSNAME 'fieldname']
[<column-constraint> ●●●]
```

- *table-constraint-definition* has the following syntax:

```
{ UNIQUE | PRIMARY KEY [ SYSTEM ] } ( <column-list> )
[ SYSNAME '<fieldname>' ]
```

```
| FOREIGN KEY ( columnname ) REFERENCES parent-table-  
name  
[ <referential-triggered-action> ]
```

- *UNIQUE* must map to a Model 204 ORDERED UNIQUE field. For discussion, see “Specifying a multicolumn UNIQUE key” on page 58.
- *PRIMARY KEY [SYSTEM]*, where SYSTEM is an extension with which you can have the Model 204 SQL Server generate and manage a unique primary key, as described on “Using system-generated keys” on page 67.
- *SYSNAME 'fieldname'* maps a column to the named Model 204 field; see “Column naming and the SYSNAME extension” on page 58.
- *FOREIGN KEY* and *REFERENCES* clauses for defining referential integrity constraints are supported for nested tables only.

Syntax rules The CREATE TABLE syntax rules follow:

- Table name must be unique within a schema.
- A table must have at least one column defined.
- CHECK table constraint is not supported in Model 204 SQL. If you include a CHECK clause in your SQL DDL, it does not become part of the SQL catalog definition, although you receive no syntax error.

Model 204 SQL does support the WITH CHECK OPTION for views, however.

Mapping table names to file names

The SQL Server maps the table you define to a Model 204 file. Your CREATE TABLE statement determines both the name of the SQL table and the name of the Model 204 file to which the table is mapped. You can implicitly or explicitly identify the Model 204 file to which the table is mapped, or you can have Model 204 generate a unique file name.

Implicitly identifying the file means mapping the table you name in the CREATE TABLE to a file with the same name. Explicitly identifying the file means mapping the table you name to a file that you specify with the keyword SYSNAME. Automatic system generation of the file name is based on your CREATE TABLE specification and is invoked by setting the Model 204 parameter SQLFILE.

As shown in Table 4-3, you indicate which of these file-naming options you want by whether you specify a SYSNAME clause in your CREATE TABLE statement and by your setting of the Model 204 SQLFILE parameter.

Table 4-3. Naming the corresponding Model 204 file

Method	Add SYSNAME <i>filename</i> to CREATE TABLE <i>tablename</i> ?	SQLFILE setting	Resulting file name
Implicit	No	0	<i>tablename</i>
Explicit	Yes	0 or 1	<i>filename</i>
System generated	No	1	Uniquely determined by Model 204

The SYSNAME clause is an optional Model 204 extension. If you specify a SYSNAME value, that value is the file name to which your table is mapped. This is true *regardless* of the SQLFILE setting.

SQLFILE is a Model 204 CCAIN parameter whose default setting of zero turns off system generation of file names. If the SQLFILE value is one, and no SYSNAME clause is specified, Model 204 generates a unique file name for the specified SQL table.

Processing file names

File names you provide explicitly or implicitly are subject to Model 204 file naming rules.

If you are implicitly naming a file (no SYSNAME, SQLFILE=0), the SQL table name you specify is assumed to be the name of the Model 204 file. If this table name does not conform to the Model 204 file naming rules, it is truncated and/or compressed to satisfy the Model 204 rules. (Model 204 file names cannot have more than eight characters, underscore characters, initial numbers, or certain character combinations.)

This assumed name is recorded in the SQL catalog as the Model 204 file name. If the assumed name does not match an existing Model 204 file name, you get an error when your application queries the database with that file in the query.

For example, with no SYSNAME and SQLFILE=0, Model 204 SQL maps the SQL table OUR_OLD_DATA to the Model 204 file OUROLDDA.

If you identify the file explicitly with a SYSNAME clause, the name processing is simpler: if the name you specify in the SYSNAME clause is greater than eight characters, it is truncated to eight characters and stored in the catalog as the file name. Any violations of Model 204 file naming rules are not detected until your application queries the database with that file in the query.

For more information about Model 204 file naming rules, see the *Model 204 File Manager's Guide*.

Using CLOB or BLOB data

The CREATE TABLE statement supports the Character Large Object (CLOB) and Binary Large Object (BLOB) data types.

To define a column containing CLOB or BLOB data, specify the keyword "CLOB" or "BLOB" for the data type in the CREATE TABLE statement.

The following example defines a TABLE called MANUALS that contains all of the manuals for Model204. RECTYPE and NAME are CHARACTER columns that identify the format and the name of the manual. MANUAL is a CLOB column that contains the actual text of the manual.

```
CREATE TABLE MANUALS
( RECTYPE
  CHAR(15),
  NAME
  CHAR(25)
  MANUAL
  CLOB)
```

The following example defines a table containing employee information and a picture of the employee.

```
CREATE TABLE EMPLOYEE
( ID
  CHAR(15),
  NAME
  CHAR(20),
  PICTURE
  BLOB)
```

Defining columns

This section describes column definition syntax and considerations that apply generally to both nested and non-nested SQL tables. Information that is nested-table specific is presented in "Creating nested tables" on page 62.

Column definition statement

Syntax *columnname* <datatype>
 [SYSNAME 'fieldname']
 [<column-constraint> ...]

Parameters where:

- *columnname* conforms to rules for an SQL identifier (see "Naming SQL objects" on page 50).
- *datatype* is the column's data format, the options for which are:

```
CHAR[ACTER] [(length)] | NUM[ERIC]
[(precision [,scale])] | DEC[IMAL]
[(precision [,scale])]
| INT[EGER] | SMALLINT | FLOAT [(precision)]
| REAL | DOUBLE PRECISION | CLOB | BLOB
```

For more information about choosing data types, see “Matching Model 204 and SQL data formats” on page 31.

- *SYSNAME 'fieldname'* maps a column to a Model 204 field; see “Column naming and the SYSNAME extension” on page 58.
- *column-constraint* specifies the following syntax:

```
[NOT NULL] [UNIQUE | PRIMARY KEY [SYSTEM]]
| REFERENCES parent-table-name
[ <referential-triggered-action> ]
```

- *NOT NULL* column maps to a Model 204 field that has a non-null, non-empty value on every record in the Model 204 file. The Model 204 SQL Server does not allow you to violate this rule in an SQL DML update. If you add a NOT NULL column to an existing table, be sure the corresponding Model 204 field has non-null, nonempty values in all the records in the file. For more information about Model 204 SQL handling of nulls, see “Handling NOT NULL, UNIQUE, and multiply occurring data” on page 38.
- *UNIQUE* must map to a Model 204 ORDERED UNIQUE field. For discussion about multiple-column uniqueness, see “Specifying a multicolumn UNIQUE key” on page 58.
- *PRIMARY KEY [SYSTEM]*, where SYSTEM is an extension with which you can have the Model 204 SQL Server generate and manage a unique primary key, as described on “Using system-generated keys” on page 67.
- *REFERENCES* clause for defining a referential integrity constraint is supported for nested tables only, as described on “Nested tables require a referential constraint definition” on page 65.

Syntax rules

Only one PRIMARY KEY clause is allowed per table. Specifying more than one brings an error message.

A primary key call is based on a Model 204 field with unique and ordered attributes.

Unlike the SQL standard, PRIMARY KEY is syntactically independent of NOT NULL. Specifying PRIMARY KEY without NOT NULL is not a syntax error. However, regardless of whether you specify NOT NULL, when you specify PRIMARY KEY, the SQL Server includes NOT NULL checking by default.

Unlike the SQL standard, UNIQUE is independent of NOT NULL. If you specify UNIQUE, NOT NULL is not implied.

The DEFAULT clause for column definition is not supported in Model 204 SQL. If you include a DEFAULT clause in your SQL DDL, it does not become part of the SQL catalog definition, although you receive no syntax error.

The CHECK column constraint is not supported in Model 204 SQL. If you include a CHECK clause in your SQL DDL, it does not become part of the SQL catalog definition, although you receive no syntax error.

Model 204 SQL does support the WITH CHECK OPTION for views, however.

Mapping columns to Model 204 fields

The following are requirements and recommendations for mapping SQL columns to Model 204 fields:

- You cannot map two different columns in the same table to the same Model 204 field.
- To ensure accurate and efficient data handling, you must map Model 204 fields to SQL columns that have compatible data types. For information about specifying data types for selected columns, see “Compatibility of Model 204 and SQL data formats” on page 31. See also “Specifying attributes” on page 115 for additional data type requirements.
- You can map a Model 204 INVISIBLE field to an individual SQL column, but the column’s usability in DML operations is restricted. For example, you can use the column in certain circumstances in the WHERE clause of a SELECT statement but cannot use the column in the SELECT list. You can use the column as the target of an INSERT, but cannot use it as the target or in the source expression of an UPDATE.

For more information about restrictions on DML operations with columns mapped to INVISIBLE fields, see “Using SQL DML against INVISIBLE fields” on page 164.

You cannot map an INVISIBLE field to a column that serves as a primary key nor to a nested table column. You can map an INVISIBLE field to a *multicolumn unique* constraint key, but not to the individual columns that comprise the key. For information about multicolumn unique keys, see “Specifying a multicolumn UNIQUE key” on page 58 and, in the Table Specification facility, “Defining multicolumn unique keys (Multi-Column Unique panel)” on page 120.

- If a Model 204 file is a sorted or hash key file, specify the sort or hash key as a column in the table for the file if the table is a parent or base table and the Model 204 FILEORG parameter has the X'02' option (key required) set. Failure to do so does not result in a DDL error message, but DML file insert attempts fail.

Also for such files, you cannot update the sort or hash key with SQL DML. SQL UPDATE statements fail if a column that maps to the sort or hash key is included.

The Model 204 SQL Table Specification facility *requires* you to specify the sort or hash key as a column. However, if you do not want to include a column for the sort or hash key, you can edit the DDL generated by the TSF (if you are setting up a read-only table or if the sort or hash key is not required, for example).

- Model 204 UNIQUE fields are guaranteed to have unique values but are not guaranteed to have no null values. When you map an SQL UNIQUE column to such a field, the SQL NOT NULL constraint is not implied. This decoupling of UNIQUE and NOT NULL is a Model 204 SQL extension to the SQL standard.

If you include NOT NULL in an SQL UNIQUE column's definition, the SQL Server prevents updates that violate the NOT NULL condition, as well as provides Model 204 uniqueness protection.

Column naming and the SYSNAME extension

Model 204 SQL maps the SQL column you name to a Model 204 field. Unless you designate with the SYSNAME clause the Model 204 field you are mapping to the SQL column, the SQL Server assumes the SQL column name is the Model 204 field name. If your column name does not match the Model 204 field name, you are notified of the error when you attempt a DML query involving the column.

An SQL column name can contain as many as 18 characters (A–Z, 0–9, and underscore), and it must have no embedded blanks. Model 204 field names can have 255 characters, embedded blanks, and a variety of special characters; only certain character combinations are restricted.

SYSNAME allows you to resolve any conflicts between existing Model 204 names and SQL naming rules. Any pre-existing SQL naming format you have can be retained without modification.

For an example of using SYSNAME, see the next section, “Specifying a multicolumn UNIQUE key”, which also describes how multicolumn key names are modified and stored in the SQL catalog.

For more information about Model 204 field naming rules, see the *Model 204 File Manager's Guide*.

Specifying a multicolumn UNIQUE key

You can provide uniqueness checking for an SQL column by mapping it to a Model 204 field that has the ORDERED UNIQUE field attribute. Any SQL operation on such a column that violates its uniqueness in the table is not allowed. To provide such a uniqueness constraint for the combination of the values of two or more columns in a table, you can designate a multicolumn unique key.

In Model 204 SQL DDL, a multicolumn unique key is a constraint key that has no SQL name and is not queriable. You map such a key to a Model 204 UNIQUE index field that your Model 204 file manager must add to the Model 204 file. This special field, which must be ORDERED CHAR, is a concatenation of the fields that correspond to the SQL columns in the key. Directions for defining and populating such a field follow.

Once the key is defined to the SQL catalog, SQL updates to any of the fields automatically update the index field that was added to support the multi-column unique definition. SQL INSERTs add values to the index, DELETEs remove values, and UPDATEs modify values.

Defining the key

You can use the Table Specification facility (see Chapter 5) or manually define the multicolumn unique key in the DDL you submit to the SQL catalog.

To manually define a multicolumn unique key:

- Make sure the definitions of the columns that are to comprise the key include a NOT NULL specification.
- Include a UNIQUE constraint clause after the column definitions. Follow the UNIQUE keyword with a parenthesized sequence of the names of the columns forming the key. The order of the columns in parentheses determines the order in which they are concatenated to build the supporting Model 204 UNIQUE index field.

In the following example, two columns are concatenated to form a unique key.

```
CREATE TABLE SITE
( ORG_ID CHAR(8) SYSNAME 'ORG ID' NOT NULL,
  SITE_ID CHAR(4) SYSNAME 'SITE ID' NOT NULL,
  SITE_NAME CHAR(36) SYSNAME 'SITE NAME' ,
  UNIQUE (ORG_ID, SITE_ID) SYSNAME 'ORG SITE ID INVIS' )
```

In this example, a SYSNAME clause is used in each column definition, including the multicolumn unique key. Although using SYSNAME is optional for the multicolumn unique key, remember that the Model 204 SQL Server generates an assumed field name if you do not specify one with SYSNAME.

Without the SYSNAME clause following the UNIQUE clause in the example above, the Model 204 SQL Server concatenates the individual names ORG_ID and SITE_ID with an ampersand character (&) in between to get ORG_ID&SITE_ID. This name is stored in the SQL catalog as the name of the Model 204 index field.

The maximum number of columns you may concatenate depends on your data. You cannot exceed the Model 204 limit of 255 characters for the combined lengths of the concatenated field values. In addition, you cannot exceed the 255 characters for the key name (either the SYSNAME value for the Model 204 field or, if SYSNAME is not used, the concatenation of the SQL column names).

The Model 204 field that corresponds to the SQL multicolumn unique key is normally INVISIBLE to save Table B space. The key's constituent columns (ORG_ID and SITE_ID in the example above) cannot be mapped to INVISIBLE fields.

Populating the index field

Once the Model 204 index field and multicolumn unique key are defined, you must populate the index. If you are defining a key for a table column that maps to a new Model 204 file, Model 204 SQL automatically populates the index as you insert records through SQL.

If the Model 204 file data already exists, you must populate the index with values that are concatenations of the data values in the fields that are mapped to the columns comprising the key. You can populate the index for an existing file in two ways:

- Automatically, with an SQL UPDATE statement. Use this method if you are maintaining the associated Model 204 file exclusively with SQL.

Issue an SQL UPDATE in which you SET one of the multicolumn unique key columns equal to itself. This triggers an automatic building or rebuilding of the index for any multicolumn unique key that includes the column. If the column is a member of more than one multicolumn unique key, issuing such an UPDATE triggers the rebuilding of all of them.

For example, you can issue the following UPDATE statement to populate the index for the key (ORG SITE ID INVIS):

```
UPDATE SITE SET ORG_ID=ORG_ID
```

As a precaution, make a backup copy of the file before the UPDATE.

- Manually, with Model 204 User Language or Host Language Interface manipulation. Use this method if you are maintaining the associated Model 204 file with User Language or the Host Language Interface. You can use the algorithm described in "Using the multicolumn unique key algorithm".

If you allow updates to the Model 204 file with User Language or the Host Language Interface, you must ensure that you continue to manually maintain the index field.

If you want or need to repopulate an index field, use either of the following methods:

- If the field is INVISIBLE, delete the field (with the Model 204 DELETE command), define it again (with the Model 204 DEFINE command), then repopulate the field using the SQL UPDATE method.
- If the field is not INVISIBLE, flush the old index by redefining (with the Model 204 REDEFINE command) the index field without the ORDERED attribute. Then redefine the field again, this time with the ORDERED CHAR attribute. Finally, repopulate the field using the SQL UPDATE method.

Using the multicolumn unique key algorithm

This algorithm defines for each column in the key the method for generating the field value to be concatenated into the key. The algorithm is affected only by the SQL column attribute; how the data is stored in Model 204 does not matter. Each data value is individually generated as described in Table 4-4. The values are then concatenated as described and stored in the index:

1. For each component field, define the data value to a variable according to column data type, as shown in Table 4-4.

Table 4-4. Multicolumn unique key encoding rules

Column data type	Encoding rules
CHARACTER	Remove trailing blanks, then convert to counted string (character string with one-byte prefix that specifies the number of characters in the string).
INTEGER or SMALLINT	Store as 4-byte IBM binary integer.
DECIMAL	Convert to a right-justified string with length dependent on scale: scale=0: length=precision + 1 scale=precision: length=precision + 3 scale>precision: length=precision + 2
NUMERIC	Same as DECIMAL.
REAL DOUBLE FLOAT	For floating point data, use the SQL UPDATE method described on “Populating the index field” on page 60. Model 204 SQL converts floating point data according to proprietary rules that you cannot reliably duplicate.

2. Concatenate the generated variables, removing trailing blanks (X'40'), leaving embedded blanks, and keeping one blank if the length of the concatenated key after this processing is zero.
3. Store the concatenated value as your index value.

Algorithm example

Build an index value from the following columns and values:

```
COL1 is DECIMAL (7, 2)          (sample value: 10.2)
COL2 is CHARACTER (6)          (sample value: 'JOHN')
COL3 is DOUBLE                  (sample value: 16.0)
COL4 is INTEGER                 (sample value: 1)
```

The concatenated value is:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Creating nested tables

A nested table is a base table with columns that map to Model 204 multiply occurring fields or groups of fields. A nested table is associated with a single base table parent by a unique table column key, which joins the nested table to the parent table.

The NESTED USING clause is the optional clause in a CREATE TABLE statement that identifies the table as a nested table. The abridged CREATE TABLE syntax for a nested table below shows this NESTED clause extension.

Nested tables are Model 204 SQL DDL extensions. There are no extensions to Model 204 SQL DML for using nested tables. For a DML query, nested tables are logically represented like a typical SQL table. Querying nested tables is discussed in Chapter 7.

This section describes the DDL coding rules and requirements for the creation of nested tables. For information about creating nested tables with the TSF, see “Defining nested tables” on page 108. For introductory information about Model 204 nested tables, see Chapter 3.

Nested table statements

Syntax

```
CREATE TABLE tablename NESTED USING columnname
( column-definition | table-constraint-definition
  [ , column-definition | table-constraint-definition ] •••
)
```

Parameters *where:*

- *column-definition* has the following syntax

```
columnname datatype [ SYSNAME 'fieldname' ]
[ column-constraint ] •••
```

- *column-constraint* has the following syntax:

```
NOT NULL [ UNIQUE ]
| REFERENCES parent-table-name
[ referential-triggered-action ]
```

referential triggered action has the following syntax:

```
ON UPDATE CASCADE [ ON DELETE CASCADE ]
| ON DELETE CASCADE [ ON UPDATE CASCADE ]
```

- *table-constraint-definition* has the following syntax:

```

unique-constraint-definition
| referential-constraint-definition

unique-constraint-definition has the following syntax:
UNI QUE ( col umname ) [ SYSNAME ' fi el dname' ]

referential constraint definition has the following syntax:
[FOREI GN KEY ( col umname)
REFERENCES parent-table-name
[referenti al -tri ggered-acti on]]

referential triggered action has the following syntax:
ON UPDATE CASCADE [ ON DELETE CASCADE ]
| ON DELETE CASCADE [ ON UPDATE CASCADE ]

```

Syntax rules

Unless otherwise specified, all the rules that apply to tables apply also to nested tables.

Rules for nested table columns

Only one column can be referenced by a NESTED clause, and only one NESTED clause is allowed per table.

Nested table must have one foreign key column, and can have no more.

Nested table must have at least one column in addition to the foreign key column.

All columns in a nested table must be defined as NOT NULL.

Column in a nested table cannot be mapped to a field with the Model 204 field attribute INVISIBLE.

Column in a nested table may not be mapped to a field with the Model 204 field attribute UPDATE AT END.

Multicolumn composite UNIQUE key is not allowed for a nested table.

Rules for PRIMARY KEY and FOREIGN KEY

Parent table that a foreign key refers to with the REFERENCES clause must have a PRIMARY KEY. This PRIMARY KEY can be a system-generated key.

- A PRIMARY KEY must map to a Model 204 UNIQUE ORDERED field.
- PRIMARY KEY cannot map to a Model 204 INVISIBLE field.

If a foreign key is defined twice, first with a REFERENCES clause and then with a FOREIGN KEY clause, the statement is accepted only if the two clauses are identical and reference the same column.

SQL error message -4703 is generated if a FOREIGN key is defined twice for a nested table and the two keys do not reference the same column. For example:

Acceptable:

```
CREATE TABLE NESTEDINVENTOR2 NESTED USING PART_NO
(PART_NO
  DECIMAL(8) NOT NULL
  REFERENCES INVENTORY,
ON_HAND
  SYSNAME 'ON HAND'
  FLOAT(4) NOT NULL,
LOCATION
  CHAR(255) NOT NULL,
FOREIGN KEY (PART_NO) REFERENCES INVENTORY)
```

Returns SQL error -4703:

```
CREATE TABLE NESTEDINVENTOR2 NESTED USING PART_NO
(PART_NO
  DECIMAL(8) NOT NULL
  REFERENCES INVENTORY,
ON_HAND
  SYSNAME 'ON HAND'
  FLOAT(4) NOT NULL,
LOCATION
  CHAR(255) NOT NULL,
FOREIGN KEY (ON_HAND) REFERENCES INVENTORY)
```

Data type of the FOREIGN KEY of a nested table must match exactly (both in type and length) the data type of the PRIMARY KEY of the referenced table. Violation causes an error message to be issued.

When paired in a nested table relationship, PRIMARY KEY and FOREIGN KEY can specify only single-column keys. For multicolumn primary key functionality, use PRIMARY KEY SYSTEM.

For more information about foreign keys, see “Nested tables require a foreign key” on page 65. Using system generated keys is discussed on “Using system-generated keys” on page 67. For more information about multiple-column primary keys, see “Simulating multicolumn primary keys” on page 68.

Rules for SYSTEM and SYSNAME

SYSTEM keyword is optional; but if SYSTEM is specified, PRIMARY KEY must also be specified. The column associated with PRIMARY KEY SYSTEM must have a data type of INTEGER.

SYSNAME cannot be specified with PRIMARY KEY SYSTEM. SYSTEM causes the mapping of the column to a system-generated field. Specifying SYSNAME, which implies a link to a Model 204 field name, conflicts with this automatic mapping. Violation causes an error message to be issued.

Neither SYSTEM nor SYSNAME can be specified with the foreign key column of a nested table. Violation causes an error message to be issued.

SYSTEM and SYSNAME are Model 204 SQL extensions. Using SYSTEM is discussed in “Using system-generated keys” on page 67; using SYSNAME is discussed on “Mapping table names to file names” on page 53 and on “Column naming and the SYSNAME extension” on page 58.

Mapping multiply occurring groups

Nested table rows are ordered and retrieved by matching occurrence: the first row is mapped to the first occurrences of the repeating field values, the second row is mapped to the second occurrences of the repeating field values, and so on. The rank of occurrence (first, second, and so on) is determined by physical storage order in the file. For an example showing this row-to-occurrence mapping, see “Simulating normalization of Model 204 record data” on page 28.

To preserve the matching occurrence ordering, each member of a group of multiply occurring values mapped to nested table columns must have the same number of occurrences. Otherwise, retrievals of these columns are not reliable. In addition, since null values invalidate matching occurrence ordering, nested table columns must be NOT NULL.

If you want to add a new column to an existing nested table, the column must have as many rows as the other table columns and must have a value for each of the existing rows.

Nested tables require a foreign key

A nested table must have a foreign key column. A foreign key is equivalent to its related primary key in the parent table. The unique values of the primary or foreign key are used to locate the Model 204 records with the repeating field values.

You cannot update the foreign key directly with SQL DML. Updates to the primary key are propagated to the foreign key. You must specify the foreign key in an SQL INSERT; it is used to locate the parent record.

Examples of specifying the foreign key in DDL follow.

Nested tables require a referential constraint definition

A referential constraint definition (REFERENCES clause) is required in conjunction with the NESTED clause to define a nested table and its relationship to its parent. This constraint protects SQL database integrity by ensuring that all the values of a column in the nested table (the foreign key) match all the values of the primary key of the referenced parent table. Operations that violate a defined referential constraint are not allowed.

Only one referential constraint is allowed per table. If you specify two REFERENCES clauses for a single table (one in a column definition and one in a foreign key definition), the clauses must be the same and reference the same column.

How you specify a referential constraint depends on whether you are using the column definition or table constraint definition option of the Model 204 SQL CREATE TABLE statement. Examples of each follow. In addition, the format changes slightly if you are using a system-generated primary key. Examples with system-generated primary keys are in "Using system-generated keys" on page 67.

Table constraint format

There must be a FOREIGN KEY clause that refers to the column (the foreign key) specified in the NESTED clause. The FOREIGN KEY clause must be followed by a REFERENCES clause referring to the parent table. For example:

```
CREATE TABLE PEOPLE
( NAME CHAR(60) NOT NULL PRIMARY KEY
  HIRE_DATE CHAR(8) )

CREATE TABLE TASKS NESTED USING FNAME
( TASK CHAR (25) NOT NULL,
  FNAME CHAR (60) NOT NULL,
  FOREIGN KEY (FNAME) REFERENCES PEOPLE )
```

Column constraint format

The column that is specified in the NESTED clause must have a REFERENCES clause referring to the parent table in the column definition. For example:

```
CREATE TABLE PEOPLE
( NAME CHAR(60) NOT NULL PRIMARY KEY,
  HIRE_DATE CHAR(8) )

CREATE TABLE TASKS NESTED USING FNAME
( TASK CHAR (25) NOT NULL,
  FNAME CHAR (60) NOT NULL REFERENCES PEOPLE )
```

CASCADE is the only referential triggered action

A referential triggered action is a delete rule (ON DELETE action) and/or an update rule (ON UPDATE action) that governs what action a system takes if a referential constraint is violated. A system can refuse to execute a constraint-violating request, or it can cascade (that is, can automatically execute an operation that compensates for the violation).

For example, if you change a value of the primary key of the parent table referenced by a nested table, you violate the referential integrity (REFERENCES clause) of the foreign key. A system can protect the referential integrity by preventing you from changing the primary key value. Or a system can cascade, automatically associating the nested table rows that refer to the changed primary key value with the new primary key value. The Model 204 SQL Server uses only the cascade action; it does not prevent you from changing the primary key value.

If you change a value of the primary key of the parent table from *n1* to *n2*, the Model 204 SQL Server changes all nested table foreign key references from *n1* to *n2*.

A referential triggered action is specified with a REFERENCES clause used in a nested table definition. For the column in a nested table defined in the NESTED clause, the only Model 204 SQL Server referential triggered action that you can specify is the CASCADE action. Updates and deletes are automatically cascaded.

If you do not specify an update or delete rule, the rule is set to CASCADE by default.

Using system-generated keys

A primary key is a column that has values that uniquely identify each record in the table. A nested table parent is required to have such a column. If no single column is a unique identifier in a nested table parent, or if a combination of two or more columns comprise the unique identifier, you must have the Model 204 SQL Server generate and manage a unique primary key.

This system-generated key is a nonupdatable, unique integer key automatically assigned to each row occurrence by the system when the row is inserted. In fact, this key is the Model 204 internal record number, and other than being nonupdatable, it is like any other relational column. You can retrieve it with SELECT and use it in predicates. You can give it any name you want.

In some queries, system-generated keys might be more efficient than specified single-column keys, because they provide direct access to the data. However, the system-generated values for SYSTEM keys are not necessarily preserved across a file reorganization: after a reorganization, the value for a system-generated column might change for any particular row occurrence. Because of this impermanence, avoid operations with the value of this column if the result of the operations is required for longer than the current session.

Defining system-generated keys

To define a system-generated key, specify the SYSTEM modifier to the PRIMARY KEY constraint in the parent table column definition. The following examples use the column constraint format of CREATE TABLE.

Example 1 - Primary key is not system-generated: Primary key is not system-generated. This example is repeated from “Column constraint format” on page 66. Here NAME is a unique identifier for each record in the parent table.

```
CREATE TABLE PEOPLE
( NAME CHAR(60) NOT NULL PRIMARY KEY,
  HIRE_DATE CHAR(8) )

CREATE TABLE TASKS NESTED USING FNAME
( TASK CHAR (25) NOT NULL,
  FNAME CHAR (60) NOT NULL REFERENCES PEOPLE )
```

The column name used for the primary key in the parent table is different from its related foreign key in the nested table, but this is not a requirement in SQL.

Example 2 - Primary key is system-generated: Primary key is system-generated. In this example, assume NAME and HIRE_DATE together form a unique identifier for each record in the parent table. Such a composite key requires you to use a system-generated primary key, because the primary key for nesting must be a single column.

To define a system-generated key, specify a name for the system-generated primary key in the parent table column definition, and follow the token PRIMARY KEY with the modifier SYSTEM:

```
CREATE TABLE PEOPLE
( PKEY INTEGER NOT NULL PRIMARY KEY SYSTEM,
  NAME CHAR(60) NOT NULL
  HIRE_DATE CHAR(8) )

CREATE TABLE TASKS NESTED USING PKID
( TASK CHAR (25) NOT NULL,
  PKID INTEGER NOT NULL REFERENCES PEOPLE )
```

Notice that the FNAME column definition included in example 1 is not included in the definition in example 2. In example 1, FNAME is the foreign key, that is, it is equivalent to the primary key NAME. In example 2, PKEY replaces NAME as the primary key. A foreign key in TASKS equivalent to PKID must be included.

For information about multicolumn keys for nonnested tables, see “Specifying a multicolumn UNIQUE key” on page 58.

Simulating multicolumn primary keys

As stated in example 2, Model 204 SQL does not allow you to define a multicolumn primary key in conjunction with a nested table foreign key. In such a case, however, you can use a system-generated primary key to get the functionality of a multicolumn primary key.

In addition to the definition of the system-generated primary key shown in example 2, add a uniqueness constraint definition for the columns that together form a unique identifier for each record in the parent table. (This constraint requires designating HIRE_DATE as NOT NULL.)

```
CREATE TABLE PEOPLE
( PKEY INTEGER NOT NULL PRIMARY KEY SYSTEM,
  NAME CHAR(60) NOT NULL
  HIRE_DATE CHAR(8) NOT NULL
  UNIQUE (NAME, HIRE_DATE) SYSNAME 'HIRE ID' )
```

The system-generated key satisfies the Model 204 SQL requirement for a single-column unique primary key. The multicolumn unique key definition for NAME and HIRE_DATE preserves their uniqueness.

To SELECT the composite key columns (NAME and HIRE_DATE) and the multiply occurring column (TASK), you need to use the following joined-table SELECT instead of a simple non-joined SELECT:

```
SELECT NAME, HIRE_DATE, TASK
FROM PEOPLE, TASKS
WHERE PKEY=PKID
```

Inserts into tables with system-generated keys

You cannot insert column values into a table that has a system-generated key unless you specify the target-column list. For example, the following insert into table PEOPLE, whose primary key PKEY is system-generated, is valid. Model 204 SQL automatically provides a value for PKEY:

```
INSERT INTO PEOPLE (NAME, HIRE_DATE)
VALUES ('CJDATE', '10/01/90')
```

SQL syntax rules dictate that inserts without a column list require you to provide values for all the columns in the table. Because you cannot provide the primary key value (because PKEY is system-generated and you cannot update a system-generated key), removing (NAME, HIRE_DATE) from the example above causes the insert to fail.

These rules for inserts into tables that have system-generated keys apply to base and parent tables. For inserts into nested tables that have system-generated keys, you *must* provide the primary key value from the parent table.

Creating views

Unlike tables and columns, views do not map directly to Model 204 files and fields. A view is a selected set of columns and rows from one or more SQL tables or views that can be displayed as a unit. A view contains no data but instead is defined to access data in one or more tables.

The Model 204 SQL CREATE VIEW statement includes extended SELECT functionality that enables the simulation of Model 204 groups.

For a discussion and detailed example of the DDL statement security considerations affecting view creation and access, see “Model 204 SQL view privileges” on page 89.

CREATE VIEW statement

Syntax `CREATE VIEW <viewname> [(<column-list>)]`
 `AS <query-expression> [WITH CHECK OPTION]`

Parameters where:

- *viewname* conforms to rules for SQL identifier.
- *column-list* has the following syntax:

 `columnname [,columnname] ...`

 query-expression
- A *query-specification* or a UNION of query specifications, where query specification is a SELECT statement (with no ORDER BY clause).
- *WITH CHECK OPTION* checks DML inserts and updates to ensure that they do not violate view definition conditions.

Syntax rules The CREATE VIEW syntax rules are:

- Query specifications joined by the UNION operator may contain no joined tables and cannot include GROUP BY or HAVING.
- Query specifications joined by the UNION operator cannot contain an EXISTS clause.

Rules for updating views

The following rules govern whether you can update a particular view.

- The definition of the view cannot include the UNION keyword or the DISTINCT keyword.
- You can refer to only one table in the FROM clause.
- If the CREATE VIEW statement includes a WHERE clause, the WHERE clause cannot include a subquery.
- The CREATE VIEW statement cannot include a GROUP BY clause or a HAVING clause.
- If the CREATE VIEW statement includes a SELECT clause, the SELECT clause cannot contain expressions.

This means that DDL used to define a view cannot grant the privileges UPDATE, INSERT, or DELETE to any view that ignores the “Rules for updating views” on page 70. For example, the following view DDL generates an error.

```
SET SCHEMA DEMO
SET USER AGENT1
DROP VIEW TEST_VIEW
CREATE VIEW TEST_VIEW (TEST_ID, POLICY_NO)
    AS SELECT TEST_TABLE.TEST_ID, CLIENTS.POLICY_NO

    FROM TEST_TABLE, CLIENTS

GRANT SELECT, UPDATE, DELETE, INSERT ON TEST_VIEW TO ADMIN
```

Because you cannot update a view that is based on more than one table—TEST_TABLE and CLIENTS—this DDL generates the following error:

```
SQL Error -551 AGENT1 does not have the privilege to perform operation GRANT on object DEMO.TEST_VIEW.
```

However the following GRANT SELECT clause is valid:

```
GRANT SELECT ON TEST_VIEW TO ADMIN
```

Only the updating clauses—UPDATE, INSERT, DELETE—are disallowed.

Guideline for view definitions

The following general rule for view definitions governs what is allowed in a view definition and how you can select against that view:

- Query in a DML SELECT against a view is evaluated by substituting the view definition for the references to the view in the DML SELECT. The query that results from this substitution or translation process must always be a valid SQL SELECT statement.

Using SQL views in Model 204 SQL DDL

Use SQL views for the following operations:

- To implement more efficiently your security strategy

Once a view is defined, the Model 204 SQL Server checks only the privileges to access the view itself, not the privileges for each of the elements of the view. For example, you might want to secure data by record type in a mixed record type file. After defining the views by record type, you can then limit access to each type of record by selectively granting privileges to access each view.

For more information about privileges for views, see page 4-47.
- To provide security that is similar to Model 204 field level security

Using GRANT to selectively permit updates to individual columns of a table can be degrading to DML processing performance. You can instead define a view with these columns and grant limited access to the view. Such a view is easier to define and maintain and is the only way to also grant SELECT access to just those columns.

- To simulate Model 204 file groups

File groups are not directly supported in Model 204 SQL DDL. However, you can use a Model 204 SQL CREATE VIEW extension to create views that simulate Model 204 groups. See “Simulating file groups” on page 72.

- To map Model 204 files that contain mixed record types

Define individual tables to the SQL catalog that map to files that have a variety of types of records. You can isolate the individual record types by defining a separate view for each one. See “Mapping files with mixed record types” on page 73.

Simulating file groups

An SQL table cannot be mapped to a Model 204 file group. You can simulate a group, however, by creating a view that is comprised of a concatenation of tables that are mapped to the files in the group. The following requirements apply to any Model 204 file group that SQL accesses.

- The first file in the file group must have the same name as the group.
- The file with the same name as the group must include field definitions for all the fields that exist in any other file in the group.
- The file with the same name as the group can be empty except for the field definitions.

First, map an SQL table to each of the files in the Model 204 group. Then create a view defined as a UNION (or UNION ALL) of SQL SELECT statements, each of which selects all the rows in one of the tables mapped to the group. This union of SELECT statements is extended functionality to the CREATE VIEW statement.

The following example shows three table definitions and then a view definition based on those tables that simulates a Model 204 file group:

```
CREATE TABLE POLICIES_89
  ( POL_NO INTEGER, ACC INTEGER, STATE CHAR(2) )

CREATE TABLE POLICIES_90
  ( POL_NO INTEGER, ACC INTEGER, STATE CHAR(2) )

CREATE TABLE POLICIES_91
  ( POL_NO INTEGER, ACC INTEGER, STATE CHAR(2) )

CREATE VIEW MA_POLICIES (POLICY_NO, ACCIDENTS) AS
```

```

SELECT POL_NO, ACC FROM POLICIES_89 WHERE STATE = 'MA'
UNION ALL
SELECT POL_NO, ACC FROM POLICIES_90 WHERE STATE = 'MA'
UNION ALL
SELECT POL_NO, ACC FROM POLICIES_91 WHERE STATE = 'MA'

```

You can use SELECT statements against the view to query the “group” of tables. However, you *cannot* update the file group through these views. An example of a SELECT statement against the view is:

```

SELECT POLICY_NO FROM MA_POLICIES WHERE ACCIDENTS > 5

```

Mapping files with mixed record types

An example of views defined for mixed record type files follows. The example shows manually generated DDL that defines two views of the CLIENTS file from the Model 204 demonstration database. After mapping all the fields (regardless of record type) to a single base table, you define a view for each record type:

```

CREATE SCHEMA AUTHORIZATION GEORGE
CREATE VIEW DRIVERS
    (DATE_OF_BIRTH, DRIVER_ID, FULLNAME, MARITAL_STATUS,
     POLICY_NO, SEX, STATE) AS
SELECT
    DATE_OF_BIRTH, DRIVER_ID, FULLNAME, MARITAL_STATUS,
    POLICY_NO, SEX, STATE
FROM CLIENTS
WHERE RECTYPE = 'DRIVER'
GRANT ALL PRIVILEGES ON DRIVERS TO PUBLIC

CREATE VIEW POLICIES
    (ADDRESS, AGENT, ANNIV_DATE, CITY, DATE_OF_BIRTH,
     FULLNAME, POLICY_NO, POLICYHOLDER, STATE,
     TOTAL_PREMIUM, ZIP) AS
SELECT
    ADDRESS, AGENT, ANNIV_DATE, CITY, DATE_OF_BIRTH,
    FULLNAME, POLICY_NO, POLICYHOLDER, STATE,
    TOTAL_PREMIUM,
    ZIP
FROM CLIENTS
WHERE RECTYPE = 'POLICYHOLDER'
GRANT ALL PRIVILEGES ON POLICIES TO PUBLIC

```

For the sake of simplicity, these views do not include the CLIENTS nested table columns. These columns map to multiply occurring Model 204 fields.

Maintaining views

You are responsible for ensuring that your views remain valid over time. The Model 204 SQL Server does not warn you when a view is invalidated and does not prevent you from issuing DDL that renders a view invalid.

The DDL statements you can use for views are:

```
CREATE VIEW
DROP VIEW
GRANT
REVOKE
```

You can change the roster of users that can access a view (with GRANT and REVOKE), but you cannot change the view definition itself. Once a view is defined, you can modify it only by deleting it (with DROP VIEW) and redefining a new one (with CREATE VIEW).

You can modify objects that the view references, but the modifications are not propagated to the view definition itself. For example, if you drop a table that is referenced by a view, the change is not propagated to the view definition. The view becomes invalid, yet the view definition remains in the catalog. You are not notified that your view is invalid until the time of DML query validation.

The Model 204 SQL Server does not delete invalid views from the SQL catalog unless you explicitly drop the view or drop the schema to which the view belongs.

Querying views

Queries against views whose definitions contain UNION ALL, GROUP BY, HAVING, or SELECT DISTINCT have the following restrictions.

Views defined with UNION ALL

SELECT statements in a query expression that contains the UNION operator cannot reference any views that have definitions that contain the UNION operator.

An SQL DML statement

- That references a view that has a UNION operator cannot itself use a UNION operator.
- Cannot reference in a subquery a view that has a UNION operator.
- That references a view that has a UNION operator cannot use that view in a join expression. That is, no other view or table can be specified in the FROM clause of the query specification used in the DML statement.

An SQL SELECT statement

- That references a view that has a UNION operator cannot apply the GROUP BY clause nor any of the aggregating functions (COUNT, AVG, MAX, MIN, SUM).

Views defined with GROUP BY, HAVING, or SELECT DISTINCT

Query against a view defined with

- GROUP BY, HAVING, or both, can have no other views or tables in the FROM clause, and cannot have a WHERE, GROUP BY, or HAVING clause.
- SELECT DISTINCT can have no other views or tables in the FROM clause, and cannot have a GROUP BY or HAVING clause. Such queries must specify SELECT *.
- GROUP BY along with SELECT DISTINCT cannot have WHERE, GROUP BY, or HAVING clauses.

Setting the schema and user context

This section describes how to determine, indicate, and change the schema and user context, and introduces the SET SCHEMA and SET USER statements.

Determining the default schema context

At the beginning of an SQL session, the default schema context (name) is the Model 204 user ID established at login (with trailing blanks removed). The default schema name is assigned to any SQL objects you indicate in your DDL for the entire session, unless you specify another schema name. This default context is in effect except when a CREATE SCHEMA transaction is active (see “Indicating schema name and owner” on page 51). You can change the default context with the SET SCHEMA statement.

Prefixing the schema name to an SQL object

After SQL objects are created, you may need to make adjustments to your schemas to accommodate new users, changes to privileges, or new data. To revise SQL objects already defined in the SQL catalog, you cannot use CREATE SCHEMA, because it can be used only once per schema, that is, when you initially create the schema. Issuing a CREATE SCHEMA that names an existing schema is an error.

To selectively modify SQL objects that reside in different schemas, you need to identify the schema to which the objects belong. You can do so by specifying the schema name as qualification along with the name of the object (table, view) you are adding or modifying. For example:

schemaname.tablename

You can qualify the name of a column with a table name or with a table name and a schema name. For example:

```
schemaname.tablename.columnname
```

If you do not specify the schema name along with the object, the schema name assigned is the current default schema.

You can qualify a table or view name only with the schema name that is the current default, if you are issuing CREATE TABLE or CREATE VIEW as part of a CREATE SCHEMA transaction. This restriction does not apply to GRANT statements in a CREATE SCHEMA transaction.

You can reset or change the current default schema by using SET SCHEMA.

Using SET SCHEMA

You can change the default schema name in SQL DDL or DML by using the following Model 204 SQL extension statement:

```
SET SCHEMA schemaname
```

SET SCHEMA defines the current default schema context. Statements following SET SCHEMA are assumed to apply to this schema. This default remains in effect for the entire session or until reset by another SET SCHEMA statement.

SET SCHEMA allows you to avoid continual specification of the schema name with SQL objects you are modifying or using. If no schema name is appended to an SQL object you add or modify after issuing SET SCHEMA, the schema name of the object defaults to the current setting of SET SCHEMA.

Any user can issue SET SCHEMA, because it has no effect on a schema definition other than establishing the context. Once the schema context is set, however, permission to operate with DDL or DML on the SQL objects in the schema depends on the individual statement. The privileges required to issue individual Model 204 SQL DDL statements are summarized on “DDL statement-level security” on page 86.

Using SET USER

In Model 204, authority to issue commands is based on the user role or type. A system manager (determined by login ID) typically is the pivotal user with greatest authority. In SQL, authority is based on object ownership, the authorization ID of the schema to which an object belongs. Model 204 SQL statement security combines these characteristics: to create schemas and tables, you must be a system manager and your login ID must match the authorization ID for the schema to which the object belongs.

However, this approach has the following drawback: to create a schema that has an SQL authorization ID other than the system manager's, the system

manager has to give system manager privileges to that SQL user. The Model 204 SQL extension statement SET USER resolves this drawback.

SET USER, available to system managers only, changes the current SQL user context. A system manager issues SET USER ABC and in effect acquires the SQL authorization ID ABC and its associated SQL privileges. At the same time, the system manager retains Model 204 login ID privileges.

SET USER thus enables a kind of superuser, who can issue SQL statements for another SQL user without having to log in as that user or give that user system manager privileges. The system manager gains immediate access to, and authority to change, all defined SQL objects.

The SET USER syntax is:

Syntax `SET USER authorization-id`

SET USER can be used with SQL DML or DDL statements. For an example showing how SET USER is used, see “SQL statement security example” on page 89.

Altering SQL objects

ALTER TABLE allows you to change the definition of a table. New columns may be added with the ADD clause. Existing columns may be modified with the MODIFY clause. Columns may be removed using the DROP clause.

ALTER TABLE also lets you shift the relative positions of the columns in a table. To protect against losing track of the column positions, you should always specify the column names when issuing an SQL INSERT.

ALTER TABLE statement

Syntax `ALTER TABLE tablename
 ADD column-definition
 | DROP columnname
 | MODIFY column-parameters`

Parameters where:

- *column-definition* has the following syntax:

```
columnname <datatype> [ SYSNAME 'fieldname' ]  
[ <column-constraint> ] ...
```

column-constraint has the following syntax

```
[ NOT NULL ] [ UNIQUE | PRIMARY KEY [ SYSTEM ] ]  
| REFERENCES parent-table-name  
[ <referential-triggered-action> ]
```

Individual parameters are described on “CREATE TABLE statement” on page 52 and “Creating nested tables” on page 62.

- *column-parameters has the following syntax:*

```
columnname [datatype] [SYSNAME 'fieldname']  
[[NOT] NULL | [NOT] UNIQUE]
```

- *MODIFY* is discussed in “Using MODIFY column” on page 79.
- *NOT NULL* column must map to a Model 204 field that has a non-null, nonempty value on every record in the Model 204 file. The Model 204 SQL Server does not allow you to violate this rule in an SQL DML update. If you add a NOT NULL column to an existing table, be sure the corresponding Model 204 field has non-null, nonempty values in all the records in the file. For more information about Model 204 SQL handling of nulls, see “How Model 204 SQL processes dirty data” on page 37.

Syntax rule

As described on “Prefixing the schema name to an SQL object” on page 75, the name of the table can be optionally specified along with the schema name as qualification:

```
schemaname.tablename
```

Using ADD column

ADD adds a column to a table but does not update any view definitions that reference the table.

Use ADD for adding columns to a table when column position in the table is not important.

With ADD, all columns are added to a table in the last position of the column list. If you DROP a column and then ADD an updated version of that column, the updated column occupies a different position in the table than it did before you executed DROP and ADD. Such a change in order of the column data can introduce errors into queries that use SELECT * or INSERT (without a column list) and that depend on the correct position of the column data.

Using DROP column

Dropping a column deletes the column from the table and deletes any privileges granted for this column. It has no effect on any view definitions that reference this column.

Note: Because tables cannot be left empty, you cannot drop the last column in a table. Because nested tables cannot be left with only a foreign key column, you cannot drop the last nonforeign key column in a nested table.

If you want to drop a referenced primary key column, you must first drop the nested table that contains the REFERENCES clause; if not, you receive an error message.

You cannot DROP a column that is part of a multicolumn unique key.

Using MODIFY column

Use MODIFY for the following operations:

- To change the UNIQUE or NOT NULL status of a column.
- To change a column definition other than making an addition or a deletion, use MODIFY instead of using DROP and ADD.
- To change a column's data type or field mapping clause or attributes.
- When the position of the column data in a table must not be disturbed.

MODIFY affects only the column definition elements you specify; the rest of the definition remains as is. For example, for a column originally defined as INTEGER NOT NULL that you want to change to DECIMAL (11,2) NOT NULL, specify only:

```
ALTER TABLE tablename MODIFY columnname DECIMAL (11,2)
```

You cannot ALTER or MODIFY an SQL object that does not already exist. You cannot ALTER or MODIFY a column that is part of a multicolumn unique key.

Only modify the UNIQUE or NOT NULL status of a column if the table is empty. If the UNIQUE or NOT NULL status of a column is modified on a nonempty table, you must ensure that this definition is compatible with the data in the existing Model 204 file. If it is not, an error might occur when processing an SQL request against this file.

Note: If more extensive modification to a table definition is required, you can drop the table and redefine it using CREATE TABLE as you did initially to set up the table. Remember, the Model 204 SQL catalog is not active, so dropping a table does not affect the actual file data.

Dropping SQL objects

You can delete SQL objects from the SQL catalog with the Model 204 SQL DROP statements and clause listed below. These deletion statements are discussed in turn in this section.

```
DROP TABLE <tablename>
DROP VIEW <view name>
DROP SCHEMA <schemaname>
```

Reminder : You can avoid an inadvertent or unanticipated loss of data due to deletion of SQL objects by backing up CCACAT before executing a DROP or

by using the catalog reporting utility (CCACATREPT). With CCACATREPT you can generate a copy of the catalog DDL before you execute a DROP.

CCACATREPT is discussed in Chapter 6. For more information about backing up CCACAT, see “Backup and restore” on page 18.

Dropping tables

DROP TABLE causes the following actions to occur:

- Catalog entry for the table and its columns is deleted.
- Any privilege and constraint records that reference this table are deleted.

Once DROP TABLE eliminates all entries for a table from the SQL catalog, the table no longer exists in the SQL catalog. However, the Model 204 file associated with the table remains unaffected.

To avoid catalog data inconsistencies, drop SQL objects that depend on other objects before you drop the objects that are depended upon. For example, you must drop nested tables that reference a parent table before you drop the parent. Otherwise, your DROP TABLE statement is rejected. Also, drop any views associated with a table before you drop the table.

For convenience, you can qualify the table name by prefixing the schema name.

Dropping views

Views are permanent objects but do not map to Model 204 files or fields directly. The view's definition in terms of other tables or views is stored in the catalog in the form of a view record.

When you drop a view, the view record and all privilege records (records of users granted access to the view) associated with the dropped view are deleted from the catalog. DROP VIEW has no effect on the base table(s) associated with the view.

The only time the Model 204 SQL Server deletes a view from the SQL catalog is when you explicitly drop it or drop the schema to which it belongs. You can render a view invalid by deleting objects referenced by the view, but the invalid view definition remains in the catalog. You are notified of the invalidity when you next attempt to access the data files through this view.

For convenience, you can qualify the view name by prefixing the schema name.

Dropping schemas

Dropping a schema deletes the SCHEMA record in the catalog and all TABLE, PRIVILEGE, and CONSTRAINT records that reference this schema.

Dropping a schema also deletes views belonging to the schema and all PRIVILEGE records associated with them.

Granting privileges for SQL objects

To perform an operation on an SQL object, you must hold the necessary privilege for that combination of operation and object. That privilege might result from ownership of the object or from being granted that privilege by another user (with the GRANT statement). You can change privilege assignments by adding privileges with subsequent GRANT statements or by deleting privileges with the REVOKE statement.

This section provides the statement syntax for GRANT and REVOKE and discusses elements of their use that are special to Model 204 SQL.

GRANTs are for adding privileges

GRANT statements always and only add privileges. If you want to change privilege assignments, you can add privileges with subsequent GRANT statements. To delete or diminish the current level of privileges you must use REVOKE, not GRANT. The Model 204 SQL Server allows a new GRANT for an object to replace an earlier GRANT for that object only to the extent that the new GRANT expands the current set of privileges.

For example, if you try to reduce the current privileges for an object by issuing a new GRANT that allows fewer privileges, the new GRANT is ignored. To reduce the scope of given privileges you must use REVOKE.

GRANT and REVOKE handle nearly all SQL security

SQL access to a Model 204 file is protected exclusively by Model 204 login security and Model 204 SQL GRANT and REVOKE statements. Existing Model 204 file access security is not enforced by the Model 204 SQL Server.

Since GRANT and REVOKE are the principal security sources, the final SQL file access safeguard is SQL statement security. That is, you can permit only certain users per SQL object to issue GRANT and REVOKE statements.

For more information about the privileges required for execution of the individual Model 204 SQL DDL statements, see “DDL statement-level security” on page 86.

GRANT statement

The GRANT statement is the *privilege definition* option of the CREATE SCHEMA statement. The syntax is:

Syntax

```
GRANT <privileges> ON <object-name>
  TO <grantee> [, <grantee> ] ...
  [ WITH GRANT OPTION ]
```

Parameters

where:

- *privileges* has the following syntax:

```
ALL PRIVILEGES | action [,action ] ...
```

- *action* has the following syntax:

```
SELECT | INSERT | DELETE  
| UPDATE [ ( columnname [,columnname ] ... ) ]
```

- *object-name* is the table or view name.

- *grantee* has the following syntax:

```
PUBLIC | authorization-id
```

Usage notes

The privileges you can grant apply only to DML operations for specified SQL objects or to defining which users can issue additional GRANT statements for specified SQL objects. For information about privileges for issuing DDL statements, see “DDL statement-level security” on page 86.

Model 204 SQL DDL has no REFERENCES privileges.

Unlike the other privileges, the UPDATE option can be applied to a specified list of columns. If no column list is specified with UPDATE, by default it is assumed that all columns in the table are included.

You can use UPDATE to provide privileges for selective access to certain columns in the database. However, using UPDATE with a column list typically yields poorer performance than using a view of these columns and granting selective access via the view.

You cannot grant UPDATE privileges on the columns of a system-generated primary key, because such a key by definition cannot be updated.

Whether you specify UPDATE by itself or specify UPDATE followed by a list of the columns in the table, you can affect subsequent privilege assignments. This is discussed further in “Granting and altering column UPDATE privileges” on page 83.

REVOKE statement

The REVOKE statement is a Model 204 SQL extension with which you can revise the privileges given by the GRANT statement. A table's owner can REVOKE privileges for any authorization ID. Other users can REVOKE privileges for those rights they were granted with the WITH GRANT OPTION. The syntax is:

Syntax

```
REVOKE [ GRANT OPTION FOR ] privileges  
ON object-name  
FROM grantee [,grantee ] ...
```


Parameters

where:

- *privileges* is the same as for the GRANT statement.
- *object-name* is the same as for the GRANT statement.
- *grantee* has the following syntax:

```
PUBLIC | authorization-id
```

Usage notes

Each time you issue REVOKE, it revokes one of the following:

- Entire privilege (SELECT, UPDATE, INSERT, or DELETE) or list of privileges
- Ability of the specified user or users to grant this privilege to another user

Revocation of privileges does not cascade. That is, if your privileges to grant updates on a particular table are revoked, the update privileges for that table you can have granted to other users are not revoked. For example, USERA grants update privileges with grant option on TABLET to USERB, and USERB grants the same privileges to USERC. If USERA later revokes USERB'S TABLET privileges, USERC'S TABLET privileges are *not* affected.

Similarly, if USERB's privileges to grant updates on TABLET are revoked, the revoking action does not cascade to VIEWV, which references TABLET. USERB can still use VIEWV, and USERC can still use VIEWV.

You receive an error message if you issue REVOKE against an unauthorized user (or against an authorized user whose name is misspelled).

Granting and altering column UPDATE privileges

The REVOKE and ALTER TABLE statements in Model 204 SQL DDL allow for changes over time to the columns of a table and to the privileges for updating those columns. To avoid unwanted effects from such changes over time, you need to understand how the Model 204 SQL Server handles grants of column updating privileges.

The Model 204 SQL catalog stores your column UPDATE privileges for a table in one of the following ways:

- Single marker that indicates you can update all the columns in the table
- List of the individual columns you can update

The method of storage depends on the form of the UPDATE clause used to assign your privileges. The first method results from a GRANT statement UPDATE clause that does not specify an individual column list. By default, all columns in the table are included in the privilege. The second method results from a GRANT statement UPDATE clause that specifies an individual column list.

The storage method is significant, because subsequent changes to the table's columns or privileges can produce different outcomes depending on the initial storage method.

Column UPDATE examples

This section has a series of examples showing how the effects on a user's table column privileges of subsequent ALTER TABLE, REVOKE, and GRANT statements can depend on the format of the UPDATE clause of the initial GRANT statement.

Effect of ADD and DROP in ALTER TABLE statement

Assuming table TABLEZ has columns COL_A, COL_B, and COL_C and proper grant authorization, consider the following GRANT statements, both of which grant UPDATE privileges to each of the columns in TABLEZ:

```
GRANT UPDATE ON TABLEZ TO JUAN
```

```
GRANT UPDATE (COL_A, COL_B, COL_C) ON TABLEZ TO MARIA
```

The Model 204 SQL Server stores this information approximately as follows, where * means all columns, and GRANT OPTION, which refers to the WITH GRANT OPTION of GRANT UPDATE, is N (no) unless specified in the GRANT statement:

USER	PRIVILEGE	COLUMNS	GRANT OPTION
JUAN	UPDATE	*	N

USER	PRIVILEGE	COLUMNS	GRANT OPTION
MARIA	UPDATE	COL_A	N
		COL_B	N
		COL_C	N

Now, note the effect of the following statement on Juan and Maria's UPDATE privileges:

```
ALTER TABLE TABLEZ ADD COL_D
```

Juan's stored UPDATE privilege information, though physically unchanged, now includes the ability to update the newly added COL_D. However, Maria's unchanged privileges do not include the ability to update COL_D.

Continuing, note the effect of the following statements on Juan and Maria's UPDATE privileges:

```
ALTER TABLE TABLEZ DROP COL_A
```

```
ALTER TABLE TABLEZ ADD COL_A
```

Juan's stored UPDATE privilege information remains physically unaffected, and he can still update all the columns in TABLEZ: COL_A, COL_B, COL_C, and COL_D. However, Maria's UPDATE privilege for COL_A gets dropped when COL_A is dropped from TABLEZ (see "Using DROP column" on page 78 for the additional actions propagated when a column is dropped). If COL_A is added back to the table, Maria's UPDATE privileges do not change and she can update only COL_B and COL_C.

Effect of REVOKE

Continuing the conditions of the previous example, note the effect of the following REVOKE statement. Remember, Juan can update all columns in TABLEZ, namely, COL_A, COL_B, COL_C, and COL_D.

```
REVOKE UPDATE ( COL_C ) ON TABLEZ TO JUAN
```

The Model 204 SQL Server *must* change the storage format and store Juan's privilege information in list format:

USER	PRIVILEGE	COLUMNS	GRANT OPTION
JUAN	UPDATE	COL_A	N
		COL_B	N
		COL_D	N

If the REVOKE statement had preceded the ALTER TABLE statements in the example, Juan's UPDATE privileges would have been affected like Maria's were. That is, instead of automatically expanding and contracting with changes to the table, they would apply only to the explicitly named original columns, and they could be dropped if a table column were dropped.

Effect of WITH GRANT OPTION

Return to the example situation for Juan after the first GRANT statements and before the ALTER TABLE statements. Juan's UPDATE privileges are stored as follows:

USER	PRIVILEGE	COLUMNS	GRANT OPTION
JUAN	UPDATE	*	N

Note the effect of the following statements on Juan's UPDATE privileges:

```
GRANT UPDATE (COL_A) ON TABLEZ TO JUAN
```

```
GRANT UPDATE (COL_A) ON TABLEZ TO JUAN WITH GRANT OPTION
```

The first GRANT statement has no effect on Juan's privileges, which already include the privilege to update COL_A. But the WITH GRANT OPTION of the second GRANT statement introduces information that Juan's stored privileges do not include. Consequently, the Model 204 SQL Server stores Juan's privilege information in list format as follows:

USER	PRIVILEGE	COLUMNS	GRANT OPTION
JUAN	UPDATE	COL_A	Y
		COL_B	N
		COL_C	N

DDL statement-level security

Table 4-5 displays the privileges required for execution of the individual Model 204 SQL DDL statements. These privilege requirements are checked when you submit your DDL to the CVI utility.

System manager privileges are determined by the Model 204 login user ID.

This section also contains a discussion of view privileges and an example showing the application of statement security rules, especially for CREATE VIEW and GRANT.

Note: You can replace Model 204 SQL statement security with privilege checking by an external security package. You provide user exits to the security package in a Model 204-defined format, as described in the *Model 204 Security Interfaces Manual*. The SQL Server passes to the user exit all the information necessary to perform privilege checking identical to the Model 204 SQL privilege checking. The extent of the checking done is an option of the user exit.

Table 4-5. DDL statement security

To issue...	You must have...	Comments
CREATE SCHEMA	Model 204 system manager privileges AND The schema authorization ID must match the Model 204 login ID of the user issuing the CREATE SCHEMA statement	Ensure the match between login ID and authorization ID by issuing SET USER before CREATE SCHEMA. If no authorization ID is specified in the CREATE, the login ID is used.

Table 4-5. DDL statement security (Continued)

To issue...	You must have...	Comments
CREATE TABLE	<p>Model 204 system manager privileges AND A Model 204 login ID that matches the authorization ID for the schema containing the table</p>	<p>Ensure the match between login ID and authorization ID by using SET USER.</p> <p>If CREATE TABLE is part of a CREATE SCHEMA transaction, you may not create a table that is qualified by a schema name other than the schema that is the current default.</p> <p>The table owner (containing schema's authorization ID) automatically gets all privileges (including WITH GRANT OPTION) for the table.</p> <p>The Model 204 file to which the table maps is <i>not</i> opened during the processing of the CREATE TABLE DDL.</p>
CREATE VIEW	<p>A Model 204 login ID that matches the authorization ID for the view's schema <i>and</i> an authorization ID that matches the authorization ID for every object referenced in the view (that is, you own the schema and all the objects referenced in the view)</p> <p>OR</p> <p>A Model 204 login ID that matches the authorization ID for the view's schema <i>and</i> the authorization ID for the view's schema has at least SELECT privileges on every object referenced in the view</p>	<p>You may not create a view that is qualified by a schema name other than the schema that is the current default.</p> <p>The view owner (containing schema's authorization ID) automatically gets view privileges that match the level of privileges for the objects referenced in the view.</p> <p>If the view is not logically updateable, the level of privileges the view owner automatically gets may not exceed SELECT privileges.</p>
DROP SCHEMA	<p>Model 204 system manager privileges AND A Model 204 login ID that matches the authorization ID for the schema</p>	<p>Ensure the match between login ID and authorization ID by issuing SET USER before DROP SCHEMA.</p>
DROP TABLE	<p>Model 204 system manager privileges AND A Model 204 login ID that matches the authorization ID for the schema containing the table</p>	<p>The authorization ID of the schema that contains the table must match the Model 204 login ID of the user issuing the DROP TABLE statement.</p> <p>Ensure the match between login ID and authorization ID by issuing SET USER before DROP TABLE.</p>
DROP VIEW	A Model 204 login ID that matches the authorization ID for the schema containing the view	

Table 4-5. DDL statement security (Continued)

To issue...	You must have...	Comments
ALTER TABLE	Model 204 system manager privileges OR A Model 204 login ID that matches the authorization ID for the schema containing the table	Ensure the match between login ID and authorization ID by issuing SET USER before ALTER TABLE.
GRANT (for a table)	A Model 204 login ID that matches the authorization ID for the schema containing the table on which the privilege is being granted OR By the WITH GRANT OPTION clause of some previous GRANT statement, that specifically granted the right to GRANT privileges of the named type for this table	You may GRANT privileges for a table that is qualified by a schema name other than the schema that is the current default.
GRANT (for a view)	A Model 204 login ID that matches the authorization ID for the schema containing the view on which the privilege is being granted, <i>and also</i> the authorization ID for the view's schema must have the corresponding privilege with WITH GRANT OPTION on all tables and views referenced in any part of the view definition OR By the WITH GRANT OPTION clause of some previous GRANT statement, that specifically granted the right to GRANT privileges of the named type for this view	You may GRANT privileges for a view that is qualified by a schema name other than the schema that is the current default.
REVOKE	A Model 204 login ID that matches the authorization ID for the schema containing the object on which the privilege is being revoked OR By the WITH GRANT OPTION clause of some previous GRANT statement, that specifically granted the right to GRANT privileges of the named type for this object	REVOKE only deletes privileges from the object named in the REVOKE statement. The revoking action does not cascade to views that reference the object.
SET SCHEMA	Any Model 204 SQL user	The DDL statements you can issue within this schema context are determined at the specific statement level.

Table 4-5. DDL statement security (Continued)

To issue...	You must have...	Comments
SET USER	Model 204 system manager privileges	Sets the current SQL authorization ID for the duration of the user session or until the next SET USER.

Model 204 SQL view privileges

View creation privileges are designed to uphold the security definition of the base tables and columns referenced in the view. These base objects require protection because privileges to use a view are checked when you issue a DML statement against the view, while the privileges for objects referenced by the view are checked when DDL is processed. Model 204 SQL DDL statement security, therefore, prevents you from using a view to circumvent the privileges for the base objects referenced by the view.

Three of the fundamental principles upheld by Model 204 SQL statement security are:

- You can only create views that satisfy the following rule: the authorization ID for the view's schema must have at least SELECT privileges on every object referenced in any part of the view definition. This rule holds whether you are a system manager or the owner of the view.
- You can neither create nor grant privileges for a view that makes the view base objects more accessible than your privileges for the base objects allow. For example, you cannot grant update privileges for a view if you have only the SELECT privilege on the objects referenced by the view.
- You can update a base object through a view only if you are the owner of the object or if you are granted the privilege to update from the owner of the object.

Since only the privilege records *for the view* and not for the base objects referenced by the view are checked when a DML request is processed, a view definition stands on its own. The view definition does not automatically reflect any changes to its base objects that occur after the view is created. If a base object referenced in a view is deleted, the view definition is not changed. You are responsible for the integrity over time of your view definitions.

SQL statement security example

This section contains an example of a sequence of SQL DDL transactions and CVI utility messages. Following the example are explanatory comments.

The example shows the effect of some of the basic rules governing DDL statement security, especially for view definition and the granting of view privileges. The example uses the demonstration database files (CLIENTS, VEHICLES and CLAIMS03).

In the example, an INSURANCE schema is created containing:

- Three base tables
- Three views, one per base table
- Three schemas, corresponding to agents
- Privileges for each object

The following SQL objects, contained in the agent schemas, are defined by the schema owners:

- Views against the base tables
- Views against other views
- Privileges for each object

MISMAN is the only system manager. MISDEPT and BOB are common users. JOHN and MARY are agents. Transactions are followed by messages with return codes indicating the validity of the transaction. Boldface numbers to the right of the example statements are used in the comments following the example.

Example

User MISDEPT logs in (not a system manager):

```
1 CREATE SCHEMA INSURANCE AUTHORIZATION MISDEPT)
      SQL Error -551: User MISDEPT does
      not have the privilege to perform
      operation CREATE on object SCHEMA.
```

System manager logs in (user MISMAN):

```
2 SET USER MISDEPT
      Completion Code: 0

3 CREATE SCHEMA INSURANCE AUTHORIZATION MISDEPT
  CREATE TABLE CLIENTS ...
  GRANT SELECT ON CLIENTS TO BOB WITH GRANT OPTION
  CREATE TABLE VEHICLES ...
  GRANT SELECT ON VEHICLES TO BOB
  CREATE TABLE CLAIMS03 ...
  CREATE VIEW CLIENTS_V AS SELECT * FROM CLIENTS
    WHERE AGENT = USER
  GRANT ALL PRIVILEGES ON CLIENTS_V TO PUBLIC WITH GRANT
    OPTION
  CREATE VIEW VEHICLES_V AS SELECT * FROM VEHICLES
    WHERE AGENT = USER
```



```

GRANT ALL PRIVILEGES ON VEHICLES_V TO JOHN WITH GRANT
OPTION
CREATE VIEW CLAIMS03_V AS SELECT * FROM CLAIMS03
WHERE AGENT = USER
GRANT ALL PRIVILEGES ON CLAIMS03_V TO MARY

```

Completion Code: 0

4 SET USER BOB

Completion Code: 0

5 CREATE SCHEMA AUTHORIZATION BOB

Completion Code: 0

6 SET USER JOHN

Completion Code: 0

7 CREATE SCHEMA AGENT_J16 AUTHORIZATION JOHN

Completion Code: 0

8 SET USER MARY

Completion Code: 0

9 CREATE SCHEMA AGENT_M05 AUTHORIZATION MARY

Completion Code: 0

User BOB logs in (not a system manager):

10 SET SCHEMA BOB

Completion Code: 0

```

11 CREATE VIEW YOUNG_DRIVERS AS SELECT *
FROM INSURANCE.CLIENTS WHERE DATE_OF_BIRTH > 671231

```

Completion Code: 0

12 GRANT SELECT ON YOUNG_DRIVERS TO PUBLIC

Completion Code: 0

```

13 CREATE VIEW STOLEN_CARS AS SELECT *
FROM INSURANCE.VEHICLES WHERE INCIDENT = 'ST'

```

Completion Code: 0

14 GRANT SELECT ON STOLEN_CARS TO PUBLIC

SQL Error -551: User BOB does not have the privilege to perform operation GRANT on object STOLEN_CARS.

15 CREATE VIEW COLLISIONS AS SELECT *
FROM INSURANCE.CLAIMS03 WHERE CLAIM_TYPE = 'C'

SQL Error -551: BOB does not have the privilege to perform operation SELECT on object CLAIMS03.

User JOHN logs in (not a system manager):

16 SET SCHEMA AGENT_J16

Completion Code: 0

17 CREATE VIEW MYVEHICLES AS SELECT *
FROM INSURANCE.VEHICLES

SQL Error -551: JOHN does not have the privilege to perform operation SELECT on object VEHICLES.

18 CREATE VIEW MYVEHICLES AS SELECT *
FROM INSURANCE.VEHICLES_V

Completion Code: 0

19 GRANT SELECT ON MYVEHICLES TO MARY

Completion Code: 0

User MARY logs in (not a system manager):

20 SET SCHEMA AGENT_M05

Completion Code: 0

21 CREATE VIEW MYCLAIMS03 AS SELECT * FROM INSURANCE.CLAIMS03

SQL Error -551: MARY does not have the privilege to perform operation SELECT on object CLAIMS03.

22 CREATE VIEW MYCLAIMS03 AS SELECT *
FROM INSURANCE.CLAIMS03_V

Completion Code: 0

23 GRANT SELECT ON AGENT_J16.MYVEHICLES TO JOHN

SQL Error -551: MARY does not have
the privilege to perform operation
GRANT on object MYVEHICLES.

24 CREATE VIEW MYCLIENTS AS SELECT * FROM INSURANCE.CLIENTS_V

Completion Code: 0

System manager logs in (user MISMAN):

25 SET USER MISDEPT

Completion Code: 0

26 DROP SCHEMA INSURANCE

Completion Code: 0

27 SET USER BOB

Completion Code: 0

28 DROP SCHEMA BOB

Completion Code: 0

29 SET USER JOHN

Completion Code: 0

30 DROP SCHEMA AGENT_J16

Completion Code: 0

31 SET USER MARY

Completion Code: 0

32 DROP SCHEMA AGENT_M05

Completion Code: 0

Statement security example comments

- Statement 1 fails because MISDEPT is not a system manager.
- Statement 2 allows the system manager, in effect, to log in as user MISDEPT. This is necessary so the system manager can define the INSURANCE schema to have owner MISDEPT. SQL objects can be created only by their owners. That is, the login ID of the user issuing an SQL CREATE must match the specified or implied authorization ID of the current schema.

- Statement 3 makes MISDEPT the owner of the INSURANCE schema and all the objects in the schema. Note that although system manager MISMAN issued the CREATE SCHEMA for INSURANCE, the privileges to access INSURANCE are automatically granted only to the object's owner MISDEPT. MISMAN cannot access INSURANCE unless the system manager explicitly includes a GRANT of privileges to MISMAN for INSURANCE.
- Statements 4, 6, and 8 allow the system manager to create schemas owned by the specified users. Later in the example, the users define the SQL objects that belong to their schemas.
- Unlike statements 7 and 9, statement 5 does not include a schema name value, so the default name is the authorization ID, BOB.
- Statement 10 is necessary to change the schema context (from MARY to BOB) so that BOB can add objects to his schema. Statements 16 and 20 are similar.
- Statements 11 and 13 are valid, because view owner BOB was granted (by MISMAN) the SELECT privilege for the tables referenced in his views. BOB cannot create a view that references the CLAIMS03 table, for example, because he does not have any privileges for that table (see statement 15).

Statements 11 and 13 also show the use of the schema name, INSURANCE, as a qualifier for the CLIENTS and VEHICLES tables. Without the qualifier, CLIENTS is assumed to belong to the default schema, BOB, and the statement fails.

- Statement 12 is valid, since BOB was granted the SELECT privilege WITH GRANT OPTION on the referenced table (CLIENTS). The SELECT privilege and WITH GRANT OPTION is implicitly granted to the YOUNG_DRIVERS view owner (BOB). That is, the maximum privileges available for the view are equal to those that the view owner has on the object referenced by the view, and these privileges are also implicitly granted to the view owner for the view.
- Statement 14 fails because the SELECT privilege on the referenced table VEHICLES was granted to BOB *not* including WITH GRANT OPTION. View STOLEN_CARS is created with only the SELECT privilege (not including WITH GRANT OPTION) implicitly granted to the view owner. STOLEN_CARS view is, therefore, viewable only by BOB. No one else (including MISMAN) can SELECT this view, and no one (including BOB) can update the view.
- Statements 15, 17, and 21 fail, because the view owners-to-be lack the SELECT privilege for objects they want to reference in these views.
- Statement 18 creates MYVEHICLES, a view of a view. *All* privileges are implicitly granted to the MYVEHICLES owner (JOHN), including WITH GRANT OPTION, because MYVEHICLES is logically updateable and

JOHN has all privileges on the referenced view VEHICLES_V. If the view were not logically updateable, the implicit owner privileges would be SELECT WITH GRANT OPTION.

- Statement 19 succeeds because JOHN's privileges for MYVEHICLES (which equal his privileges for the referenced view VEHICLES_V) include WITH GRANT OPTION.
- Statement 22 creates MYCLAIMS03, a view of a view. All privileges are implicitly granted to the MYCLAIMS03 owner (MARY), *not* including WITH GRANT OPTION, because MYCLAIMS03 is logically updateable and MARY has all privileges on the referenced view VEHICLES_V.
- Statement 23 fails because MARY's privileges for MYVEHICLES (granted in statement 19) do not include WITH GRANT OPTION.
- Statement 24 creates MYCLIENTS, a view of a view. All privileges are implicitly granted to the MYCLIENTS owner (MARY), including WITH GRANT OPTION, because MYCLIENTS is logically updateable and all privileges on the referenced view VEHICLES_V are granted to PUBLIC WITH GRANT OPTION.
- Statements 25 through 32 show the system manager logging in as various users and then deleting from the SQL catalog (with DROP) the schemas owned by each user. This method is necessary because to drop a schema you must be the schema owner and a system manager. That is, the login ID of the user issuing an SQL DROP SCHEMA must match the specified or implied authorization ID of the current schema; and the user issuing an SQL DROP SCHEMA must be a system manager.

SQL DDL processing

Each DDL statement is committed upon successful execution without regard to the SQL Auto Commit setting for the data source.

5

Creating DDL with the Table Specification Facility

In this chapter

- Overview
- Introduction to the Table Specification facility (TSF)
- Using TSF panels
- Creating or modifying a base table (Main Menu panel)
- Defining column names (Column List panel)
- Defining column attributes (Column Attributes panel)
- Completing table definitions (Completion panel)
- Defining multicolumn unique keys (Multi-Column Unique panel)
- Specifying GRANT authority (Grant Authority panel)
- Viewing DDL at the terminal (Completion panel)
- Generating DDL to an output file (Completion panel)

Overview

The Model 204 SQL Table Specification facility (TSF) looks at an existing Model 204 file, reads its field names and their attributes, and allows you to specify how to map the file to an SQL table. The TSF provides an interactive, menu-driven facility that generates a subset of

DDL statements based on your specifications. You can use the generated DDL as input to the CVI utility to define your SQL table to the SQL catalog.

You use the TSF to create new tables, not to modify existing ones.

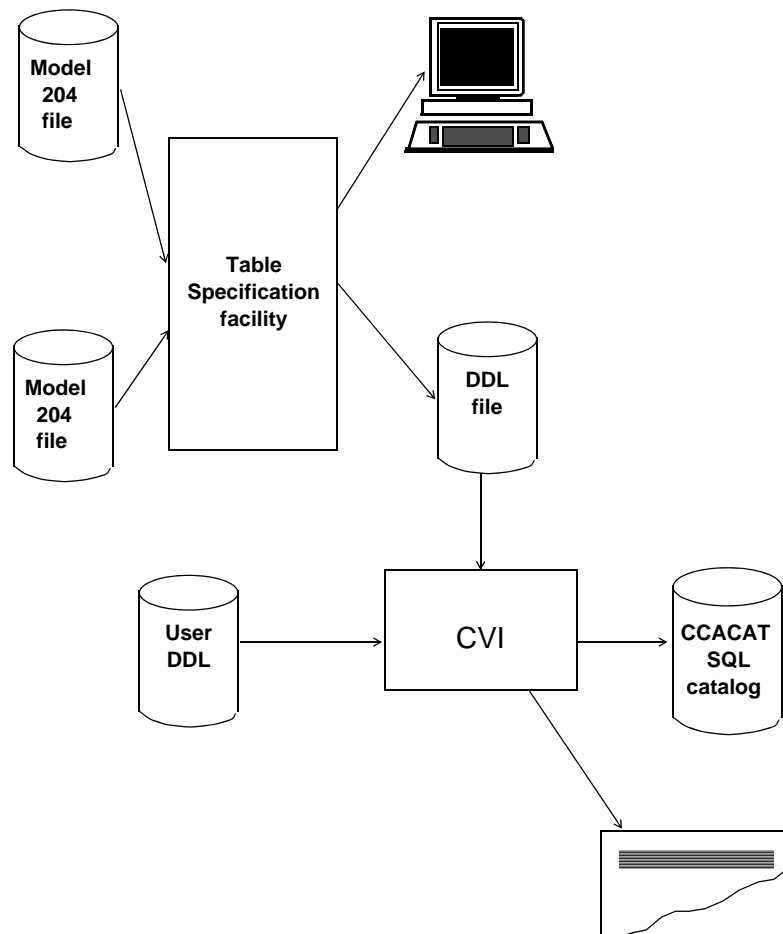
The TSF is the Model 204 subsystem CCATSF.

Introduction to the Table Specification facility (TSF)

DDL processing

Figure 5-1 shows the relationship between the Model 204 files, the TSF, manually created DDL, the CVI utility, and the SQL catalog.

Figure 5-1. DDL processing overview



TSF processing sequence

Table 5-1 summarizes the general processing that goes on within the Table Specification facility.

Table 5-1. TSF processing

Stage	Description
1	You identify the name of the SQL table you are defining and the Model 204 file to which the SQL table maps.
2	Table Specification facility reads Table A, the in-file dictionary, of the Model 204 file and displays the existing Model 204 fields.
3	You identify fields that are to be included in the SQL table and specify SQL column names.
4	For each column, you specify the SQL attributes.
5	If necessary, you specify multi-column unique indices via special panels.
6	If necessary, you specify GRANT options via special panels.
7	Table Specification facility generates SQL DDL and writes it to an external sequential file.

DDL statements generated by the TSF

Note: Currently the CCATSF subsystem does not generate DDL statement delimiters, semicolons by default, which are required by the CVI utility. You can add delimiters by hand or use another SQL utility that does not require DDL delimiters. The Connect★ ODBC unsupported utilities, DDLWIN and CLIIVP, are examples of utilities which expect DDL input lines without delimiters.

The TSF is designed for the initial definition of SQL catalog objects. It generates the following DDL statements:

```
CREATE SCHEMA
CREATE TABLE
GRANT
```

Of course, you can also manually generate the DDL statements that the TSF produces. You must manually prepare catalog operations that require the following DDL statements:

```
ALTER TABLE      DROP VIEW
CREATE VIEW        GRANT (for view privileges and for existing tables)
DROP SCHEMA        REVOKE
DROP TABLE        SET SCHEMA
```

Besides convenience, the advantage of using the TSF is that it returns the current state of the Model 204 file data. Relying on manual creation of DDL for the catalog leaves you without this data consistency safeguard. Remember,

changes to a cataloged Model 204 file are not automatically reflected in the SQL catalog. The person creating DDL manually is responsible for being aware of the current definition of the Model 204 file.

Model 204 SQL DDL extensions generated by the TSF

The TSF-generated DDL includes the following Model 204 SQL extensions to standard SQL. These extensions are described where appropriate in the individual panel discussions in this chapter. For more information about these extensions, see Chapter 4.

Model 204 extension	Provides...
<code>SYSNAME</code> <i>filename</i> or <i>fieldname</i>	<p>Aliasing of both table and column names. The <code>SYSNAME</code> value is the actual name of the Model 204 file or field; it does not have to comply with SQL table and column naming rules.</p> <p>The <code>SYSNAME</code> clause is required only if the SQL table or column name (after any modification to meet Model 204 naming conventions) differs from the Model 204 file or field name. The TSF automatically applies a name-correcting algorithm to your SQL table and column names to determine whether a <code>SYSNAME</code> clause is needed.</p>
<code>SYSTEM</code> qualifier for <code>PRIMARY KEY</code>	<p>System-generated primary key in cases where no suitable column is available or where two or more columns are used as a composite primary key, for example, to link a nested table to its parent table.</p> <p>In the TSF, the system-generating action of <code>SYSTEM</code> is available as a default on the Column List panel (TSF2). TSF has no explicit prompt for <code>SYSTEM</code>.</p>
<code>NESTED USING</code> clause	<p>Definition for the table as a nested table and indicates the table column (foreign key) that links the nested table to its parent table. This extension permits the mapping of Model 204 multiply occurring fields to a set of columns in a nested table.</p> <p>If you designate a table as nested on the TSF Main Menu, you must also specify the nested table linking or foreign key.</p>
<code>REFERENCES</code> <i>parent table name</i>	<p>Qualifier (and constraints) for the foreign key column of a nested table.</p> <p>The Model 204 SQL <code>REFERENCES</code> clause functionality is nonstandard in that it is required for a nested table and cannot be used in any other context.</p> <p>The TSF has no explicit prompt for the <code>REFERENCES</code> clause. If you designate a nested table Primary Key on the TSF Main Menu, the generated DDL for the nested table foreign key includes a <code>REFERENCES</code> clause in the <code>CREATE TABLE</code> statement.</p>

SQL and Model 204 data consistency

When you use the TSF, it displays the current Model 204 file definitions. With these you specify SQL DDL mappings of the files and populate the SQL catalog. If no changes were made to the file before you submitted this DDL to the SQL catalog, you can be sure that the catalog and the Model 204 file are consistent.

If you edit the TSF-generated DDL before using it to populate the SQL catalog, you are responsible for ensuring that your edits are based on the current Model 204 file definition.

Using TSF panels

Figure 5-2 on page 102 illustrates the order in which you would likely use the TSF panels to define DDL. Each of these panels is discussed separately in the following sections of this chapter.

You do not need to complete the entire table definition process in one session. You can complete part of the process, log off, and return later to the TSF subsystem and pick up where you left off.

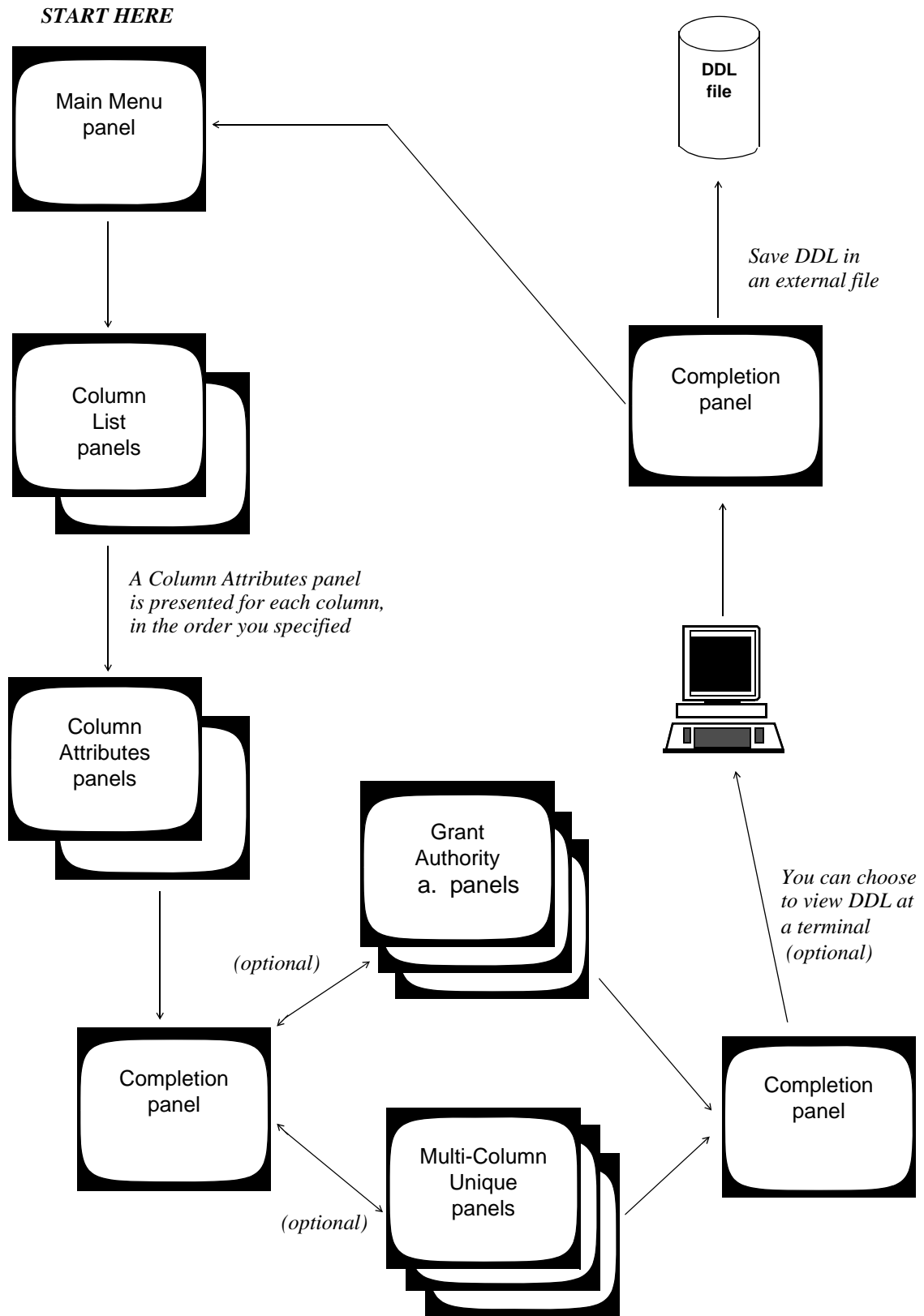
The Model 204 files that you name during a TSF session are opened by TSF processing. Make sure these files are available to the Model 204 Online run.

Panel conventions

The Table Specification facility has the following panel conventions:

- Panel number (for example, TSF1) appears in the upper left corner of the panel. The panel title (for example, Main Menu) appears in the center of the top line of the panel. The version number appears in the upper right corner of the panel.
- Error messages appear at the bottom of the panel. Input areas that are in error are tagged with an asterisk (*) and highlighted.
- Command line (==>) is near the bottom of the panel. Use the command line to enter commands in lieu of PF keys. The minimum abbreviation for a command is displayed in capital letters and is generally the first three characters (for example, DEL for DELeTe).

Figure 5-2. TSF panel map



ENTER and PF key conventions

At the bottom of each TSF panel is a list of the PF keys that you can use on the panel. Table 5-2 displays the TSF key conventions.

Table 5-2. Table Specification facility Enter and function keys

Key	Performs this function...
Enter	On menus, processes your selections; on update panels, edits the panel, but does not store any updates
PF1	Accesses online help
PF2	Refreshes the screen without processing any updates
PF3	Leaves the current panel or application (if you are on the Main Menu) without storing any updates
PF4	Moves screen left
PF5	Moves screen right, or stores definition and provides fresh panel (on Grant Authority or Multi-Column Unique panels)
PF7	Scrolls back to the previous screen of data without storing updates, or displays the previous data definition (on the Grant Authority or Multi-Column Unique panels) after storing updates
PF8	Scrolls forward to the next screen of data without storing updates, or displays the next data definition (on the Grant Authority or Multi-Column Unique panels) after storing updates
PF9	Deletes the current Model 204 file definition
PF10	Takes you to the Completion panel
PF11	Takes you to the Column Attributes panel
PF12	Leaves the current panel or application (if you are on the Main Menu) and commits all updates

Logging in

Before you can log in to the TSF, the CCATSF subsystem must already have been started (with the Model 204 START SUBSYSTEM command).

To log in to TSF, at your Model 204 Online prompt, type:

CCATSF

and press Enter.

The Table Specification facility Main Menu appears.

Creating or modifying a base table (Main Menu panel)

After you log in to TSF, the Table Specification facility Main Menu panel (TSF1) is displayed.

This section includes a description of each of the panel fields and nonstandard PF keys of the Main Menu, general usage notes for the panel, and information about how to use the TSF to create nested tables.

The descriptions of the individual panel fields include directions for how to use the panel, such as whether you immediately press Enter to record your entry or whether you fill in other fields before you press Enter.

Figure 5-3 shows an example of the Main Menu with user entries in four of the fields. These entries initiate the definition of Mark's CLIENTS table, which is mapped to the Model 204 demonstration database CLIENTS file. Examples in subsequent sections follow this mapping to completion.

Figure 5-3. Main Menu panel

TSF1	MODEL 204 Table Specification Facility (TSF)	7.1.0
Main Menu		
Schema Authorization:	MARK_____	
SQL Table Name:	CLIENTS_____	
Schema Name (optional):	_____	
MODEL 204 File Name:	_____	
File Password:	_____	
SQL Table Type:	B (B=base P=parent N=nested)	
Primary Key (if TYPE=P or N):	_____	
Parent Table (if TYPE=N):	_____	
Identify SQL table to be defined (AUTHORIZATION & NAME)		
==>		
1=HElP 2=REFresh 3=QUIt		

Press Enter, then fill in SCHEMA and Model 204 File name.

Creating SQL objects in the context of a schema

The TSF generates a stream of SQL DDL statements for creating or modifying SQL tables and columns. Each such stream includes a CREATE SCHEMA statement naming the schema to which the DDL applies.

The schema name must be unique within the SQL catalog, and issuing CREATE SCHEMA for a schema that already exists is an error.

Because the TSF does not validate each schema name against previous schemas, you must review the DDL generated by the TSF and determine whether to keep the schema name, delete it, or change it (using a SET SCHEMA statement).

Schema Authorization

At the Schema Authorization prompt, enter an authorization ID. The default authorization ID is the Model 204 login user ID. You must enter a value at this prompt, and, because the TSF does not validate authorization IDs, you must ensure that the value is a valid Model 204 login ID.

The authorization ID you enter becomes the owner of the tables you create. The authorization is generated in the CREATE SCHEMA DDL that the TSF produces for this table.

The authorization ID must follow the naming rules for a Model 204 login ID (no more than 10 characters and no underscore characters, must begin with an alphabetic character and must not contain certain character combinations). The authorization ID cannot be an SQL reserved word (see Appendix B).

To see a list of pending authorization IDs, IDs from table definitions you have not deleted from the TSF, place your cursor in the input area for this field and press PF1.

SQL security is based on authorization IDs and granted privileges.

SQL Table Name

At the SQL Table Name prompt, enter the name of the SQL table that you are defining. Multiple nested tables are allowed. You must enter a value at this prompt.

The table name can contain up to 18 characters (A–Z, 0–9, and underscore). No embedded blanks are allowed. Table names must begin with an alphabetic character.

The TSF applies certain truncation and compression rules (described further on “Mapping table names to file names” on page 53) to the SQL table name you specify and compares the resulting name to the corresponding Model 204 file name. If the resulting name does not match the Model 204 file name, the TSF automatically adds a SYSNAME clause with the Model 204 file name to the TSF-generated DDL.

To see a list of pending tables, names from table definitions you have not deleted from the TSF, place your cursor in the input area for this field and press PF1.

At this point, the other input areas are protected (unavailable for input). After inputting the authorization and table name, press Enter.

If this is a new table definition, the TSF displays a message to tell you that this is a new definition. If a table definition exists, the TSF displays a message telling you the definition is pending, and enables PF9, PF10, and PF11. To delete the existing table definition, press PF9. To proceed to the Completion panel (TSF4), press PF10. To proceed to the Column Attributes panel (TSF3), press PF11.

At this point, the other input areas are unprotected and the authorization and table name input areas are protected.

Schema Name

At the Schema Name prompt, enter the SQL schema name as you want it to appear in the CREATE SCHEMA statement. If you do not specify a schema name, Schema Name defaults to the authorization ID specified for Schema Authorization.

Model 204 File Name and Password

At the Model 204 File Name prompt, enter the name of the Model 204 file that you want defined as an SQL table. An entry for Model 204 File Name is required.

If applicable, include the file password. The password you enter must give you the authority to read the file and all the field names for which you want to set up columns in the SQL table you will create.

Note: You cannot use Model 204 file groups, although you can use individual files that may belong to a file group. Also, the Model 204 file to be used as an SQL table must be defined as a transaction backout (TBO) file. For more information about TBO files, see the *Model 204 File Manager's Guide*.

Table Type

At the Table Type prompt, enter the appropriate table type: B (Base), P (Parent), or N (Nested). The Table Type value is required.

Model 204 SQL base tables are SQL schema tables that map directly to Model 204 files and fields. By contrast, views are schema tables that map directly to SQL base tables or to other views. Base tables are created by CREATE TABLE; views are created by CREATE VIEW.

A nested table is a base table with columns that map to Model 204 multiply occurring fields or groups of fields. A nested table is associated with a single base table parent by a unique table column key which joins the nested table to the parent table.

If you designate a table as nested, the TSF-generated DDL for the table includes the NESTED USING clause in the CREATE TABLE statement.

For more information about creating nested tables with the TSF, see “Defining nested tables” on page 108.

Primary Key

At the Primary Key prompt, enter the name of the primary key if the table type is P (Parent) or N (Nested). The primary key is required and valid only for these table types.

Note: To use the TSF to define a primary key in a B (Base) table, define the table as a P (Parent) with no associated nested table.

This field identifies the SQL column name of the primary key in a parent or nested table. (By definition, in Model 204 SQL DDL the nested table primary key is also a *foreign key*: the values of the nested table primary key must match the values of the parent table primary key.) Later (on the Column List panel) you are asked to map this column name to an existing Model 204 field. If you do not provide a Model 204 field at that time, by default TSF treats this as a system-generated key.

The primary key must be mapped to a field that has unique values. The corresponding Model 204 field must have the UNIQUE (and ORDERED) attribute.

Do not enter the name of a column that maps to a Model 204 INVISIBLE field.

For more information about creating nested tables with the TSF, see “Defining nested tables” on page 108.

Parent Table

At the Parent Table prompt, enter the name of the SQL table that each nested table references. The parent table name is required for nested tables (Table Type N).

Keeping or deleting the pending definition

The definition of the actual Model 204 database file that you are defining is copied into a work record within the CCATSF subsystem only *once*, when the specification process begins. This work record represents a frozen “snapshot” picture of the Model 204 definition at the time you start the specification process.

If you are modifying an existing definition, the CCATSF subsystem does *not* read the Model 204 database file nor look at its most recent status; it instead does all its processing from the work record already built. If significant changes have been made to the definition of the Model 204 database file, then you might want to delete the “pending” definition (using PF9) and begin again.

If you continue with the pending definition, you can use PF11 to go to the Column Attributes panel (TSF3) where you can define SQL attributes for

columns already selected. Or you can use PF10 to go to the Completion panel (TSF4) to build GRANT statements and multicolumn UNIQUE keys and to generate DDL for the table.

Defining nested tables

If you are defining a nested table with the TSF, follow these guidelines:

- Map each column in a nested table (except the foreign key) to a Model 204 field that can multiply occur (that is, does not have OCCURS 1 or AT-MOST-ONE field attribute).
- Make sure that the field mapped to the parent table primary key is defined with the Model 204 field attributes UNIQUE and ORDERED.
- Be sure that each column in the nested table (except the foreign key) occurs the same number of times on the Model 204 record to which the occurrence group maps.
- Go through the TSF panels multiple times: once for the parent table, and once for each nested table. Whether you define the parent table before or after the nested table(s) does not matter.
- Specify a Primary Key value for parent and for nested tables. The nested table primary key is really a foreign key, and its data values, although not its name, must match the data values of the parent table primary key.
- Specify a Parent Table value for each nested table.
- Do not specify a multicolumn unique key for a nested table.

TSF rules

The TSF automatically enforces the following rules by limiting your panel choices:

- Columns in each nested table must be NOT NULL.
- Column in a nested table (including the foreign key) must not be mapped to a field with the Model 204 field attribute INVISIBLE.

For more information about using DDL to define nested tables, see “Creating nested tables” on page 62. Using DML to access nested tables is discussed in “Using SQL DML against nested tables” on page 168.

Defining column names (Column List panel)

After you specify the Model 204 file and the SQL table name and press ENTER from the Main Menu, the Column List panel appears.

Figure 5-4, “Column List, second panel” on page 110, and “Column List, third panel” on page 111, show a Column List panel with user column selections filled in.

Figure 5-4. Column List panel

TSF2			TSF Table Specification - Column List		7.1.0
Table: CLIENTS			MODEL 204 File: CLIENTS	Field 1 of 26	
Type : BASE					
CMD	Column Name (or "=")	MODEL 204 Field Name			
---	-----	-----			
-	_____	ADDRESS			
-	_____	AGENT			
-	_____	ANNIV DATE			
-	=_____	CITY			
-	BIRTHDAY_____	DATE OF BIRTH			
-	DRIVERID_____	DRIVER ID			
-	_____	DUMMY			
-	_____	FIELDA			
-	SORTKEY_____	FULLNAME			
-	_____	INCIDENT			
M=Move B=Before A=After					
WARNING: File contains invisible fields which are displayed as bright.					
At page one.					
===>					
<ENTER>=Validate					
1=HELp		3=QUIt	4=LEFt	5=RIght	
7=BACKward	8=FORward		10=FINAl	11=ATTribute	12=END

Figure 5-5. Column List, second panel

TSF2		TSF Table Specification - Column List		7.1.0
Table: CLIENTS		MODEL 204 File: CLIENTS		Field 11 of 26
Type : BASE				
CMD	Column Name (or "=")	MODEL 204 Field Name		
---	-----	-----		
-	_____	INCIDENT DATE		
-	_____	LASTNAME		
-	_____	MARITAL STATUS		
-	_____	NAME SOUND		
-	POLNO _____	POLICY NO		
-	_____	POLICYHOLDER		
-	= _____	RECTYPE		
-	_____	RESTRICT.INDEX		
-	_____	RESTRICTIONS		
-	_____	SETKEY		
M=Move B=Before A=After				
WARNING: File contains invisible fields which are displayed as bright.				
===>		<ENTER>=Validate		
1=HELp		3=QUIT	4=LEFt	5=RIGHT
7=BACKward		8=FORward	10=FINAl	11=ATTribute 12=END

Figure 5-6. Column List, third panel

```

TSF2                      TSF Table Specification - Column List                      7.1.0
Table: CLIENTS                      MODEL 204 File: CLIENTS                      Field 21
of 26
Type : BASE
CMD  Column Name (or "=")          MODEL 204 Field Name
---  -----
-    =_____                     SEX
-    _____                     SSN
-    =_____                     STATE
-    _____                     TOTAL PREMIUM
-    _____                     VIN
-    _____                     ZIP

M=Move      B=Before      A=After
WARNING: File contains invisible fields which are displayed as
bright.
On last page.
===>                                     <ENTER>=Validate
1=HElP           3=QUIt           4=LEfT           5=RIgHt
7=BACKward      8=FORward        10=FINAl          11=ATTribute 12=END

```

Model 204 field names

The Column List panel displays the SQL table and Model 204 file name that you entered on the Main Menu. The message at the upper right-hand corner (for example, Field 1 of 21) denotes where you are in the list of Model 204 field names.

The field names for the selected Model 204 file appear in the middle of the panel. As described in “Defining SQL column names” on page 112, you select the Model 204 fields you want defined as SQL columns by entering a column name next to the associated field name.

After you specify the column names, you can press PF11 to go to the Column Attributes panel to define the column attributes. Rocket Software recommends that you define the attributes of each column before using the Completion panel (TSF4) to request DDL generation. If you do not explicitly define attributes, default SQL attributes are assigned to each column based on its Model 204 field attributes.

Because the complete MODEL 204 Field Name list might cover more than one panel, use PF7 or PF8 to move panel by panel through the list. You can type BAC *n* or FOR *n* on the command line and move backward or forward the indicated number of lines on the panel. You can also type a number on the command line and press PF7 or PF8 to move the indicated number of lines on a panel.

Note as you scroll forward and backward through the panels, *no* selections are permanently captured until you use PF10, PF11, or PF12. If you attempt to quit without storing modifications, you receive a warning message and are asked to reconfirm the quit.

To remove a column name, overwrite with spaces.

Note: If a field name is longer than 45 characters, you can use PF4 and PF5 to scroll left and right to see the entire field name.

Changing the order of field names

Initially, the Model 204 field names appear in alphabetical order. Specify M, B, or A in the CMD (command) column and press Enter to change the order of column names and field names. M indicates the field to be moved. B (Before) and A (After) indicate where the marked field is to be placed.

The columns in the DDL CREATE TABLE statement that the TSF generates are listed according to the order of the columns on this panel. The exception to this rule is that a primary key that you have the TSF generate is always listed last in the DDL column definitions.

Defining SQL column names

Select the fields you want defined as SQL columns by entering a column name next to its associated field name. To simplify the data entry process, enter an equal sign (=) to make the column name the same as the Model 204 field name (embedded blanks and periods are translated to underscores and field names are truncated to 18 characters).

A valid column name contains the characters A–Z, 0–9, or underscore and has a maximum length of 18 characters. Embedded blanks are not allowed. The column name cannot be an SQL reserved word (see Appendix B).

The TSF compares the column name to the corresponding Model 204 field name. If the SQL name does not match the Model 204 field name, the TSF automatically adds a SYSNAME clause with the Model 204 field name to the TSF-generated DDL. The SYSNAME value is the actual name of the Model 204 field; it does not have to comply with SQL table and column naming rules.

For more information about the SYSNAME extension, see “Column naming and the SYSNAME extension” on page 58.

Note the following provisions:

- Model 204 INVISIBLE fields are displayed with bright highlighting. You can select these fields as columns (for nonnested tables), but their SQL use is restricted. For example, you cannot update such a column in SQL. INVISIBLE field restrictions are described in “Using Model 204 file data

features” on page 23 and on “Using SQL DML against INVISIBLE fields” on page 164.

- You cannot select Model 204 fields that have the attribute AT-MOST-ONE or OCCURS 1 (meaning that the field does not multiply occur) as SQL columns in a nested table (the exception to this rule is the primary key field which might be defined this way).
- If a file is a sorted or hash key file, you must select the sort or hash key as a column if the SQL table type is Parent or Base and the Model 204 FILEORG parameter has the X'02' option (key required) set. In this case, a default column name is automatically supplied to ensure that this field is selected. Default column names (which you can change) are SORTKEY or HASHKEY.
- You must select at least one column name.
- If you do not assign to any Model 204 field the SQL column you designated (on the Main Menu) as the primary key, the TSF automatically includes a system-generated key in the TSF output DDL. The DDL for the table has PRIMARY KEY SYSTEM.
- If you change the names of columns or the selections of the columns in a table after defining your GRANT and UNIQUE statements on the Grant Authority and Multi-Column Unique panels, you receive an error message when you attempt to generate DDL.

Defining column attributes (Column Attributes panel)

After you press PF11 on the Column List panel, one or more Column Attributes panels appear with default attributes for the columns selected on the Column List panel. On the Column Attributes panel, you define the SQL attributes you want for the selected columns.

The top of the Column Attributes panel has the SQL table name and the corresponding Model 204 file name. The listing below includes the SQL column names, the TSF-supplied default SQL attributes, and the attributes (abbreviated) of the associated Model 204 fields. See the *Model 204 File Manager's Guide* for descriptions of the Model 204 field attributes.

If the string of Model 204 field attributes ends with two periods and an angle bracket (..>), there are more attributes than can fit in the display area. Use PF4 and PF5 to scroll the panel to the left and right to view the attributes.

Figure 5-7 shows a Column Attributes panel before any user modifications.

Figure 5-7. Column Attributes panel

TSF3 TSF Table Specification - Column Attributes 7.1.0						
Table Name: CLIENTS		MODEL 204 File: CLIENTS Col. 1 of 8				
Column Name:	Nulls?	Format	Len	Prec	Scale	M204 Attributes
CITY	Y	CHARACTER__	255	__	__	STR REPT
BIRTHDAY	Y	CHARACTER__	255	__	__	STR ORD-NUM ONE
DRIVERID	Y	CHARACTER__	255	__	__	BIN COD OCC ORD-..>
SORTKEY	Y	CHARACTER__	255	__	__	STR REPT
POLNO	Y	CHARACTER__	6	__	__	STR KEY OCC LEN=..>
RECTYPE	Y	CHARACTER__	255	__	__	STR COD OCC ORD-..>
SEX	Y	CHARACTER__	255	__	__	STR ORD-CHAR REPT
STATE	Y	CHARACTER__	255	__	__	STR COD OCC ORD-..>

At page one.

====>

1=HElP 3=QUIt 4=LEFt <ENTER>=Validate

7=BAckward 8=FORward 10=FINAl 5=RIgHt 12=END

Figure 5-8 shows the Column Attributes panel after user modifications.

Figure 5-8. Modified Column Attributes panel

TSF3		TSF Table Specification - Column Attributes				7.1.0	
Table Name: CLIENTS		MODEL 204 File: CLIENTS		Col. 1 of 8			
Column Name	Nulls?	Format	Len	Prec	Scale	M204	Attributes
CITY	Y	CHARACTER	30			STR	REPT
BIRTHDAY	Y	CHARACTER	8			STR	ORD-NUM ONE
DRIVERID	Y	INTEGER				BIN	COD OCC ORD-...>
SORTKEY	Y	CHARACTER	30			STR	REPT
POLNO	N	CHARACTER	6			STR	KEY OCC LEN=...>
RECTYPE	Y	CHARACTER	12			STR	COD OCC ORD-...>
SEX	Y	CHARACTER	1			STR	ORD-CHAR REPT
STATE	Y	CHARACTER	20			STR	COD OCC ORD-...>

At page one.

====>

1=HELp 3=QUIT 4=LEFt <ENTER>=Validate

7=BACKward 8=FORward 10=FINAl 5=RIGHT 12=END

Specifying attributes

The TSF automatically generates default SQL attributes for each selected SQL column based on its Model 204 field attributes. If you make no revisions to these default SQL column attributes, the TSF assigns them by default. Otherwise, you can modify the selected values.

Note: Attributes that you supply must apply to *all* records in the Model 204 file, regardless of record type.

Nulls

You must indicate whether or not nulls are allowed for each selected column. Your choice determines whether the NOT NULL clause is built into the generated DDL for each column.

If you know that a field cannot contain nulls or that a field always has a value on a record, define a column as:

```
CHAR (n) NOT NULL
```

This results in more efficient queries and prevents generating different results in Model 204 User Language.

For example, when doing an index count, in SQL “not equal” (<>) is true only if there is a value for the field on the record that is not equal to the search string. In Model 204, if the field is not present on the record, the NOT NULL values are included in the count using the index, because Model 204 does not keep an index value NULL or NOT NULL. Once the NOT NULL count is established using the index, the Model 204 SQL processor has to check every record in that set to see whether or not there is a value for the field, and, for each record that does not have a value, reduce the index count by one. This can be very costly in terms of I/O and CPU consumption. Alternatively, if you define the column as CHAR (n) NOT NULL, Model 204 User Language simply does an indexed search to evaluate the “not equal” query.

For more information about matching Model 204 and SQL data formats, see “Matching Model 204 and SQL data formats” on page 31.

The NULL field defaults to Y (yes) except for the following cases in which it defaults to N (no): all columns in a nested table, and the primary key of a parent table.

Note: In some cases, TSF prevents you from changing the default value displayed for this field. Typically, the field value area is protected when the default is N.

Format

You must indicate the data type format of the column. Valid values are:

- DECIMAL or DEC
- NUMERIC
- INTEGER or INT
- SMALLINT
- FLOAT
- CHARACTER or CHAR
- REAL
- DOUBLE (for DOUBLE PRECISION)

Defaults, which you can change, are as follows:

SQL data type	Model 204 field attribute
CHARACTER	STRING
FLOAT	FLOAT
INTEGER	BINARY

Specify INTEGER or DECIMAL for real numeric data only where leading and trailing zeros of significance do *not* occur. If leading or trailing zeros have significance (for example, in Social Security Numbers or dates), choose CHARACTER.

Also specify CHARACTER if the data field can ever contain nonnumeric data values.

For more information about matching Model 204 and SQL data formats, see “Matching Model 204 and SQL data formats” on page 31.

Len (length)

You must specify a length if your data format is CHARACTER. Enter a length between 1 and 255. This is the length of the string displayed to SQL requests for this column. To prevent the display value from being truncated, enter a length large enough to hold the maximum value the corresponding Model 204 field is likely to contain.

The default value is the Model 204 length; if the Model 204 field is not preallocated or float, the default is 255.

For more information about Model 204 SQL data conversions, see “Optimizing Model 204 data conversion” on page 35 and “Observing data precision limits” on page 39.

Prec (precision)

If your data format is DECIMAL or NUMERIC, you must specify an integer value between 1 and 15 for decimal digit precision.

If your data format is FLOAT, you must specify an integer value between 1 and 53 for binary precision (between 1 and 21 is equivalent to REAL; between 22 and 53 is equivalent to DOUBLE).

For more information about the precision available for Model 204 SQL processing, see “Observing data precision limits” on page 39.

Scale

If your data format is DECIMAL or NUMERIC, you can optionally specify the scale. The scale must be an integer value less than or equal to the specified precision.

Usage note

The UNIQUE column attribute is automatically assigned or not assigned to a column in the TSF-generated DDL based on the Model 204 attribute ORDERED UNIQUE; therefore, it does not appear as an input option on the panel.

Nonstandard PF key functions

PF7 (BAckward)

In addition to its standard usage (scrolling backward 12 lines, which returns you to the previous page of data for the panel), you can use PF7 to scroll backward a number of lines that you specify. If you type BAC and a number on the command line, or if you type a number on the command line and press PF7, the panel is scrolled backward that number of lines.

PF8 (FORward)

In addition to its standard usage (scrolling forward 12 lines, which takes you to the next page of data for the panel), you can use PF8 to scroll forward a number of lines that you specify. If you type FOR and a number on the command line, or if you type a number on the command line and press PF8, the panel is scrolled forward that number of lines.

Completing table definitions (Completion panel)

After defining your last column on the Column Attributes panel, Press PF10 to access the Completion panel (TSF4). Or from the Column List panel (Figure 5-4 on page 109), press PF10 to access the Completion panel (TSF4).

Figure 5-9 shows an example of a Completion panel with user specifications for generating the DDL to the OUTDDL file.

Figure 5-9. Completion panel

```
TSF4                                TSF Table Specification - Completion                                7.1.0

Table Name:      CLIENTS                                MODEL 204 File:  CLIENTS

1  Multi-Column Unique Definition
2  Grant Authorization Screen
3  View DDL at Terminal
4  Generate DDL to Output File
5  Return to Main Menu

Selection.....: 4
"USE" Cmd Arg:  OUTDDGN_____
                  (Output file for Selection 4

===>
1=HElP                                3=QUIt
```

Completion panel functions

The Completion panel selections perform the following functions:

This Completion panel selection...	Performs this action...
Multi-Column Unique Definition	Takes you to panels to identify columns that combine together to form a unique identifier for a row of data.
Grant Authorization panel	Takes you to panels to specify the data needed to generate GRANT statements for this schema.
View DDL at Terminal	Allows you to view at a terminal the DDL generated.
Generate DDL to Output File	<p>Allows you to route DDL to a specified output location. You can specify any valid argument for a USE command. For example:</p> <pre>OUTxxxxxx, PRINTER DALLAS, \$PRINT ROUTE *</pre> <p>If the output destination is an output file (OUTxxxxx), define the output file with a disposition of MOD if you intend to generate DDL several times before processing the output file.</p>

This Completion panel selection...	Performs this action...
Return to Main Menu	Enables you to define a new table.

Selection

At the Selection prompt, enter the number of the task you want and press Enter.

“USE” Cmd Arg

If you specify Option 4 (to generate DDL), you must also specify where the output DDL is to be written. Enter a string of characters that make a valid argument for the Model 204 USE command. For example:

```
OUTxxxxxx
PRINTER DALLAS
$PRINT R *
```

Error condition

If you have changed the names of columns or the selections of the columns in a table after defining your GRANT and UNIQUE statements, the following error message appears when you attempt to generate DDL:

```
Column Names have been altered, revalidate GRANT & UNIQUE
definitions
```

Because the GRANT and UNIQUE definitions were edited with a now-altered table definition, you must reedit these GRANT and UNIQUE definitions before you can generate DDL. Select Options 1 and 2 from the Completion panel (refer to Figure 5-9 on page 119) and then revalidate each GRANT and UNIQUE definition by pressing PF8 for each definition to the last definition, and then press PF12.

Defining multicolumn unique keys (Multi-Column Unique panel)

If you select Option 1 from the Completion panel (Figure 5-9 on page 119), the Multi-Column Unique panel is displayed. You can identify a set of columns that, when concatenated together, form a unique constraint key for the rows in the table. By preserving the uniqueness of the combination of the values of these columns, you preserve the uniqueness of the combination of the Model 204 field values associated with the columns.

Figure 5-10 shows a Multi-Column Unique panel with the names filled in of the SQL columns that combine to form a unique key, and with the name of the Model 204 field (RECKEY) that is the concatenated index of the fields associated with the indicated columns.

Note: The RECKEY field in Figure 5-10 is specified for example purposes only. RECKEY is not defined in the Model 204 demonstration database. If you attempt to reproduce this example, you receive an error message. For the example to work, you must define and populate a special index field in the demonstration database.

When you complete your definition, press PF5 to edit the panel, store the updates, and provide an empty panel to generate a new multi-column unique definition. Press PF7 or PF8 to move to the previous or to the next definition.

Figure 5-10. Multi-Column Unique panel

TSF5	TSF Table Specification - Multi-Column Unique		7.1.0
Table Name:	CLIENTS	MODEL 204 File:	CLIENTS
MODEL 204 Field:	RECKEY_____		
Column Name(s) :	RECTYPE_____		
	POLNO_____		
	SORTKEY_____		

Adding a new definition.			
===>		<ENTER>=Validate	
1=HELp	3=QUIT	5=ADDnew	
7=PREvious	8=NEXt	12=END	

Specifying a multicolumn unique key

The Model 204 SQL Server supports multicolumn unique keys only if there is an ORDERED CHAR UNIQUE field (which can be INVISIBLE) defined in the Model 204 file that is generated by the concatenation of data in the individual columns or fields. That is, a Model 204 DEFINE FIELD command must be issued for this field before the multicolumn unique key is defined to the SQL catalog.

The order in which you list the columns that compose the key determines the order in which they are concatenated to build the supporting Model 204 index field.

Once the Model 204 index field and multicolumn unique key are defined, you must populate the index with data from its component fields. If you are maintaining the associated Model 204 file exclusively with SQL, you can populate the index automatically with an SQL UPDATE statement. If you are maintaining the file with User Language or the Host Language Interface, you can manually apply the same algorithm the SQL Server uses for the automatic population.

For more information about populating the multicolumn unique index, see “Populating the index field” on page 60. For more information about manually defining a multicolumn unique key, see “Specifying a multicolumn UNIQUE key” on page 58.

Note: A multicolumn UNIQUE key is not allowed in a nested table.

Model 204 Field

The Model 204 field name must be the name of a field defined in the file with the ORDERED CHAR UNIQUE attribute.

Using a multicolumn UNIQUE definition requires some modifications to the existing Model 204 file that is being used. A field name must be added for the concatenated data that results from a multicolumn UNIQUE definition. The field must have the attribute of ORDERED CHAR UNIQUE. The field must be populated with data for existing records. For more information about defining such a field, see “Usage note” on page 123.

Once the table has been defined to the SQL catalog with the multicolumn unique DDL clause, SQL updates or deletions of any of the columns corresponding to the fields automatically update or delete keys from the index field you added to support the multicolumn unique definition.

You might want to define the field as INVISIBLE in Model 204 to save duplicate storage of data. However, if you do define the field as INVISIBLE, make sure that the data in that field is not corrupted by User Language programs (SQL programs do not corrupt the data). If a User Language program deletes any records in a file with the Reuse Record Number option active, that program must also explicitly delete any INVISIBLE fields associated with those records. Refer to the *Model 204 File Manager's Guide* for information about the INVISIBLE attribute.

To get a list of fields defined in the file with the ORDERED CHAR UNIQUE attribute, place the cursor on the input field and press PF1. On the help panel, use any character to select the field value you need, and that value appears on the panel in the input field when you return to the Multi-Column Unique panel.

Note: Due to physical panel size limitations, the Model 204 field name you provide can be no longer than 58 characters. Ordinarily, Model 204 field names can be as many as 255 characters. If you require more than 58 characters for the field name, do not use the TSF to map this field. Manually include DDL creating this key before submitting it to the CVI.

Column Name(s)

You must specify at least one column name; you cannot specify more than ten. Each name must be a valid column name defined in this table.

The TSF concatenates the SQL column names you specify and compares the resulting name to the corresponding Model 204 field name. If the resulting SQL name does not match the Model 204 field name, the TSF automatically adds a SYSNAME clause with the Model 204 field name to the TSF-generated DDL. The SYSNAME value is the actual name of the Model 204 field.

To get a list of columns that can be selected, place the cursor on the field and press PF1. On the help panel use any character to select the value(s) you need and these value(s) appear on the panel at the Column Name input field(s) when you return to the Multi-Column Unique panel.

You must not specify columns that are mapped to INVISIBLE fields.

For an example of how the SQL Server concatenates the SQL column names, see “Using the multicolumn unique key algorithm” on page 61. For more information about the Model 204 SQL SYSNAME extension, see “Column naming and the SYSNAME extension” on page 58.

Usage note

Remember, if you are defining a multicolumn unique field for an existing Model 204 file that contains data, you must take the following actions:

1. Define the field that is to be the concatenated index as ORDERED CHAR UNIQUE (and preferably INVISIBLE).
2. Populate the new index with data from its component fields, as described on “Populating the index field” on page 60.

Specifying GRANT authority (Grant Authority panel)

If you select Option 2 from the Completion panel (Figure 5-9 on page 119), the Grant Authority panel appears. With the Grant Authority panel you can generate GRANT statements for the schema in which the table is being defined.

For more information about the Model 204 SQL implementation of GRANT, see “Granting privileges for SQL objects” on page 81.

Figure 5-11 shows a Grant Authority panel with user specifications for granting privileges for all four DML actions to all valid Model 204 SQL users. These

Grant option

You must select either Y for Yes or N for No.

Usage note

When you complete your definition, press PF5 to edit the panel, store the updates to the database, and provide a new empty panel to generate a new GRANT definition. Press PF7 or PF8 to move to the previous or to the next GRANT definition.

Viewing DDL at the terminal (Completion panel)

To view the generated DDL on the screen, select Option 3 from the Completion panel (Figure 5-9 on page 119). The DDL generated by the Table Specification facility appears.

To page through the output, press Enter. Pressing Enter at the last page of output brings you back to the Completion panel.

An example of DDL displayed at a terminal is shown in Figure 5-12, which continues onto the next page.

Note: The DDL at the bottom of the generated stream that defines the multicolumn unique key is displayed for example purposes only. You cannot generate such DDL using an unaltered version of the Model 204 demonstration database. The demonstration database includes no field designed to serve as a multicolumn unique key.

Figure 5-12. TSF-generated DDL

```
CREATE SCHEMA DEMO AUTHORIZATION MARK
CREATE TABLE CLIENTS
( CITY
  CHAR(30),
  BIRTHDAY
  SYSNAME 'DATE OF BIRTH'
  CHAR(8),
  DRIVERID
  SYSNAME 'DRIVER ID'
  INTEGER,
  SORTKEY
  SYSNAME 'FULLNAME'
  CHAR(30),
  POLNO
  SYSNAME 'POLICY NO'
  CHAR(6) NOT NULL,
  RECTYPE
  CHAR(12),
  SEX
  CHAR(1),
  STATE
  CHAR(20),

  >
```

Figure 5-13. TSF-generated DDL (continued)

```

        UNIQUE ( CLAIM_NO
                POLICY_NO
        SYSNAME 'RECKEY' )

GRANT ALL PRIVILEGES ON CLAIMS02 TO PUBLIC
WITH GRANT OPTION

```

Generating DDL to an output file (Completion panel)

To generate DDL into a specified output file, select Option 4 from the Completion panel in Figure 5-9 on page 119).

You must either dynamically allocate or have your system manager define or allocate this output file prior to your using it from the TSF. For best results under z/OS (and comparably for VM and VSE), make sure that the following is specified in the DD statement or ALLOCATE command for the output file (data set):

- Data set name begins with *OUT*.
- DISP=MOD, if you intend to generate DDL several times before processing the output file
- RECFM=FS, to prevent the printing to the file of carriage control characters
- LRECL=80, to facilitate the reading of the file by CVI

After generating DDL to an output file, you receive the following message:

```

*****
*                                                                 *
* WARNING: Generation of DDL to an output file causes this utility *
* to delete all work records generated by this utility.           *
* If you do not wish to delete these work records at this time    *
* respond "N" to the question below, and remember that the deletion *
* of these records will be a manual process under your control.    *
*                                                                 *
*****

$$Proceed with delete (Y/N)?
>

```

Delete the work file records after you have stored the DDL from the TSF in an output file.

You can use your editor to manually adjust or add to the DDL stream generated by the TSF before you submit the DDL to the CVI.

As stated earlier, the CVI utility requires that the DDL input statement must each be delimited, which the CCATSF utility currently does not generate. This

delimiter character, the semicolon (;), must be manually supplied by the user, if the CVI utility is to be used.

6

Getting Information from the SQL Catalog

In this chapter

- Overview
- SQL catalog reporting with CCACATREPT
- CCACATREPT Main Menu
- Report Selection 1: Generate DDL
- Report Selection 2: Formatted Table/View report
- Report Selection 3: Privilege report by table/view
- Report Selection 4: Privilege report by grantee
- Querying the SQL catalog

Overview

To review the current contents of the SQL catalog you can query the catalog directly or you can use CCACATREPT, the SQL catalog reporting utility. This chapter describes CCACATREPT and the queriable views of the SQL catalog.

SQL catalog reporting with CCACATREPT

The CCACATREPT subsystem is a reporting mechanism for data in the CCACAT SQL catalog file. CCACATREPT is a menu-driven utility that

produces both online and printed output. With CCACATREPT you can produce a report that provides information about all SQL objects defined in the SQL catalog.

Catalog administrators can compare the CCACATREPT output to the Model 204 file data to determine the updates they need to make to the SQL catalog to obtain consistency with the file. They can modify the report's generated DDL and use it to repopulate the SQL catalog. SQL application programmers and SQL users can use the output to verify valid table, view, column names and column attributes.

This section describes the CCACATREPT menu and provides examples of each of the output formats. CCACATREPT produces the following types of online and printed output:

- File of valid DDL syntax, which you can use to rebuild the catalog
- Fixed format report of a table or view
- Privilege report by table or view name
- Privilege report by user

As an Application Subsystem utility, CCACATREPT is subject to typical Application Subsystem security, as described in the *Model 204 System Manager's Guide*.

Logging in

Before you can log in to CCACATREPT, the CCACATREPT subsystem must already have been started with the Model 204 START SUBSYSTEM command. To log in to CCACATREPT, at your Model 204 Online prompt enter:

```
CCACATREPT
```

and press Enter.

The CCACATREPT Main Menu is displayed.

CCACATREPT Main Menu

The Main Menu of CCACATREPT in Figure 6-1 displays the four output formats, their input parameters, and their output destination options.

Figure 6-1. CCACATREPT Main Menu

```

CATREPT1      SQL Catalog Reporting Facility (CCACATREPT)      7.1.0
              Main Menu

1  Generate DDL Statements from Existing Catalog Data
2  Formatted Table/View Report
3  Privilege Report by Table/View
4  Privilege Report by Grantee

Selection: █

Schema Name   : _____ (Reports 1, 2, 3)
Authorization ID : _____ (Reports 1, 2)
Table/View Name : _____ (Reports 1, 2, 3)
DDL Statement Type(s): _____ (Report 1)
                                           (T=Table, V=View, G=Grant)
Grantee       : _____ (Report 4)

"USE" Command Arg... : _____

===>
1=HELP      2=REFresh    3=QUIT

LU003 PLU
  
```

Using the CCACATREPT panel

After you make final panel field selections and generate a report, CCACATREPT redisplay the panel with those selections.

You can enter an asterisk (*) in an input area to select all the items of a particular category.

The command line (===>) is at the bottom of the panel. Use the command line to enter commands in lieu of PF keys. The minimum abbreviation for a command is generally the first three characters, for example, REF for REFresh. The minimum abbreviation appears in capital letters in the PF key display area.

The CCACATREPT main menu has the following PF keys:

Key	Explanation
PF1 = HELp	<p>You can access either general help information for the panel or additional individual panel field help information.</p> <p>Access general help information for this panel by placing the cursor under the input area of any panel field except Schema Name, Authorization ID, Table/View Name, or Grantee, and press PF1.</p> <p>Access individual panel field help information by placing the cursor under the input area of Schema Name, Authorization ID, Table/View Name, or Grantee. Put the cursor anywhere in the input area of the field you want, and press PF1. The current set of values in the SQL catalog for this field is displayed. If you select one of these values by entering any character on the line to the left of the value, it is returned to the input area of the field in question.</p>
PF2 = REFresh	Refresh the screen, and do not process the current data on the screen. The refreshed screen appears as it does when you first entered the panel.
PF3 = QUIT	Exit from CCACATREPT. Entries are not saved when you use PF3.

Selection field

Using the Selection field you can generate output from any of the four reports numbered one through four on the menu. Enter an integer from 1 through 4, corresponding to the reports numbered on the menu. The report number you choose determines the input parameter fields. You must supply one of the values that are listed below the Selection field. The report input parameter fields are followed by a specification in parentheses of the reports for which a value is required.

If you supply a value for a report that does not require it, you receive an error message asking you to remove your incorrect value.

Schema Name field

You must specify an entry in this field if you entered 1, 2, or 3 in the Selection field. Enter the name of a specific schema or enter an asterisk (*) for all schemas.

If your Selection field entry requires an entry for this field and you do not enter a value, by default CCACATREPT outputs all schemas.

Authorization ID field

You must specify an entry in this field if you entered 1 or 2 in the Selection field. Enter the name of a specific authorization ID or enter an asterisk (*) for all authorization IDs.

If your Selection field entry requires an entry for this field and you do not enter a value, by default CCACATREPT outputs all authorization IDs.

Table/View Name field

You must specify an entry in this field if you entered 1, 2, or 3 in the Selection field. Enter the name of a specific table or view to be reported or enter an asterisk (*) for all tables and views.

If your Selection field entry requires an entry for this field and you do not enter a value, by default CCACATREPT outputs all tables and views.

DDL Statement Type(s) field

You must specify an entry in this field if you entered 1 in the Selection field. The output from Selection 1 is standard SQL DDL generated from the Model 204 SQL catalog. In the DDL Statement Type(s) field, you identify the DDL statement types to be generated. Choose one or more of the following identifiers:

Identifier	Statement generated
T	CREATE TABLE
V	CREATE VIEW
G	GRANT
T, V, or G	CREATE SCHEMA or SET SCHEMA CREATE TABLE, CREATE VIEW, and GRANT

If your Selection field entry requires an entry for this field and you do not enter a value, by default CCACATREPT outputs all the DDL.

Grantee field

You must specify an entry in this field if you entered 4 in the Selection field. Enter the SQL identifier of a specific grantee to be reported or enter an asterisk (*) for all grantees. Privileges granted to PUBLIC are always reported.

If your Selection field entry requires an entry for this field and you do not enter a value, by default CCACATREPT outputs privileges for all grantees.

“USE” Command Arg

Use this field to route output to a destination other than to the terminal. The character string you enter in this field becomes the Model 204 USE command argument that identifies the output destination device. You can enter any USE command argument that is valid in your operating environment, for example, PRINTER FOO or \$PRINT *.

You can route your output to a file. Either dynamically allocate or have your system manager define or allocate an output file prior to your using it from CCACATREPT. For best results under z/OS (and comparably for VM and VSE), make sure that the following are specified in the DD statement or ALLOCATE command for the output file (data set):

- DD name begins with *OUT*.
- DISP=MOD, if you intend to generate multiple reports before processing the output file
- RECFM=FS, to prevent the printing to the file of carriage control characters
- LRECL=80, to facilitate the reading of the file by CVI or other DDL utility

The Model 204 USE command is described in the *Model 204 Parameter and Command Reference*.

Report Selection 1: Generate DDL

The first reporting option on the Main Menu is to generate the SQL DDL from the existing SQL catalog (CCACAT file) data. This DDL is output in valid SQL syntax that you can use to rebuild the catalog. You can also use it to report the existing catalog data in a standard format familiar to SQL users.

Note: Currently the CCACATREPT subsystem does not generate DDL statement delimiters. However, semicolons are required by the CVI utility. You can add delimiters by hand or use another SQL utility that does not require DDL delimiters. The Connect★ ODBC unsupported utilities, DDLWIN and CLIIVP, are examples of utilities which accept DDL input lines without delimiters.

Specifying report input parameters

The required input parameters are:

Required parameter	Enter...
Schema Name	Specific name or “*”
Authorization ID	Specific ID or “*”
Table/View Name	Specific name or “*”

Required parameter	Enter...
DDL Statement Types	String of identifiers for the DDL statement types to be generated Options (enter one or more, without commas or spaces) are: T=Table V=View G=Grant

Report input parameter examples

The following examples describe how the report input parameters are used.

Example 1

Specifying the parameters in this example produces CREATE SCHEMA, CREATE TABLE, CREATE VIEW, and GRANT DDL statements for all CCACAT objects.

```
Schema Name = *
Authorization ID = *
Table/View Name = *
DDL Statement Types = tvg
```

Example 2

Specifying the parameters in this example produces SET SCHEMA and CREATE TABLE DDL statements for all tables with the authorization ID of USERXXX.

```
Schema Name = *
Authorization ID = userxxx
Table/View Name = *
DDL Statement Types = t
```

Example 3

Specifying the parameters in this example produces SET SCHEMA, CREATE TABLE, and GRANT DDL statements for the table USERXXX.CLIENTS.

```
Schema Name = *
Authorization ID = userxxx
Table/View Name = clients
DDL Statement Types = tg
```

Ordering the DDL output

The format of the DDL generated by this report option is the same as that generated by the TSF.

To set the schema context, CCACATREPT always places the CREATE SCHEMA or SET SCHEMA statement before the rest of the generated DDL. If you plan to process the generated DDL against the SQL catalog, you might have to change CREATE SCHEMA to SET SCHEMA, because issuing CREATE SCHEMA for a schema that already exists is an error.

For more information about creating schemas, changing the default schema context, and SQL DDL statement processing, see “Setting the schema and user context” on page 75.

You must order statements correctly to avoid forward referencing problems when the generated DDL is processed. For example, a view of a view must reference an already created view. The generated DDL sort order follows:

1. Schemas in the order that they were originally input into the SQL catalog.
2. Within a schema, all base and parent tables in alphanumeric order, followed by all nested tables in alphanumeric order. Privileges (GRANT statements) for each table follow that table.
3. Views follow tables. They are in the order that they were originally input into the SQL catalog. GRANT statements for each view follow that view.
4. Grants are ordered by privilege, column name, GRANT statement option, and grantee.

Sample of generated DDL

The sample DDL stream in Figure 6-2 is generated by the following input parameter specifications:

```
Schema Name = demo
Authorization ID = *
Table/View Name = clients
DDL Statement Types = tvg
```

Figure 6-2. Sample DDL output

```
SET SCHEMA DEMO

CREATE TABLE CLIENTS
( ADDRESS
  CHAR(40),
  AGENT
  CHAR(20),
  ANNIV_DATE
  SYSNAME 'ANNIV DATE'
```

```

        INT,
    CITY
        CHAR(20),
    DATE_OF_BIRTH
        SYSNAME 'DATE OF BIRTH'
        INT,
    DRIVER_ID
        SYSNAME 'DRIVER ID'
        INT,
    FULLNAME
        CHAR(40) NOT NULL,
    MARITAL_STATUS
        SYSNAME 'MARITAL STATUS'
        CHAR(15),
    POLICY_NO
        SYSNAME 'POLICY NO'
        CHAR(6) NOT NULL,
    POLICYHOLDER
        CHAR(40),
    RECTYPE
        CHAR(15) NOT NULL,
    RESTRICTIONS
        CHAR(255),
    SEX
        CHAR(1),
    STATE
        CHAR(25),
    TOTAL_PREMIUM
        SYSNAME 'TOTAL PREMIUM'
        INT,
    ZIP
        CHAR(9),
    PID
        INT NOT NULL PRIMARY KEY SYSTEM)

-- GRANT STATEMENTS FOR TABLE: CLIENTS

GRANT DELETE ON CLIENTS TO PUBLIC
GRANT INSERT ON CLIENTS TO PUBLIC
GRANT SELECT ON CLIENTS TO PUBLIC
GRANT UPDATE ON CLIENTS TO PUBLIC

```

Report Selection 2: Formatted Table/View report

The Formatted Table/View report has a fixed format that provides the same information as the DDL generation report (except for granted privilege information), but in a different format.

Report input parameters

Input parameters for the report are:

- Specific schema name (or “*”)
- Specific authorization ID (or “*”)
- Specific table or view name (or “*”)

Contents of the report

The report provides the following information for tables (see the section “Sample report”):

- Name of the SQL table
- Column names in the table
- SQL attributes of table columns
- Column names making up any multi-column unique columns
- Name of the corresponding Model 204 file
- Name of the corresponding Model 204 field
- Authorization ID

Note: The report does not provide information about views that reference a given table.

The report provides the following information for views:

- Name of the SQL view
- View definition text

Sample report

In Figure 6-3 on page 138 the sample output is generated by the following input parameter specifications:

```
Schema Name = demo
Authorization ID = userxxx
Table/View Name = clients
```

The output is a formatted table/view report of the DEMO.CLIENTS table.

Figure 6-3. Sample Table/View report

```
18 NOV 09                                SQL SCHEMA, TABLE, VIEW REPORT

SCHEMA: DEMO                            AUTHORIZATION: USERXXX
```


TABLE: CLIENTS TABLE-TYPE: PARENT PRIMARY-KEY: PID
M204-FILE: CLIENTS

SQL-COLUMN-NAME	DATA-TYPE	NULL	UNQ	MODEL-204-FIELD-NAME
ADDRESS	CHAR(40)	Y	N	ADDRESS
AGENT	CHAR(20)	Y	N	AGENT
ANNIV_DATE	INT	Y	N	ANNIV DATE
CITY	CHAR(20)	Y	N	CITY
DATE_OF_BIRTH	INT	Y	N	DATE OF BIRTH
DRIVER_ID	INT	Y	N	DRIVER ID
FULLNAME	CHAR(40)	N	N	FULLNAME
MARITAL_STATUS	CHAR(15)	Y	N	MARITAL STATUS
POLICY_NO	CHAR(6)	N	N	POLICY NO
POLICYHOLDER	CHAR(40)	Y	N	POLICYHOLDER
RECTYPE	CHAR(15)	N	N	RECTYPE
RESTRICTIONS	CHAR(255)	Y	N	RESTRICTIONS
SEX	CHAR(1)	Y	N	SEX
STATE	CHAR(25)	Y	N	STATE
TOTAL_PREMIUM	INT	Y	N	TOTAL PREMIUM
ZIP	CHAR(9)	Y	N	ZIP
PID	INT	N	Y	PID

Report Selection 3: Privilege report by table/view

The Privilege report by table/view displays the DML update privileges for each table and view indicated by your input parameter specification. The update operations permitted per table and view, including WITH GRANT OPTION, are displayed for each grantee and for grants to PUBLIC.

The report has a fixed format. The sort order is by object name (schema name and table or view name), within that by grantee, and within that by column name, if applicable.

Report input parameters

Input parameters for the report are:

- Specific schema name (or "**")
- Specific table or view name (or "**")

Report display fields

Table 6-1 lists the report display fields and their meanings.

Table 6-1. Privilege report by table/view display fields

Field	Meaning
TABLE/VIEW	Name of the table or view for which indicated privileges are valid.
GRANTEE	User ID for which the privileges are granted.
SEL	If X, privilege to issue DML SELECT statement. If blank, no such privilege.
INS	If X, privilege to issue DML INSERT statement. If blank, no such privilege.
DEL	If X, privilege to issue DML DELETE statement. If blank, no such privilege.
UPD	If X, privilege to issue DML UPDATE statement. If blank, no such privilege.
OPT	If Y, option to grant this privilege to others. If N, no such privilege.
COLUMN-NAME	For UPDATE only, specific columns for which the privilege is granted. Asterisk (*) means all columns.

Sample Privilege Report by table and view

The sample output in Figure 6-4 on page 140 is generated by the following input parameter specifications:

```
Schema Name = demo
Table/View Name = *
```

The output is a privilege report for all the tables and views in the DEMO schema.

Figure 6-4. Sample Privilege Report by table/view

```
18 NOV 09      SQL PRIVILEGE REPORT -- BY TABLE/VIEW      PAGE:      1

TABLE/VIEW: DEMO.ACCIDENTS

GRANTEE          SEL-OPT  INS-OPT  DEL-OPT  UPD-OPT  COLUMN-NAME
PUBLIC           X  N     X  N     X  N     X  N
USERXXX          X  Y     X  Y     X  Y     X  Y

TABLE/VIEW: DEMO.CLAIMS03
```

Report Selection 3: Privilege report by table/view

GRANTEE	SEL-OPT	INS-OPT	DEL-OPT	UPD-OPT	COLUMN-NAME
PUBLIC	X N	X N	X N	X N	
USERXXX	X Y	X Y	X Y	X Y	

TABLE/VIEW: DEMO.CLIENTS

GRANTEE	SEL-OPT	INS-OPT	DEL-OPT	UPD-OPT	COLUMN-NAME
PUBLIC	X N	X N	X N	X N	
USERXXX	X Y	X Y	X Y	X Y	

TABLE/VIEW: DEMO.DRIVERS

GRANTEE	SEL-OPT	INS-OPT	DEL-OPT	UPD-OPT	COLUMN-NAME
PUBLIC	X N	X N	X N	X N	
USERXXX	X N	X N	X N	X Y	

TABLE/VIEW: DEMO.INSURED_VINS

GRANTEE	SEL-OPT	INS-OPT	DEL-OPT	UPD-OPT	COLUMN-NAME
PUBLIC	X N	X N	X N	X N	
USERXXX	X Y	X Y	X Y	X Y	

TABLE/VIEW: DEMO.OTHER_DRIVER

GRANTEE	SEL-OPT	INS-OPT	DEL-OPT	UPD-OPT	COLUMN-NAME
PUBLIC	X N	X N	X N	X N	
USERXXX	X Y	X Y	X Y	X Y	

TABLE/VIEW: DEMO.POLICIES

GRANTEE	SEL-OPT	INS-OPT	DEL-OPT	UPD-OPT	COLUMN-NAME
PUBLIC	X N	X N	X N	X N	
USERXXX	X N	X N	X N	X Y	

TABLE/VIEW: DEMO.VEHICLES

GRANTEE	SEL-OPT	INS-OPT	DEL-OPT	UPD-OPT	COLUMN-NAME
PUBLIC	X N	X N	X N	X N	
USERXXX	X Y	X Y	X Y	X Y	

18 NOV 09

SQL PRIVILEGE REPORT -- BY TABLE/VIEW

PAGE: 2

Report Selection 4: Privilege report by grantee

The Privilege report by grantee displays the DML update privileges for each grantee indicated by your input parameter specification. The update operations permitted, including WITH GRANT OPTION, are displayed for each table or view. Privileges granted to PUBLIC are displayed under GRANTEE=PUBLIC.

The report has a fixed format. The sort order is by grantee, within that by object name (schema name and table or view name), and within that by column name, if applicable.

Report input parameter

The required input parameter for the report is: Specific grantee (or “*”).

Report display fields

The report display fields have the same meanings as those for the Privilege report by table and view, shown in Table 6-1 on page 140.

Sample report

A sample of this report appears in Figure 6-5, generated by the following input parameter specification:

Grantee = userxxx

The output is a report of all the privileges granted to USERXXX or to PUBLIC.

Figure 6-5. Sample Privilege report by grantee

```

18 NOV 09          SQL PRIVILEGE REPORT -- BY GRANTEE          PAGE:      1

GRANTEE: PUBLIC

TABLE/VIEW          SEL-OPT  INS-OPT  DEL-OPT  UPD-OPT  COLUMN-NAME
DEMO.ACCIDENTS      X  N      X  N      X  N      X  N
DEMO.CLAIMS03       X  N      X  N      X  N      X  N
DEMO.CLIENTS        X  N      X  N      X  N      X  N
DEMO.DRIVERS        X  N      X  N      X  N      X  N
DEMO.INSURED_VINS   X  N      X  N      X  N      X  N
DEMO.OTHER_DRIVER   X  N      X  N      X  N      X  N
DEMO.POLICIES       X  N      X  N      X  N      X  N
DEMO.VEHICLES       X  N      X  N      X  N      X  N

GRANTEE: USERXXX

TABLE/VIEW          SEL-OPT  INS-OPT  DEL-OPT  UPD-OPT  COLUMN-NAME
DEMO.ACCIDENTS      X  Y      X  Y      X  Y      X  Y
DEMO.CLAIMS03       X  Y      X  Y      X  Y      X  Y

```

DEMO.CLIENTS	X	Y	X	Y	X	Y	X	Y
DEMO.DRIVERS	X	N	X	N	X	N	X	Y
DEMO.INSURED_VINS	X	Y	X	Y	X	Y	X	Y
DEMO.OTHER_DRIVER	X	Y	X	Y	X	Y	X	Y
DEMO.POLICIES	X	N	X	N	X	N	X	Y
DEMO.VEHICLES	X	Y	X	Y	X	Y	X	Y

Querying the SQL catalog

You can directly query the Model 204 SQL catalog to obtain information about the current catalog definitions. Any authorized SQL user can access (but not update) any of 18 views of the catalog contents. You use a normal SQL query and devise your own report format for information from the catalog views.

The SQL catalog contains a schema named CATALOG that is comprised of the views of the catalog contents. The views are defined by SQL object. For example, the schemas view in CATALOG contains all the schemas in the catalog by name and by authorization ID.

The CATALOG views are listed here and described individually in the rest of this section.

```

SCHEMAS (or SCHEMATA)
TABLES
TABLE_COLUMNS
COLUMNS (for tables and views)
VIEWS
TABLE_CONSTRAINTS
KEY_COLUMN_USAGE
TABLE_PRIVILEGES
COLUMN_PRIVILEGES
ODBC_TYPES
ODBC_SCALES
CONST
ODBC_COLUMNS
ODBC_SPECIAL_COLS
ODBC_TABLES
ODBC_TABLE_STATS
ODBC_KEY_STATS
ODBC_STATISTICS

```

Querying a CATALOG view

Authorized Model 204 SQL users can query any of the CATALOG views like any other SQL object. However, the views display to you only the SQL objects or information that you have the authority to see or that are granted to PUBLIC.

For example, USERXXX issues the following query against the Model 204 demonstration database:

```
SELECT * FROM CATALOG.TABLES
```

The result is a display of information about the tables USERXXX has the authority to select or update. The query's output is shown in Figure 6-6. The format of the output depends on your application and is your responsibility.

The CATALOG.TABLES columns (described in "TABLES view" on page 146), ordered from left to right, are:

```
TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, FILE_NAME,
TABLE_TYPE, COLUMN_CARDINALITY, PRIMARY_KEY, SYSTEM_KEY,
PARENT_TABLE_NAME, TIMESTAMP
```

This is the default display order, which derives from the order of the columns in the individual CATALOG view definition.

Figure 6-6. SELECT * from CATALOG.TABLES

CCACAT	CATALOG	SCHEMAS		V	4		4		19920911000004
CCACAT	CATALOG	TABLES		V	10		10		19920911000007
CCACAT	CATALOG	TABLE_COLUMNS		V	11		11		19920911000010
CCACAT	CATALOG	COLUMNS		V	8		8		19920911000000
CCACAT	CATALOG	VIEWS		V	6		6		19920911000016
CCACAT	CATALOG	TABLE_CONSTRAINTS		V	6		6		19920911000019
CCACAT	CATALOG	KEY_COLUMN_USAGE		V	5		5		19920911000022
CCACAT	CATALOG	TABLE_PRIVILEGES		V	8		8		19920911000025
CCACAT	CATALOG	COLUMN_PRIVILEGES		V	9		9		19920911000028
CCACAT	CATALOG	ODBC_TYPES		V			16		19981201142400
CCACAT	CATALOG	ODBC_SCALES		V			2		19981201142400
CCACAT	CATALOG	CONST		V			9		19981201142400
CCACAT	CATALOG	ODBC_COLUMNS		V			17		19981201142400
CCACAT	CATALOG	ODBC_SPECIAL_COLS		V			10		19981201142400
CCACAT	CATALOG	ODBC_TABLES		V			5		19981201142401
CCACAT	CATALOG	ODBC_TABLE_STATS		V			12		19981201142401
CCACAT	CATALOG	ODBC_KEY_STATS		V			12		19981201142401
CCACAT	CATALOG	ODBC_STATISTICS		V			12		19981201142401
CCACAT	USERXXX	DRIVERS		V	9				19921117183249
CCACAT	USERXXX	POLICIES		V	9				19921117183250
CCACAT	USERXXX	CLIENTS	CLIENTS	T	17	PID	1		19921117183251
CCACAT	USERXXX	ACCIDENTS	CLIENTS	T	3	PID	1	CLIENTS	19921117183253
CCACAT	USERXXX	INSURED_VINS	CLIENTS	T	2	PID	1	CLIENTS	19921117183254
CCACAT	USERXXX	VEHICLES	VEHICLES	T	19	VIN	0	CLIENTS	19921117183254
CCACAT	USERXXX	OTHER_DRIVER	VEHICLES	T	2	VIN	0	VEHICLES	19921117183256
CCACAT	USERXXX	CLAIMS03	CLAIMS03	T	13	VIN	0	VEHICLES	19921117183258

The output in Figure 6-6 is from two schemas: CATALOG and USERXXX. The CATALOG schema contains the 18 views that describe the catalog contents and is granted to PUBLIC. You can limit the output to the tables in schema USERXXX by using the following SELECT statement:

```
SELECT * FROM CATALOG.TABLES WHERE TABLE_SCHEMA='USERXXX'
```

You can list specific columns for display instead of using `SELECT *`. For example, the following statement results in the same display as that in Figure 6-6:

```
SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
       FILE_NAME, TABLE_TYPE, COLUMN_CARDINALITY,
       PRIMARY_KEY, SYSTEM_KEY, PARENT_TABLE_NAME,
       TIMESTAMP
FROM CATALOG.TABLES
```

The output in Figure 6-6 is grouped by schema, because that is how the DDL used to populate the SQL catalog was organized. An unqualified SQL catalog query returns data chronologically in the order in which the SQL objects were defined or updated in the SQL catalog. As new tables are added to which USERXXX is granted access, this convenient grouping is likely to be lost.

You can ensure that your output is appropriately ordered by sorting the rows by column value with the `ORDER BY` clause. For example, the following statement sorts the query output by table name, as shown in Figure 6-7:

```
SELECT * FROM CATALOG.TABLES WHERE TABLE_SCHEMA='USERXXX'
ORDER BY TABLE_SCHEMA, TABLE_NAME
```

Figure 6-7. `SELECT *` with `ORDER BY` from `CATALOG.TABLES`

CCACAT	USERXXX	ACCIDENTS	CLIENTS	T	3	PID	1	CLIENTS	19921117183253
CCACAT	USERXXX	CLAIMS03	CLAIMS03	T	13	VIN	0	VEHICLES	19921117183258
CCACAT	USERXXX	CLIENTS	CLIENTS	T	17	PID	1		19921117183251
CCACAT	USERXXX	DRIVERS		V	9				19921117183249
CCACAT	USERXXX	INSURED_VINS	CLIENTS	T	2	PID	1	CLIENTS	19921117183254
CCACAT	USERXXX	OTHER_DRIVER	VEHICLES	T	2	VIN	0	VEHICLES	19921117183256
CCACAT	USERXXX	POLICIES		V	9				19921117183250
CCACAT	USERXXX	VEHICLES	VEHICLES	T	19	VIN	0	CLIENTS	19921117183254

Rules for CATALOG queries

- A query against a table that does not exist returns an empty result.
- A query against a table that you do not have the authority to access returns an empty result.
- You cannot issue update SQL DML against the CATALOG views.

SCHEMAS view

Table 6-2 lists the `CATALOG.SCHEMAS` view columns and their descriptions. This view contains a row of identifying information for each schema whose authorization ID is your login ID.

SCHEMATA is an alias for SCHEMAS. You can query either SCHEMAS or SCHEMATA.

The top-to-bottom order of the columns in Table 6-2 is the left-to-right order of the columns in each row of output you receive from an unqualified `SELECT *` query against the view.

Table 6-2. SCHEMAS view columns

Column name	Data type	Description
SCHEMA_CATALOG	CHAR(18)	Name of the catalog where the schema is cataloged (CCACAT)
SCHEMA_NAME	CHAR(18)	Name of the schema
SCHEMA_OWNER	CHAR(18)	Authorization ID of the owner of the schema
TIMESTAMP	CHAR(14)	Greenwich mean time when the schema description was cataloged

TABLES view

Table 6-3 lists the CATALOG.TABLES view columns and their descriptions. This view contains a row of descriptive information for each table or view you have the authority to access.

The top-to-bottom order of the columns in Table 6-3 is the left-to-right order of the columns in each row of output you receive from an unqualified `SELECT *` query against the view.

Table 6-3. TABLES view columns

Column name	Data type	Description
TABLE_CATALOG	CHAR(18)	Name of the catalog in which the table is cataloged (CCACAT)
TABLE_SCHEMA	CHAR(18)	Name of the schema to which the table belongs
TABLE_NAME	CHAR(18)	Name of the table
FILE_NAME	CHAR(8)	Name of the Model 204 file to which the table maps
TABLE_TYPE	CHAR(1)	Type of table: either <i>T</i> (table), or <i>V</i> (view)
COLUMN_CARDINALITY	INTEGER	Number of columns defined in the table being described
PRIMARY_KEY	CHAR(18)	Name of any primary or nested key column; otherwise, null

Table 6-3. TABLES view columns (Continued)

Column name	Data type	Description
SYSTEM_KEY	CHAR(1)	0, if the primary or nested key column is not system-generated 1, if the primary or nested key column is system-generated null, if there is no primary or nested key
PARENT_TABLE_NAME	CHAR(18)	Name of parent table if the table is nested; otherwise, null
TIMESTAMP	CHAR(14)	Greenwich mean time when the table description was cataloged

TABLE_COLUMNS view

Table 6-4 lists the CATALOG.TABLE_COLUMNS view columns and their descriptions. This view contains a row of column definition information for each column in each base or nested table you have the authority to access. The TABLE_COLUMNS view contains column definition information not contained in the COLUMNS view, for example, the name of the Model 204 field to which the column maps.

The top-to-bottom order of the columns in Table 6-4 is the left-to-right order of the columns in each row of output you receive from an unqualified SELECT * query against the view.

Table 6-4. TABLE_COLUMNS view columns

Column name	Data type	Description
TABLE_CATALOG	CHAR(18)	Name of the catalog in which the table is cataloged (CCACAT)
TABLE_SCHEMA	CHAR(18)	Name of the schema to which the table belongs
TABLE_NAME	CHAR(18)	Name of the table
COLUMN_NAME	CHAR(18)	Name of the column
FIELD_NAME	CHAR(255)	Name of the Model 204 field to which the column maps
NUMERIC_PRECISION	INTEGER	Maximum numeric precision of the column values
NUMERIC_SCALE	INTEGER	Numeric scale of the column values
CHAR_MAX_LENGTH	INTEGER	Maximum length of string values, if the column data type is CHAR; otherwise, zero

Table 6-4. TABLE_COLUMNS view columns (Continued)

Column name	Data type	Description
DATA_TYPE	CHAR(10)	SQL data type of the column (not the Model 204 field attribute)
NULL_ALLOWED	CHAR(1)	1, if NULL values are allowed for the column; otherwise, 0
COL_UNIQUE	CHAR(1)	1, if the values in the column are unique; otherwise, 0

COLUMNS view

Table 6-5 lists the CATALOG.COLUMNS view columns and their descriptions. The COLUMNS view contains a row of column definition information for each column in each base table, nested table, or view you have the authority to access.

Column information for views is limited to the column name. For more view column attribute information, query the TABLE_COLUMNS view for the base tables referenced in the view definition FROM clause.

The top-to-bottom order of the columns in Table 6-5 is the left-to-right order of the columns in each row of output you receive from an unqualified SELECT * query against the view.

Table 6-5. COLUMNS view columns

Column name	Data type	Description
TABLE_CATALOG	CHAR(18)	Name of the catalog in which the column's table is cataloged (CCACAT)
TABLE_SCHEMA	CHAR(18)	Name of the schema to which the column's table or view belongs
TABLE_NAME	CHAR(18)	Name of the column's table or view
COLUMN_NAME	CHAR(18)	Name of the column
NUMERIC_PRECISION	INTEGER	Table columns: maximum numeric precision of the table column values View columns: 0 (not stored)
NUMERIC_SCALE	INTEGER	Table columns: numeric scale of the column values View columns: 0 (not stored)
CHAR_MAX_LENGTH	INTEGER	Table columns: maximum length of string values, if the column data type is CHAR; otherwise, zero View columns: 0 (not stored)

Table 6-5. COLUMNS view columns (Continued)

Column name	Data type	Description
DATA_TYPE	CHAR(10)	Table columns: SQL data type of the column (derived for views from the corresponding columns in the underlying base tables) View columns: <i>UNAVAIL</i> (not stored)

VIEWS view

Table 6-6 lists the CATALOG.VIEWS view columns and their descriptions. The VIEWS view contains a row of information including the view definition for each view you have the authority to access.

Except for the view definition, CATALOG.VIEWS does not provide information about tables or columns referenced by views you have the authority to access.

If a view definition has more than 255 characters, it is continued in the VIEW_DEFINITION column on one or more additional rows. Except for the view definition, the column values in these extra rows duplicate those in the row with the beginning of the view definition.

The top-to-bottom order of the columns in Table 6-6 is the left-to-right order of the columns in each row of output you receive from an unqualified SELECT * query against the view.

Table 6-6. VIEWS view columns

Column name	Data type	Description
TABLE_CATALOG	CHAR(18)	Name of the catalog in which the view is cataloged (CCACAT)
TABLE_SCHEMA	CHAR(18)	Name of the schema to which the view belongs
TABLE_NAME	CHAR(18)	Name of the view
VIEW_DEFINITION	CHAR(255)	The view definition source string without the <i>CREATE VIEW viewname</i> phrase; you may want to include TABLE_SCHEMA and TABLE_NAME in your query to help identify the view definition output
CHECK_OPTION	CHAR(1)	1, if WITH CHECK OPTION clause is present in the view definition; otherwise, 0
UPDATABLE	CHAR(1)	1, if data may be inserted, deleted, or updated through this view; otherwise, 0

TABLE_CONSTRAINTS view

Table 6-7 lists the CATALOG.TABLE_CONSTRAINTS view columns and their descriptions. The TABLE_CONSTRAINTS view contains a row of information for each multi-column unique constraint in a base table you have the authority to access.

This view displays only multi-column unique constraints; primary key and foreign key constraints are not displayed. The names of the columns participating in each multi-column unique key are displayed in the KEY_COLUMN_USAGE view.

The top-to-bottom order of the columns in Table 6-7 is the left-to-right order of the columns in each row of output you receive from an unqualified SELECT * query against the view.

Table 6-7. TABLE_CONSTRAINTS view columns

Column name	Data type	Description
CONSTRAINT_CATALOG	CHAR(18)	Name of the catalog in which the constraint table is cataloged (CCACAT)
CONSTRAINT_SCHEMA	CHAR(18)	Name of the schema to which the constraint table belongs
TABLE_NAME	CHAR(18)	Name of the table in which the constraint was defined
CONSTRAINT_FIELD	CHAR(255)	Name of the UNIQUE Model 204 field that is used to generate the Model 204 index and constraints
CONSTRAINT_TYPE	CHAR(1)	<i>U</i> , for unique
TIMESTAMP	CHAR(14)	Greenwich mean time when the table constraint was cataloged

KEY_COLUMN_USAGE view

Table 6-8 lists the CATALOG.KEY_COLUMN_USAGE view columns and their descriptions. The KEY_COLUMN_USAGE view contains a row of information for each multicolumn unique constraint in each base table you have the authority to access. The information includes the name of each column participating in a multicolumn unique key.

This view is only for columns that participate in multicolumn unique constraints; primary key and foreign key constraints are not displayed.

The top-to-bottom order of the columns in Table 6-8 is the left-to-right order of the columns in each row of output you receive from an unqualified `SELECT *` query against the view.

Table 6-8. KEY_COLUMN_USAGE view columns

Column name	Data type	Description
CONSTRAINT_CATALOG	CHAR(18)	Name of the catalog in which the constraint table is cataloged (CCACAT)
CONSTRAINT_SCHEMA	CHAR(18)	Name of the schema to which the constraint table belongs
TABLE_NAME	CHAR(18)	Name of the table in which the constraint is defined
CONSTRAINT_FIELD	CHAR(255)	Name of the UNIQUE Model 204 field that is used to generate Model 204 index and constraints
COLUMN_NAME	CHAR(18)	Name of a column that participates in a multicolumn constraint; one row is returned for each column used in a multicolumn unique key

TABLE_PRIVILEGES view

Table 6-9 lists the `CATALOG.TABLE_PRIVILEGES` view columns and their descriptions. The `TABLE_PRIVILEGES` view contains a row for each table and view for which you have or have granted a DML update privilege.

The top-to-bottom order of the columns in Table 6-9 is the left-to-right order of the columns in each row of output you receive from an unqualified `SELECT *` query against the view.

Table 6-9. TABLE_PRIVILEGES view columns

Column name	Data type	Description
GRANTOR	CHAR(18)	Authorization ID of the user giving the privilege
GRANTEE	CHAR(18)	Authorization ID, which may be PUBLIC (all users), of the user given the privilege
TABLE_CATALOG	CHAR(18)	Name of the catalog in which the table or view is cataloged (CCACAT)
TABLE_SCHEMA	CHAR(18)	Name of the schema to which the table or view belongs
TABLE_NAME	CHAR(18)	Name of the table or view
PRIVILEGE	CHAR(6)	Type of privilege: SELECT, INSERT, DELETE, or UPDATE

Table 6-9. TABLE_PRIVILEGES view columns (Continued)

Column name	Data type	Description
GRANTABLE	CHAR(1)	1, if the privilege includes WITH GRANT OPTION; otherwise, 0
TIMESTAMP	CHAR(14)	Greenwich mean time when the table privilege was cataloged

COLUMN_PRIVILEGES view

Table 6-10 lists the CATALOG.COLUMN_PRIVILEGES view columns and their descriptions. The COLUMN_PRIVILEGES view contains a row for each DML update privilege you have or have granted for a column.

The top-to-bottom order of the columns in Table 6-10 is the left-to-right order of the columns in each row of output you receive from an unqualified SELECT * query against the view.

Table 6-10. COLUMN_PRIVILEGES view columns

Column name	Data type	Description
GRANTOR	CHAR(18)	Authorization ID of the user giving the privilege
GRANTEE	CHAR(18)	Authorization ID, which may be PUBLIC (all users), of the user given the privilege
TABLE_CATALOG	CHAR(18)	Name of the catalog in which the table or view is cataloged (CCACAT)
TABLE_SCHEMA	CHAR(18)	Name of the schema to which the table or view belongs
TABLE_NAME	CHAR(18)	Name of the table or view
COLUMN_NAME	CHAR(18)	Name of a column from the list of columns for which update privilege is granted; one row is returned for each such column
PRIVILEGE	CHAR(6)	Type of privilege: SELECT, INSERT, DELETE, or UPDATE
GRANTABLE	CHAR(1)	1, if the privilege includes WITH GRANT OPTION; otherwise, 0
TIMESTAMP	CHAR(14)	Greenwich mean time when the column privilege was cataloged

ODBC_TYPES view

Table 6-11 lists the CATALOG.ODBC_TYPES and their descriptions. The ODBC_TYPES view contains a row of information for each discrete data type supported by the Connect★ drivers.

Table 6-11. ODBC_TYPES view columns

Column name	Data type	Description
TYPE_NAME	CHAR(16)	Name of a data type that can be used for defining columns to be accessed via ODBC
DATA_TYPE	CHAR(15)	Data type code number for the data type, as specified in the ODBC API
PRECISION_VAL	INTEGER(4)	Maximum precision for this data type. 0 is returned, if precision is not applicable.
LITERAL_PREFIX	CHAR(1)	Character used to prefix a literal. For example, a single quotation mark (') is used for CHAR data type literals.
LITERAL_SUFFIX	CHAR(1)	Character used to terminate a literal, for example, a single quotation mark (') is used to terminal a CHAR data type literal
CREATE_PARAMS	CHAR(15)	Description of the creation parameters for this data type. For example, LENGTH is specified for a CHAR data type.
NULLABLE	SMALLINT(2)	Flag indicating whether columns with this data type are nullable. 1 indicates yes 0 means null not allowed.
CASE_SENSITIVE	SMALLINT(2)	1 means columns of this type are case sensitive. 0 means they are not.
SEARCHABLE	SMALLINT(2)	2 means columns of this type can be used in a WHERE clause on a SELECT, except with LIKE. 3 means they can be used also with LIKE.
UNSIGNED_ATTRIBUTE	SMALLINT	1 means the data type is unsigned. 0 means it is signed.
MONEY	SMALLINT(2)	1 means a monetary data type. 0 means it is not.

Table 6-11. ODBC_TYPES view columns (Continued)

Column name	Data type	Description
AUTO_INCREMENT	SMALLINT(2)	1 means columns of this type are automatically incremented (set to a unique value when a new row is inserted). 0 means they are not automatically incremented.
LOCAL_TYPE_NAME	CHAR(10)	Localized name for this data type on this data source, that is different from the standard data type name
RADIX	SMALLINT(2)	Number base used for showing numeric values for this data type. For example, 10 means that decimal values are shown.
MINIMUM_SCALE	SMALLINT(2)	Minimum scale associated with this data type.
MAXIMUM_SCALE	SMALLINT(2)	Maximum scale associated with this data type.

ODBC_SCALES view

Table 6-12 lists the CATALOG.ODBC_SCALES view columns and their descriptions.

Table 6-12. ODBC_SCALES view columns

Column name	Data type	Description
CHAR_SCALE	CHAR(4)	
ODBC_SCALE	SMALLINT(2)	

CONST view

Table 6-13 lists the CATALOG.CONST view columns and their descriptions.

Table 6-13. CONST view columns

Column name	Data type	Description
DOUBLE_NULL	DOUBLE PRECISION(8)	
INTEGER_NULL	INTEGER(4)	
REAL_NULL	REAL(4)	
SMALLINT_NULL	SMALLINT(2)	
CHAR1_NULL	CHAR(1)	
CHAR12_NULL	CHAR(12)	

Table 6-13. CONST view columns (Continued)

Column name	Data type	Description
CHAR18_NULL	CHAR(18)	
CHAR128_NULL	CHAR(128)	
CHAR255_NULL	CHAR(255)	

ODBC_COLUMNS view

Table 6-14 lists the CATALOG.ODBC_COLUMNS view columns and their description. The ODBC_COLUMNS view contains a row of information for each discrete column defined to a catalogued table, and describes the type of data that can be held in that column.

Table 6-14. ODBC_COLUMNS view columns

Column name	Data type	Description
TABLE_QUALIFIER	CHAR(18)	Name of the catalog in which the column's table is catalogued (CCACAT)
TABLE_OWNER	CHAR(18)	Name of the owning schema (or ID) to which the column's table belongs
TABLE_NAME	CHAR(18)	Name of the table containing the column
COLUMN_NAME	CHAR(18)	Name of the column
DATA_TYPE	SMALLINT(2)	Integer code for the data type that is defined for this column (as described in the ODBC_TYPES view)
TYPE_NAME	CHAR(16)	Name of the data type used for this column
PRECISION_VAL	INTEGER(4)	Maximum number of significant digits available for values for this column
LENGTH	INTEGER(4)	Length in bytes of the data type used for this column
SCALE	CHAR(4)	Number of digits available to the right of the decimal point for values for this column
RADIX	SMALLINT(2)	The number base used for numerical values for this column. 10 indicates decimal values.
NULLABLE	SMALLINT(2)	1 means NULLS ALLOWED for this column. 0 means NULLS are not allowed.

Table 6-14. ODBC_COLUMNS view columns(Continued)

Column name	Data type	Description
REMARKS	CHAR(254)	General remarks on the column
TABLE_TYPE	CHAR(14)	Either TABLE for a base table, or VIEW, for a view defined against a table
SCOPE	SMALLINT(2)	
PRIMARY_KEY	CHAR(18)	The primary key for the column's table, if this table is a parent table
PARENT_TABLE_NAME	CHAR(18)	The parent table name, if the column's table is a nested table
SYSTEM_KEY	CHAR(1)	The name of the system generated key used as a unique identifier, if the column's table is a parent table.
NON_UNIQUE		
TYPE_CODE		
SEQ_IN_INDEX		

ODBC_TABLES view

Table 6-15 lists the CATALOG.ODBC_TABLES view columns and their descriptions. The ODBC_TABLES view contains a row of information for each discrete catalogued table or view.

Table 6-15.

Column name	Data type	Description
TABLE_QUALIFIER	CHAR(18)	Name of the catalog in which the table is catalogued (CCACAT)
TABLE_OWNER	CHAR(18)	Name of the owning schema (or ID) to which the table belongs
TABLE_NAME	CHAR(18)	Name of the table or view
TABLE_TYPE	CHAR(12)	Either TABLE for base table or VIEW for a view defined against a table
REMARKS	CHAR(254)	General notes on the table

ODBC_SPECIAL_COLS view

Table 6-16 lists the CATALOG.ODBC_SPECIAL_COLS view columns and their descriptions. The ODBC_SPECIAL_COLS view contains a row of information

for each discrete column that is used to uniquely identify rows within a table. This view may also be used to show other specialized columns in the future.

Table 6-16. ODBC_SPECIAL_COLS view columns

Column name	Data type	Description
TABLE_QUALIFIER	CHAR(18)	Name of the catalog in which the column's table is catalogued (CCACAT)
TABLE_OWNER	CHAR(18)	Name of the owning schema (or ID) to which the column's table belongs
TABLE_NAME	CHAR(18)	Name of the table containing the specialized column
SCOPE	SMALLINT(2)	
COLUMN_NAME	CHAR(18)	Name of the specialized column. Typically this will be the name of the primary key of a parent table
DATA_TYPE	SMALLINT(2)	Integer code for the data type that is defined for this column (as described in the ODBC_TYPES view).
TYPE_NAME	CHAR(16)	Name of the data type used for this column
PRECISION_VAL	INTEGER(4)	Maximum number of significant digits available for values for this column
LENGTH	INTEGER(4)	Length in bytes of the data type used for this column
SCALE	SMALLINT(2)	Number of digits available to the right of the decimal point for values for this column

ODBC_TABLE_STATS view

Table 6-17 lists the CATALOG.ODBC_TABLE_STATS view columns and their descriptions. The ODBC_TABLE_STATS view contains a row of information for each discrete catalogued table or view.

Table 6-17. ODBC_TABLE_STATS view columns

Column name	Data type	Description
TABLE_QUALIFIER	CHAR(18)	Name of the catalog in which the table is catalogued (CCACAT)
TABLE_OWNER	CHAR(18)	Name of the owning schema (or ID) to which the table belongs
TABLE_NAME	CHAR(18)	Name of the table or view

Table 6-17. ODBC_TABLE_STATS view columns (Continued)

Column name	Data type	Description
NON_UNIQUE	SMALLINT(2)	
INDEX_QUALIFIER	CHAR(18)	
INDEX_NAME	CHAR(18)	
TYPE_CODE	SMALLINT(2)	
SEQ_IN_INDEX	SMALLINT(2)	
COLUMN_NAME	CHAR(18)	
COLLATION	CHAR(1)	
CARDINALITY	INTEGER(4)	
PAGES	INTEGER(4)	

ODBC_KEY_STATS view

Table 6-18 lists the CATALOG.ODBC_KEY_STATS view columns and their descriptions.

Table 6-18.

Column name	Data type	Description
TABLE_QUALIFIER	CHAR(18)	Name of the catalog in which the table is cataloged (CCACAT)
TABLE_OWNER	CHAR(18)	Name of the owning schema (or ID) to which the table belongs
TABLE_NAME	CHAR(181)	Name of the table
NON_UNIQUE	SMALLINT(2)	1 means there is no unique identifier for row in the table. 0 means there is a unique identifier.
INDEX_QUALIFIER	CHAR(18)	
INDEX_NAME	CHAR(18)	Name of primary index for the table, if the table is a parent table
TYPE_CODE	SMALLINT(2)	Type of index
SEQ_IN_INDEX	SMALLINT(2)	
COLUMN_NAME	CHAR(18)	Name of column used as primary index for the table
COLLATION	CHAR(1)	Collation sequence for rows in the table. A = ascending D = descending ' ' = not applicable

Table 6-18. (Continued)

Column name	Data type	Description
CARDINALITY	INTEGER(4)	Number of rows in the table, or unique values in the primary index
PAGES	INTEGER(4)	Number of pages used to store the index or table.

ODBC_STATISTICS view

Table 6-19 lists the CATALOG.ODBC_STATISTICS view columns and their descriptions. The ODBC_STATISTICS view contains a row of information for each discrete catalogued table or view, and is used to return statistics about the table and its indexes.

Table 6-19.

Column name	Data type	Description
TABLE_QUALIFIER	CHAR(18)	Name of the catalog in which the table is catalogued (CCACAT)
TABLE_OWNER	CHAR(18)	Name of the owning schema (or ID) to which the table belongs
TABLE_NAME	CHAR(180)	Name of the table or view
NON_UNIQUE	SMALLINT(2)	1 means there is no unique identifier for row in the table or view. 0 means there is a unique identifier.
INDEX_QUALIFIER	CHAR(18)	
INDEX_NAME	CHAR(18)	Name of primary index for the table, if the table is a parent table
TYPE_CODE	SMALLINT(2)	Type of index
SEQ_IN_INDEX	SMALLINT(2)	
COLUMN_NAME	CHAR(18)	Name of column used as primary index for the table
COLLATION	CHAR(1)	Collation sequence for rows in the table. A = ascending D = descending ' ' = not applicable
CARDINALITY	INTEGER(4)	Number of rows in the table, or unique values in the primary index
PAGES	INTEGER(4)	Number of pages used to store the index or table.

7

Model 204 SQL Data Manipulation Language

In this chapter

- Overview
- Using Model 204 SQL DML
- Using SQL DML against INVISIBLE fields
- Using SQL DML against nested tables
- Options in the SELECT LIST statement
- SQL INNER JOIN features
- SQL OUTER JOIN features

Overview

An important goal of Model 204 SQL Data Manipulation Language (DML) is to match the form and function of the ANSI SQL 1989 standard and some of the ANSI SQL 1992 standard DML. Model 204 SQL extensions mainly affect DDL. You should not need to alter your existing SQL DML applications to access a Model 204 database.

This chapter describes the Model 204 SQL DML extensions as well as ways to use SQL DML to access data stored in Model 204 INVISIBLE fields, multiply occurring fields, and file groups.

Using Model 204 SQL DML

The Model 204 SQL Server supports all standard SQL DML. This section has a reminder about data definition accuracy and information about privileges for using SQL DML statements, setting SQL isolation level, and restrictions on interspersing SQL DML and DDL.

Maintaining data definition consistency

The SQL data definition information stored in the Model 204 SQL catalog for a Model 204 file is independent of the Model 204 Dictionary metadata for that file. It is also independent of and unchanged by any updates to the file's Table A data, that is, the file's fields and their attributes.

Caution: If you run a Model 204 SQL application against a Model 204 file, you are responsible for ensuring consistency between the Model 204 SQL catalog entries and the Model 204 file's Table A data. You must ensure that if the Model 204 file definition changes, the corresponding SQL definitions are updated. Reviewing the SQL catalog contents by direct query or with the Model 204 SQL catalog reporting utility (CCACATREPT) can help you to maintain the consistency between the SQL catalog and the Model 204 file.

DML statement privileges

To issue SQL DML query and update statements (SELECT, INSERT, DELETE, and UPDATE) against an SQL catalog object, the object must have your authorization ID or you must have been granted the privilege for the specified operation and for the specified object. These privileges are specified with SQL GRANT and REVOKE statements only.

For more information about GRANT and REVOKE and individual statement security, see Chapter 4.

Setting SQL isolation level

Model 204 SQL supports the following levels of record locking in its statement processing:

Isolation level 3 (Serializable)	SQL standard requires that concurrently executing statements be serializable, that is, the processing of the statements must yield the same results as if the statements were executed serially, one complete statement after another. Because it is costly to performance, the serializability option is not recommended.
Isolation level 1 (Cursor Stability)	A shared lock is obtained for a found set of records. After the records are read, the lock is dropped. This is the same as Model 204 User Language FIND statement processing.

Isolation level 0 (Dirty Read)	No locks are obtained for a found set of records. This is the same as Model 204 User Language FIND WITHOUT LOCKS statement processing.
-----------------------------------	----------------------------------------------------------------------------------------------------------------------------------------

Isolation level 0 is not recommended for applications using the following:

- SQL nested tables
- SQL views that are used for record security
- Model 204 reuse record number (RRN) files
- Model 204 files that are being simultaneously updated with User Language DELETE ALL RECORDS statements and SQL DML

SQL 32-bit Connect★ PC clients indicate the isolation level of statement processing by specifying an SQL Isolation Level option in the Model 204 ODBC Driver - Configuration Data Source dialog box or in the connection string for JDBC or .NET Framework connections.

Executing SQL DML and DDL simultaneously

Another user might issue SQL DDL against the SQL catalog at the same time that you are issuing SQL DML against the Model 204 database or to query the SQL catalog. However, the SQL DDL can update only SQL objects that are not being accessed by SQL DML statements.

Mixing SQL DML and DDL

The Model 204 SQL Server allows SQL DML statements interspersed with SQL DDL. An SQL DDL statement can follow an SQL DML statement only if the DML statement has first been committed. Model 204 SQL DDL statements result in an automatic Model 204 COMMIT.

- For JDBC and ODBC

When Autocommit, the default, is on, any DDL can follow an INSERT, UPDATE, or DELETE statement without an intervening commit or rollback. However, DDL after a SELECT statement without an intervening commit or rollback will result in the following SQL error:

```
SQL Error -7342. DML transaction in progress. DDL dis-
allowed until COMMIT/ROLLBACK, for EXECIMM, in EXECUTE
completion routine.
```

- For .NET Framework

DDL statements may not be issued while a transaction is active.

Using SET SCHEMA and SET USER

SET SCHEMA and SET USER are Model 204 SQL extensions that can be used with SQL DML or DDL statements.

SET SCHEMA allows any user to change the default schema name in SQL DDL or DML. Statements following SET SCHEMA are assumed to apply to this schema name. The SET SCHEMA syntax follows:

Syntax `SET SCHEMA schemaname`

For more information about SET SCHEMA, see “Altering SQL objects” on page 77.

SET USER allows a system manager to set or modify the current SQL authorization ID, in effect logging in as another SQL user without having to enter a password. SET USER therefore gives the system manager access to and authority to change all defined SQL objects.

The SET USER syntax follows:

Syntax `SET USER authorization-id`

For more information about SET USER, see “Altering SQL objects” on page 77.

Using SQL DML against INVISIBLE fields

Model 204 INVISIBLE fields are translated to SQL columns that have restricted functionality: you can use them to qualify searches for data, but they are not themselves retrievable. For example, you can use such columns in certain circumstances in the WHERE clause of a SELECT statement, but not in a SELECT list.

Note: Model 204 files that contain INVISIBLE fields not mapped to multicolumn unique constraints should be maintained by User Language or Host Language Interface applications and not by SQL applications. This recommendation applies regardless of whether the fields are mapped to SQL columns.

Using SQL columns mapped to INVISIBLE fields

The restrictions on SQL DML operations against columns mapped to INVISIBLE fields are listed below for each DML statement. In this section, an SQL column mapped to an INVISIBLE field is called an *invisible column*.

SELECT statement

You cannot use an invisible column in the SELECT list.

If there is an invisible column in a table against which a SELECT * operates, the column is eliminated from the statement output, and you receive no warning

message. If the SELECT * is in an INSERT query, or in a query involving UNION, the statement is rejected.

In the following examples, the PARTS table has invisible column INV_S; the SUPPLIERS table has invisible column INV_IN. This SELECT fails, because you cannot use an invisible column in the SELECT list:

```
SELECT PNO, INV_S FROM PARTS WHERE PNO=1234
```

These SELECT statements are acceptable, but you cannot connect them with UNION:

```
SELECT * FROM PARTS WHERE PNO='P1'
```

```
SELECT * FROM PARTS WHERE COLOR='Red'
```

The use of invisible columns in a SELECT statement WHERE clause is limited. See “WHERE clause” on page 166.

UPDATE statement

You cannot use an invisible column as the target of or in the source expression of an UPDATE.

For example, this UPDATE fails:

```
UPDATE PARTS SET INV_S='Red' WHERE PNO=1234
```

The use of invisible columns in an UPDATE statement WHERE clause is limited. See the section “ORDER BY clause” on page 166.

INSERT statement

You can use an invisible column as the target column.

For example, you can issue this statement:

```
INSERT INTO PARTS (PNO,PNAME,INV_S)
VALUES ('123','Widget','Red')
```

Any SELECT query involved is subject to the limitations of the SELECT statement (see “SELECT statement” on page 164).

DELETE statement

If you DELETE a row that contains an invisible column value, the invisible column value is not deleted.

The use of invisible columns in a DELETE statement WHERE clause is limited. See “ORDER BY clause” on page 166.

GROUP BY clause

You cannot use an invisible column as a column in a GROUP BY clause.

HAVING clause

You cannot use an invisible column in a HAVING clause unless the invisible column is contained in a WHERE clause of a subquery contained in the HAVING clause, in which case the use of the column is subject to the WHERE clause rules.

ORDER BY clause

You cannot use an invisible column as a column in an ORDER BY clause.

WHERE clause

The restrictions on the use of invisible columns in WHERE clauses of SQL DML statements are not easily generalized. Most cases are covered by the following rules. The remaining more complicated rules are listed on “Additional restrictions on WHERE clause comparisons” on page 167.

The use of invisible columns in WHERE clauses of SQL SELECT, INSERT, UPDATE, and DELETE statements is subject to the following restrictions:

- You cannot use invisible columns if the WHERE clause is part of a view definition that includes the WITH CHECK OPTION.
- You cannot use invisible columns in arithmetic expressions.

For example, the following statement fails:

```
SELECT SNO, STCODE FROM SUPPLIERS
WHERE (STCODE = 20 * (INV_IN + 300) OR SNAME='NUT')
```

- In addition to the basic comparisons (<, <=, =, >, >=, ^=) you can use an invisible column with the following SQL operators:

```
LIKE
BETWEEN
IN (but not the subquery form)
```

For example, the following statement fails (subquery form of IN):

```
SELECT PNO FROM PARTS WHERE INV_S IN
(SELECT PNAME FROM PARTS P WHERE P.PNO='S1')
```

- If the INVISIBLE field has the KEY attribute, comparisons using the following operators are not allowed:

```
<
<=
>
```

```
>=
LIKE
BETWEEN
```

- You can compare an invisible column to only the following:
 - Literal
For example:

```
SELECT * FROM SUPPLIERS WHERE INV_IN > 20
```
 - Parameter
For example, where ? is a parameter marker whose values are substituted by a program at execution time:

```
SELECT SNO, SNAME FROM SUPPLIERS WHERE INV_IN=? AND SNO=?
```
 - An outer reference, in which a WHERE clause in a subquery refers to a value from an outer query
For example:

```
SELECT SNAME FROM S
WHERE 'P2' IN
( SELECT INV_IN FROM SP
  WHERE INV_IN=S.SNO )
```
 - An expression of literals or parameters or outer references
For example:

```
SELECT * FROM SUPPLIERS
WHERE INV_IN = 20 * (? + 300) AND SNO='S1'
```

Additional restrictions on WHERE clause comparisons

The restrictions on the use of invisible columns in WHERE clauses include the following additions to the rules for comparisons described previously:

- You can compare an invisible column to an uncorrelated (no outer references) subquery, except the following types of quantified subqueries:

```
=ANY
=SOME
IN
^=ANY
^=SOME
```

For example, the following statement is valid:

```
SELECT PNO FROM PARTS WHERE INV_S > ALL
( SELECT PNAME FROM PARTS P WHERE P.PNO='S1' )
```

- If the invisible column is NOT NULL, comparisons with ^= or NOT LIKE are restricted: the value compared to the invisible column cannot include parameters, cannot be a subquery, and cannot include outer references if these might yield the null value.
- If the invisible column is not NOT NULL, comparisons with ^= or NOT LIKE are prohibited.
- If the WHERE clause is broken out into one or more comparisons connected by OR (disjuncts), with NOTs factored down to the individual comparison, each disjunct containing an invisible column comparison can contain only comparisons (invisible or visible) that refer to the same parent table and that satisfy all the preceding WHERE clause rules.

Using SQL DML against nested tables

The Model 204 SQL nested table design is an extension to standard SQL that enables an SQL application to access Model 204 multiply occurring fields and groups. Using SQL DML against Model 204 SQL nested tables is essentially the same as using DML against typical nonnested SQL tables.

Sample file and SQL mapping

The DML examples in this section are based on the same sample file and SQL catalog mapping used in the discussion of nested table DDL in Chapter 3.

A Model 204 file has the following fields:

Field	Frequency of occurrence
NAME	Once per record
HIRE_DATE	Once per record
REV_DATE	Multiple times per record
SALARY	Multiple times per record
TITLE	Multiple times per record
TASK	Multiple times per record

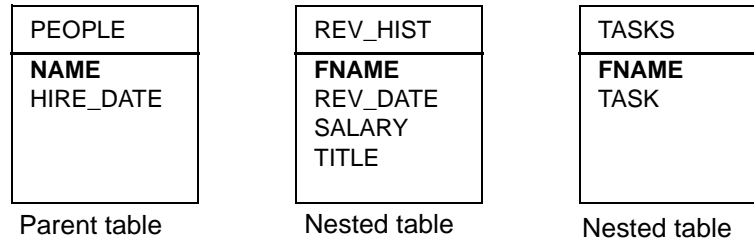
REV_DATE, SALARY, and TITLE are a repeating group that occurs once each salary review.

For the purposes of the example series, we consider the following cases:

- NAME is a unique identifier and is mapped to the primary key column of a parent table
- The NAME field is a unique identifier and is used as the primary key in the parent table

Case 1

The file is mapped to one SQL parent table (PEOPLE) with two columns (NAME and HIRE_DATE) and two nested tables (REV_HIST and TASKS):



The parent table is linked to the nested tables by the common values of the primary key NAME in the parent and the foreign key FNAME in the nested tables. The following DDL provides this mapping:

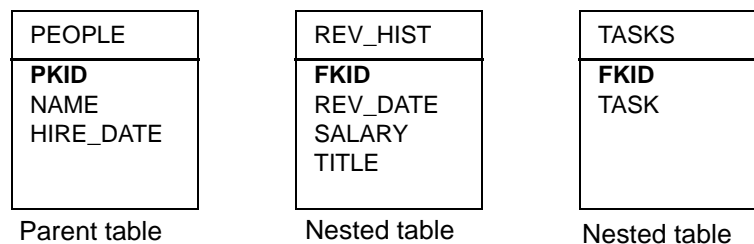
```
CREATE TABLE PEOPLE
( NAME CHAR(60) NOT NULL PRIMARY KEY,
  HIRE_DATE CHAR(8) )

CREATE TABLE REV_HIST NESTED USING FNAME
( DATE INTEGER NOT NULL,
  SALARY DECIMAL (11,2) NOT NULL,
  TITLE CHAR (50) NOT NULL,
  FNAME CHAR (60) NOT NULL REFERENCES PEOPLE )

CREATE TABLE TASKS NESTED USING FNAME
( TASK CHAR (25) NOT NULL,
  FNAME CHAR (60) NOT NULL REFERENCES PEOPLE )
```

Case 2

There is no unique identifier, so a system-generated key (PKID) is the parent table primary key. The file is mapped to one SQL parent table (PEOPLE) with three columns (PKID, NAME, and HIRE_DATE) and two nested tables (REV_HIST and TASKS):



The parent table is linked to the nested tables by the common values of the primary key PKID in the parent and the foreign key FKID in the nested tables.

The names PKID and FKID are user-chosen. The following DDL provides this mapping:

```
CREATE TABLE PEOPLE
( PKID INTEGER NOT NULL PRIMARY KEY SYSTEM,
  NAME CHAR(60) NOT NULL,
  HIRE_DATE CHAR(8) )

CREATE TABLE REV_HIST NESTED USING FKID
( DATE INTEGER NOT NULL,
  SALARY DECIMAL (11,2) NOT NULL,
  TITLE CHAR (50) NOT NULL,
  FKID INTEGER NOT NULL REFERENCES PEOPLE )

CREATE TABLE TASKS NESTED USING FKID
( TASK CHAR (25) NOT NULL,
  FKID INTEGER NOT NULL REFERENCES PEOPLE )
```

For more information about system-generated keys, see “Using system-generated keys” on page 67.

DML example series

The examples in the series are presented by database operation and usually have a sample query specification for Case 1 (primary key maps to database field) and Case 2 (primary key is system-generated). The query specifications represent SQL cursor-based SELECT statements.

Retrieving a particular occurrence of a multiply occurring group

The task is to select a particular occurrence of the repeating group of field values that are mapped to the REV_HIST nested table. Each occurrence of the group is a set of related values (one value from each of the members of the group). The nested table definition in the example in this section maps each row in REV_HIST to a particular occurrence of the group. Therefore, retrieving a particular occurrence of the group translates to selecting a particular row from REV_HIST.

Case 1

```
SELECT REV_DATE, SALARY, TITLE
FROM   REV_HIST
WHERE  FNAME='JOHN SLOWFOOT' AND REV_DATE BETWEEN 19900101
      AND 19900331
```

Case 2

The system-generated key (PEOPLE) qualifies the selection:


```
SELECT REV_DATE, SALARY, TITLE
FROM   REV_HIST, PEOPLE
WHERE  NAME='JOHN SLOWFOOT' AND REV_DATE BETWEEN 19900101
      AND 19900331 AND PKID = FKID
```

Retrieving a range or series of occurrences

The following example selects only some of the values per occurrence of the group over a range of occurrences: the salaries of people who were senior secretaries from 1989 through 1991.

Case 1

```
SELECT FNAME, SALARY
FROM   REV_HIST
WHERE  REV_DATE BETWEEN 19890101 AND 19911231 AND
       TITLE='SR SECTY'
```

Case 2

```
SELECT NAME, SALARY
FROM   REV_HIST, PEOPLE
WHERE  REV_DATE BETWEEN 19890101 AND 19911231 AND
       TITLE='SR SECTY'
       AND PKID = FKID
```

Retrieving any or all occurrences based on a condition

The following example selects the names of those for whom all occurrences meet the condition, that is, all whose salary values are above 40,000. You can replace ALL with ANY in the example and therefore select the names of those who have any of their salary values above 40,000.

Case 1

```
SELECT NAME
FROM   PEOPLE
WHERE  40000 < ALL (SELECT SALARY FROM REV_HIST
                   WHERE FNAME = NAME)
```

Case 2

```
SELECT NAME
FROM   PEOPLE
WHERE  40000 < ALL (SELECT SALARY FROM REV_HIST
                   WHERE PKID = FKID)
```

Retrieving at least *n* occurrences based on a condition

The following example selects the names of those who have any of their salary values above 40,000 only if at least three names qualify.

Case 1

```
SELECT NAME
FROM   PEOPLE
WHERE  2 < (SELECT COUNT (*) FROM REV_HIST WHERE FNAME =
           NAME AND 40000 < SALARY)
```

Case 2

```

SELECT NAME
FROM   PEOPLE
WHERE  2 < (SELECT COUNT (*) FROM REV_HIST WHERE
           PKID = FKID AND 40000 < SALARY)

```

Correlating a table and a nested table

The examples in this section mix column selections from a nested table and its parent table and use them for output like the following, where the NAME and HIRE_DATE values for entries in the PEOPLE table are reported along with their TASK values from the nested table TASKS:

```

Maria Pena
July 9, 1988
    New prod plan
    Budget

```

```

John Slowfoot
November 22, 1989
    Project A
    Project B

```

The method of deriving this output is fetching with a join query.

The following query selects from the nested table TASKS and its parent PEOPLE. The ORDER BY clause guarantees the ordering of the data.

Case 1

```

SELECT  NAME, HIRE_DATE, TASK
FROM    PEOPLE, TASKS
WHERE   NAME = FNAME
ORDER BY NAME, TASK

```

Case 2

```

SELECT  NAME, HIRE_DATE, TASK
FROM    PEOPLE, TASKS
WHERE   PKID = FKID
ORDER BY PKID, TASK

```

Working with nested table constraints

Referential constraint checking is done per row.

The NOT NULL constraint, which prevents SQL applications from updates that introduce null values into an SQL nested table column, is enforced only for SQL. A User Language application is not prevented from introducing nulls into

the Model 204 fields mapped to SQL nested columns, although this introduction is likely to invalidate queries involving these fields. Such an invalidation is an example of why you must be careful, especially with nested tables, when you use both SQL and User Language to maintain file data.

Porting nested table applications

A Model 204 SQL nested table application is readily portable to other environments if the application does not have system-generated primary keys in the parent and if the environment ported to supports CASCADE for updates and deletes.

Options in the SELECT LIST statement

Correlation name feature

The name in the SELECT LIST statement can be assigned to a new name by either:

- AS keyword
- Equal sign (=) sign
- A white space.

If the name represents a column, the newly assigned name may be referred to in other clauses. The following examples are supported:

```
SELECT COUNT(*) AS FILECOUNT FROM TABLE5

SELECT T1.TITLE TITLE, CHAPTER = T2.TITLE
      FROM BOOK T1, CHAPTERS T2
      WHERE TITLE = 'WWII' AND CHAPTER = 'US'
```

The following example is not supported:

```
SELECT DOUBLE_ADVANCE = (ADVANCE * 2)
      FROM PUBLISHERS
      WHERE DOUBLE_ADVANCE > 1000
```

Note: Correlation names may not carry over to a query at a different level of an SQL statement.

Wildcard asterisk (*) for an individual table feature

```
SELECT T.TITLE, T.*, P.*
      FROM TITLES T, PUBLISHERS P
      WHERE T.ID = P.ID
```

CURRENT_TIME keyword

The `CURRENT_TIME` keyword is assigned as `CHAR(11)` in the form of *hh:mm:ss.tt*, representing the local time at the time of the query execution.

```
SELECT CURRENT_TIME, COUNT(*) FROM TABLE5
```

```
10:05:24.38
```

Note: In this example query and those to follow, the `COUNT(*)` is added merely for the purpose of generating a single row of output.

CURRENT_DATE keyword

The `CURRENT_DATE` keyword is assigned as `CHAR(10)` in the form of *yyyy-mm-dd* to represent the date at the time of the query execution.

```
SELECT CURRENT_DATE, COUNT(*) FROM TABLE5
```

```
2001-03-08
```

CURRENT_TIMESTAMP keyword

The `CURRENT_TIMESTAMP` keyword is assigned as `CHAR(22)` in the form of *yyyy-mm-dd hh:mm:ss.tt*, the time and the date at the time of the query execution.

```
SELECT CURRENT_TIMESTAMP, COUNT(*) FROM TABLE5
```

```
2001-03-08 10:05:24.38
```

USER keyword

The `USER` keyword is assigned as `CHAR(10)`. It represents the name of the user who logged on to the thread.

```
SELECT USER, COUNT(*) FROM TABLE5
```

```
USER101
```

SQLVERSION keyword

The `SQLVERSION` keyword is assigned as `CHAR(16)`. It represents the version code of the RSQL module and the released date in the form: *n.n.na mm/dd/yy*

```
SELECT SQLVERSION, COUNT(*) FROM TABLE5
```

```
7.1.0C 05/30/09
```

This keyword is intended for diagnostic purposes.

SQLERROR keyword

The SQLERROR keyword is assigned as CHAR(5). It represents the last error code detected and recorded in the RSQL module. Six locations are reserved to store the last six error codes. For example:

```
SELECT SQLERROR, SQLERROR, COUNT(*) FROM TABLES  
  
+0000, -0206
```

The SQLERROR keyword is intended for diagnostic purposes.

SQLSTATE keyword

The SQLSTATE keyword is assigned as CHAR(32). It represents the significant part of the last error message or the object name which caused the error, as recorded in the RSQL module. For example, SQLSTATE will contain 'UNITCOST', if the previous error was:

```
SQL Error -408. A value is not compatible with its object  
column UNITCOST, for EXECIMM, in EXECUTE completion rou-  
tine
```

The SQLSTATE keyword is intended for diagnostic purposes.

SQL INNER JOIN features

CROSS JOIN feature

The CROSS JOIN feature produces a cross-product of a table join in which no join condition is specified. For example:

```
SELECT * FROM TITLES CROSS JOIN PUBLISHERS
```

The previous query is equivalent to the following example:

```
SELECT * FROM TITLES, PUBLISHERS
```

NATURAL JOIN feature

The NATURAL JOIN feature combines two tables on all columns that share the same names. For example, if PUB_ID and AUTHOR are the only two columns which are common to both tables, then the following query with the NATURAL JOIN clause produces the same results as the query with the WHERE clause:

```
SELECT * FROM TITLES NATURAL JOIN PUBLISHERS  
  
SELECT * FROM TITLES, PUBLISHERS
```

```
WHERE TITLES.PUB_ID = PUBLISHERS.PUB_ID
      AND TITLES.AUTHOR = PUBLISHERS.AUTHOR
```

ON clause feature

The conditional join uses an ON clause to impose the join condition. It is interchangeable with the WHERE clause. For example, the following query uses an ON clause:

```
SELECT * FROM TITLES JOIN PUBLISHERS
      ON TITLES.PUB_ID = PUBLISHERS.PUB_ID
```

The previous query is equivalent to the following query which employs a WHERE clause:

```
SELECT * FROM TITLES, PUBLISHERS
      WHERE TITLES.PUB_ID = PUBLISHERS.PUB_ID
```

USING clause feature

The NATURAL JOIN feature matches all column names in one table with the same column names in a second table. If only some column names in one table match those column names in a second table, you can explicitly specify those columns with the USING clause, as follows:

```
SELECT * FROM TITLES JOIN PUBLISHERS
      USING (PUB_ID, AUTHOR)
```

The previous query is equivalent to a query in which the USING keyword is replaced by the ON clause, such as

```
SELECT * FROM TITLES, PUBLISHERS
      ON (TITLES.PUB_ID = PUBLISHERS.PUB_ID
          AND TITLES.AUTHOR = PUBLISHERS.AUTHOR)
```

Note: The ON clause is *not* allowed to contain a subquery.

SQL OUTER JOIN features

Unlike the INNER JOIN feature, the OUTER JOIN feature preserves the unmatched rows from one of the two tables, depending on the keywords LEFT and RIGHT.

The following two equivalent queries are examples of a LEFT OUTER JOIN clause.

```
SELECT * FROM TITLES LEFT OUTER JOIN PUBLISHERS
      USING (PUB_ID, AUTHOR)
```

```
SELECT * FROM TITLES LEFT OUTER JOIN PUBLISHERS
      ON (TITLES.PUB_ID = PUBLISHERS.PUB_ID
```

```
AND TITLES.AUTHOR = PUBLISHERS.AUTHOR)
```

Note: The keyword OUTER in these queries is optional.

In these examples, each row in the first table, TITLES, in a LEFT OUTER JOIN clause must be included in the result. If no matching value is found in the second table, PUBLISHERS, the corresponding columns of PUBLISHERS are filled with NULLs.

The RIGHT OUTER JOIN clause operates similarly to a LEFT OUTER JOIN clause, except that the RIGHT or the second table is the parent table and its rows are preserved. For example, the previous two LEFT OUTER JOIN queries can be expressed as RIGHT OUTER JOIN queries.

```
SELECT * FROM PUBLISHERS RIGHT OUTER JOIN TITLES
      USING (PUB_ID, AUTHOR)
```

```
SELECT * FROM PUBLISHERS RIGHT OUTER JOIN TITLES
      ON (TITLES.PUB_ID = PUBLISHERS.PUB_ID
          AND TITLES.AUTHOR = PUBLISHERS.AUTHOR)
```

Note: The ON clause is *not* allowed to contain a subquery.

SQL outer join features expanded

SQL join expressions with join types LEFT, RIGHT, CROSS, and UNION are supported where any table can be a base or a nested table or a viewed table.

- Usage Notes**
- In a query a Left Outer Join and Right Outer Join can be combined in any order.
 - In a view definition a Left Outer Join and Right Outer Join can be combined in any order.
 - A nested table can be used in any place where a base table can be used.
 - A view can be used in any place where a base table can be used,
 - The ON clause can contain any columns from the joined tables.
 - A subquery can contain a subquery.

In the following example Join can be either a Left or Right join.

SQL table expression T1 Join T2 ON Pred1 Join T3 ON Pred2 Join T4 ON Pred3

Model 204 evaluates the previous code in the following sequence of steps:

1. Table T1 joined with Table T2 by ON Pred1
2. The expression (T1 Join T2 ON Pred1) is joined with T3 by ON Pred2
3. The expression ((T1 Join T2 ON Pred1) Join T3 ON Pred2) is joined with

T4 by ON Pred3

A

Model 204 SQL DDL Syntax

In this appendix

- Overview
- DDL syntax
- Notes for syntax display

Overview

This appendix outlines in complete detail the syntax of Model 204 SQL DDL. Consult the American National Standards *ANSI X3.135-1989 Database Language SQL* document for syntax details not outlined here. The notation conventions used in this appendix are listed at the end of the Preface of this document.

The following rules apply to this syntax:

1. SQL object names (that is, schema name, table name, and so on) conform to rules for an SQL identifier: a name must be 18 characters or less and it cannot be identical to an SQL keyword (the list of keywords is detailed in the standard). Model 204 SQL Reserved Words are listed in Appendix B.
2. End of line (newline) is implementation defined.
Note: The end of line character for use with DVI is the semicolon (;).
3. Numbers in parentheses to the right of some of the syntax lines are keys to notes that appear at the end of the syntax diagrams.

4. Model 204 SQL DDL extensions to the standard are printed in ***bold italics***; statements or clauses that are not part of the ANSI SQL 1989 standard or the ANSI SQL 1992 standard, but that are anticipated to be part of the emerging standard are printed in *italics*.

DDL syntax

```

tablename ::=
    [schemaname.] <table>

table ::=
    basetable | viewname

schema ::=
    CREATE SCHEMA <schema-authorization-clause>
    [ schema-element • • • ]

schema-authorization-clause ::=
    schemaname

1 | AUTHORIZATION authorization-id

1 | schemaname AUTHORIZATION authorization-id

schema-element ::=
    <table-definition>
    | <view-definition>
    | <privilege-definition>

table-definition ::=
    CREATE TABLE <tablename>
    [ <file-mapping-clause> | NESTED USING <nested-key> ]
    ( <table-element> ,••• )

file-mapping-clause ::=
    SYSNAME 'filename'

nested-key ::=
    columnname

table-element ::=
    <column-definition>
    | <table-constraint-definition>

2 column-definition ::=
    columnname <datatype>
    [ <field-mapping-clause> ]
    [ <column-constraint> ••• ]

datatype ::=

```

- 3 CHAR[ACTER] [(length)]
- 4 | NUM[ERIC] [(precision [,scale])]
 | DEC[IMAL] [(precision [,scale])]
 | INT[EGER]
 | SMALLINT
- 5 | FLOAT [(precision)]
 | REAL
 | DOUBLE PRECISION
 | BLOB
 | CLOB
- field-mapping-clause* ::=
 SYSNAME 'fieldname'
- 6 column-constraint ::=
 NOT NULL
- 7 | <unique-specification>
 | <references-specification>
- table-constraint-definition ::=
 <unique-constraint-definition>
 | <referential-constraint-definition>
- unique-constraint-definition ::=
 <unique-specification> (<column-list>)
- 8 [SYSNAME 'fieldname']
- unique-specification ::=
 UNIQUE | PRIMARY KEY [**SYSTEM**]
- 9 referential-constraint-definition ::=
 FOREIGN KEY (columnname)
 <references-specification>
- references-specification ::=
- 9 REFERENCES **parent-table-name**
 [<referential-triggered-action>]
- referential-triggered-action ::=
 <update-rule> [<delete-rule>]
 | <delete-rule> [<update-rule>]
- update-rule ::= ON UPDATE CASCADE

```

delete-rule ::= ON DELETE CASCADE

view-definition ::=
    CREATE VIEW <viewname> [( <column-list> )]

10      AS <query-expression> [WITH CHECK OPTION]

set user statement ::=

11      SET USER authorization-id

set schema statement ::=
    SET SCHEMA schemaname

drop schema statement ::=

12      DROP SCHEMA schemaname

drop table statement ::=

12      DROP TABLE <tablename>

drop view statement ::=

12      DROP VIEW <viewname>

alter table statement ::=
    ALTER TABLE <tablename> <alter-table-action>

alter-table-action ::=

13      ADD <column-definition>
        | DROP columnname
        | MODIFY <column-parameters>

column-parameters ::=

14      MODIFY columnname
        [<datatype>]
        [<field-mapping-clause>]
        [<modify-attribute-list> ...]

modify-attribute-list ::=
    [ NOT ] NULL
    | [ NOT ] UNIQUE

privilege-definition ::=
    GRANT <privileges> ON <object-name>
    TO <grantee> , ...

```

```

[ WITH GRANT OPTION ]

privileges ::=
    ALL PRIVILEGES
    | <action> ,...

action ::=
    SELECT
    | INSERT
    | DELETE

15    | UPDATE [( <column-list> )]

column list ::=
    columnname, ...

object-name ::=
    <tablename> | <viewname>

grantee ::=
    PUBLIC
    | authorization-id

revoke statement ::=
    REVOKE [ GRANT OPTION FOR ] <privileges>
    ON <object-name>
    FROM <grantee> , ...

```

Notes for syntax display

The numbers for the comments below correspond to the numbers in **bold** in the preceding syntax listing.

1. Since the AUTHORIZATION value maps to a Model 204 user ID, it can be no longer than 10 characters.
2. The column definition DEFAULT clause is not supported in Model 204 SQL.
3. Referred to as *character string* type in the standard. The default length is 1.
4. NUMERIC, DECIMAL, INTEGER, and SMALLINT are referred to as *exact numeric* types in the standard. The precision values are interpreted as decimal precision.
5. FLOAT, REAL, and DOUBLE PRECISION are referred to as *approximate numeric* types in the standard. The precision value is interpreted as binary precision.
6. CHECK column constraint is not supported in Model 204 SQL.
7. Unlike the standard, UNIQUE is independent of NOT NULL. If you specify

UNIQUE, NOT NULL is not implied.

In addition, PRIMARY KEY is syntactically independent of NOT NULL.

Specifying PRIMARY KEY without NOT NULL is not a syntax error.

However, regardless of whether you specify NOT NULL, when you specify PRIMARY KEY, the SQL Server includes NOT NULL checking by default.

8. If the SYSNAME extension is omitted in a multicolumn unique key definition, by default the Model 204 SQL Server assumes a concatenation of the constituent column names, separating the names with an ampersand (&) character. For example, the default concatenation of (SSN, AGE, NAME) is SSN&AGE&NAME.
9. REFERENCES and the referential constraint definition apply only to nested tables. Specification of a REFERENCES clause in a context other than for a nested table produces a warning message.
10. A query expression is a query specification or a UNION of query specifications, where query specification is a SELECT statement (with no ORDER BY clause).
11. Modifies the current SQL authorization ID. System manager privileges are required.
12. DROP SCHEMA physically deletes from CCACAT the definition of all tables, views, and associated grant statements.
DROP TABLE deletes the table definition record from CCACAT and deletes privileges and constraints associated with the table.
DROP VIEW deletes any subordinate views (that is, VIEWS of VIEWS) from CCACAT as well as privileges granted for the view.
13. All columns are added to a table in the last position of the column list. If you DROP a column and then ADD an updated version of that column, the updated column might occupy a different position in the table than it did before you issued DROP and ADD.

Such a change in order of the column data can introduce errors into queries that use SELECT * or INSERT (without a column list) and that depend on the correct position of the column data. Use MODIFY for changes to column definitions other than deletions and additions.
14. The MODIFY *columnname* clause supports all aspects of column definition (see discussion in the previous note). Specify only the attributes being modified. You also can modify the field mapping clause and data type.
15. Unlike the standard, no REFERENCES privileges are supported.

B

Model 204 SQL Reserved Words

In this appendix

- Overview
- Reserved words

Overview

This appendix lists the Model 204 SQL reserved words, which are words that you cannot use as names of SQL objects. This list includes standard and emerging standard SQL reserved words and Model 204 SQL proprietary reserved words (in italics).

Reserved words

ADD	DROP	MAX	SMALLINT
ALL	END	MIN	SOME
ALTER	ESCAPE	MODIFY	SORT
AND	EXEC	MODULE	SQL
ANY	EXISTS		SQLCODE
AS		NATURAL	SQLERROR
ASC	FETCH	<i>NESTED</i>	SQLSTATE
AUTHORIZATION	FLOAT	NOT	SQLVERSION
AVG	FOR	NULL	SUM
	FOREIGN	NUM	<i>SYSNAME</i>
BEGIN	FORTRAN	NUMERIC	<i>SYSTEM</i>
BETWEEN	FOUND		
BLOB	FROM	OF	TABLE
BY	FULL	ON	TO
		OPEN	TYPE
CASCADE	GO	OPTION	
CAST	GOTO	OR	<i>UNCATALOG</i>
CHAR	GRANT	ORDER	UNION
CHARACTER	GROUP	<i>ORDERED</i>	UNIQUE
CHECK		OUTER	UPDATE
CLOB	<i>HASH</i>		USER
CLOSE	HAVING	PASCAL	<i>USING</i>
COBOL		PLI	
COMMIT	IN	POSITIONED	VALUES
CONTINUE	<i>INDEX</i>	PRECISION	VIEW
COUNT	INDICATOR	PRIMARY	
CREATE	INNER	PRIVILEGES	WHENEVER
CROSS	INSERT	PROCEDURE	WHERE
CURRENT	INT	PUBLIC	WITH
CURRENT_DATE	INTEGER		WORK
CURRENT_TIME	INTO	REAL	
CURRENT_TIMESTAMP	IS	REFERENCES	
CURSOR		RESTRICT	GE
	JOIN	REVOKE	LE
DEC		RIGHT	NE
DECIMAL	KEY	ROLLBACK	
DECLARE			
DEFAULT	LANGUAGE	SCHEMA	
DELETE	LEFT	SECTION	
DESC	LIKE	SELECT	
DISTINCT		SET	
DOUBLE			

C

SQL DDL Mapping of the Demonstration Database

In this appendix

Overview

DDL stream

Overview

This section displays DDL that maps the Model 204 demonstration database files to Model 204 SQL tables. This sample DDL also includes two views of one of the tables.

If you use this mapping at your site for test purposes, please note the following:

- This DDL maps to the Version 6.1 demonstration database files. Ensure that you have the correct set of files attached to your Online.
- You must supply a value for the placeholder xxxx in the DDL below (for SCHEMA NAME).

After this cataloging of the demonstration database files, the query:

```
SELECT RECTYPE, FULLNAME, SEX, STATE FROM CLIENTS
WHERE POLICY_NO = 100648
```

results in the following:

DRIVER	BALDWIN, LEE D	M	CAL-
--------	----------------	---	------

IFORNIA			
DRIVER	BALDWIN, MARY C	F	CALIFORNIA
POLICYHOLDER	BALDWIN, MARY C		CALIFORNIA

DDL stream

CLIENTS table

```

CREATE SCHEMA xxxx;
CREATE TABLE CLIENTS
( ADDRESS
  CHAR(40),
  AGENT
  CHAR(20),
  ANNIV_DATE
  SYSNAME 'ANNIV DATE'
  INTEGER,
  CITY
  CHAR(20),
  DATE_OF_BIRTH
  SYSNAME 'DATE OF BIRTH'
  INTEGER,
  DRIVER_ID
  SYSNAME 'DRIVER ID'
  INTEGER,
  FULLNAME
  CHAR(40) NOT NULL,
  MARITAL_STATUS
  SYSNAME 'MARITAL STATUS'
  CHAR(15),
  POLICY_NO
  SYSNAME 'POLICY NO'
  CHAR(6) NOT NULL,
  POLICYHOLDER
  CHAR(40),
  RECTYPE
  CHAR(15) NOT NULL,
  RESTRICTIONS
  CHAR(255),
  SEX
  CHAR(1),
  STATE
  CHAR(25),
  TOTAL_PREMIUM
  SYSNAME 'TOTAL PREMIUM'
  INTEGER,
  ZIP
  CHAR(9),

```

```

        PID INTEGER NOT NULL PRIMARY KEY SYSTEM );
GRANT ALL PRIVILEGES ON CLIENTS TO PUBLIC;

```

ACCIDENTS table, nested under CLIENTS

```

CREATE TABLE ACCIDENTS NESTED USING PID
( INCIDENT
  CHAR(2) NOT NULL,
  INCIDENT_DATE
  SYSNAME 'INCIDENT DATE'
  INTEGER NOT NULL,
  PID INTEGER NOT NULL REFERENCES CLIENTS );
GRANT ALL PRIVILEGES ON ACCIDENTS TO PUBLIC;

```

INSURED-VINS table, nested under CLIENTS

```

CREATE TABLE INSURED_VINS NESTED USING PID
( VIN
  CHAR(12) NOT NULL,
  PID INTEGER NOT NULL REFERENCES CLIENTS );
GRANT ALL PRIVILEGES ON INSURED_VINS TO PUBLIC;

```

VEHICLES table

```

CREATE TABLE VEHICLES
(BODY
  CHAR(4),
  COLLISION_PREMIUM
  SYSNAME 'COLLISION PREMIUM'
  INTEGER,
  COLOR
  CHAR(15),
  DEDUCTIBLE
  DECIMAL(3,0),
  GARAGING_LOCATION
  SYSNAME 'GARAGING LOCATION'
  CHAR(4),
  LIABILITY_LIMIT
  SYSNAME 'LIABILITY LIMIT'
  DECIMAL(4,0),
  LIABILITY_PREMIUM
  SYSNAME 'LIABILITY PREMIUM'
  INTEGER,
  MAKE
  CHAR(20),
  MODEL
  CHAR(20),
  OWNER_POLICY
  SYSNAME 'OWNER POLICY'

```

```

        CHAR(6),
    PRINCIPAL_DRIVER
        SYSNAME 'PRINCIPLE DRIVER'
        INTEGER,
    SURCHARGE
        SYSNAME 'SURCHARGE%'
        CHAR(2),
    TRANS
        CHAR(2),
    USAGE
        CHAR(60),
    VEHICLE_PREMIUM
        SYSNAME 'VEHICLE PREMIUM'
        INTEGER,
    VEHICLE_RATING
        SYSNAME 'VEHICLE RATING'
        CHAR(1),
    VEHICLE_USE_CLASS
        SYSNAME 'VEHICLE USE CLASS'
        CHAR(2),
    VIN
        CHAR(10) NOT NULL PRIMARY KEY,
    YEAR
        DECIMAL(4,0));
GRANT ALL PRIVILEGES ON VEHICLES TO PUBLIC;

```

OTHER_DRIVER table, nested under VEHICLES

```

CREATE TABLE OTHER_DRIVER NESTED USING VIN
    (OTHER_DRIVER
        SYSNAME 'OTHER DRIVER'
        CHAR(6) NOT NULL,
    VIN
        CHAR(10) NOT NULL UNIQUE REFERENCES VEHICLES);
GRANT ALL PRIVILEGES ON OTHER_DRIVER TO PUBLIC;

```

CLAIMS03 table

```

CREATE TABLE CLAIMS03
    (CLAIM_NO
        SYSNAME 'CLAIM NO'
        INTEGER NOT NULL UNIQUE,
    CLAIM_STATUS
        SYSNAME 'CLAIM STATUS'
        CHAR(15),
    CLAIM_TYPE
        SYSNAME 'CLAIM TYPE'
        CHAR(1),
    CLAIMEE

```

```

        CHAR(255),
DRIVER_INVOLVED
        SYSNAME 'DRIVER INVOLVED'
        INTEGER,
LOCATION
        CHAR(4),
MISC_CLAIM_DESC
        SYSNAME 'MISC CLAIM DESC'
        CHAR(100),
POLICY_NO
        SYSNAME 'POLICY NO'
        CHAR(6),
SETTLEMENT_AMOUNT
        SYSNAME 'SETTLEMENT AMOUNT'
        INTEGER,
SETTLEMENT_DATE
        SYSNAME 'SETTLEMENT DATE'
        INTEGER,
TIME
        CHAR(4),
VIN_INVOLVED
        SYSNAME 'VIN INVOLVED'
        CHAR(10),
WEATHER
        CHAR(20));
GRANT ALL PRIVILEGES ON CLAIMS03 TO PUBLIC;

```

VIEWS against the CLIENTS table

The following views of the CLIENTS table represent only one of many ways of constructing a set of views for the table.

POLICIES view

```

CREATE VIEW POLICIES
  (ADDRESS, AGENT, ANNIV_DATE, CITY, DATE_OF_BIRTH,
FULLNAME,
  POLICY_NO, POLICYHOLDER, STATE, TOTAL_PREMIUM, ZIP) AS
SELECT
  ADDRESS, AGENT, ANNIV_DATE, CITY, DATE_OF_BIRTH,
FULLNAME,
  POLICY_NO, POLICYHOLDER, STATE, TOTAL_PREMIUM, ZIP
FROM CLIENTS
WHERE RECTYPE = 'POLICYHOLDER';
GRANT ALL PRIVILEGES ON POLICIES TO PUBLIC;

```

DRIVERS view

```

CREATE VIEW DRIVERS

```

```
(DATE_OF_BIRTH, DRIVER_ID, FULLNAME, MARITAL_STATUS,  
POLICY_NO, SEX, STATE) AS  
SELECT  
    DATE_OF_BIRTH, DRIVER_ID, FULLNAME, MARITAL_STATUS,  
    POLICY_NO, SEX, STATE  
FROM CLIENTS  
WHERE RECTYPE = 'DRIVER';  
GRANT ALL PRIVILEGES ON DRIVERS TO PUBLIC;
```


Index

A

- ADD clause, ALTER TABLE statement 78, 84
- ALTER TABLE statement
 - effect on SQL catalog 47
 - privileges for 88
 - syntax and description 77
- approximate numeric type 185
- asterisk (*)
 - wildcard in SELECT list statement 174
- AT-MOST-ONE field attribute 113
- attributes
 - Model 204 field 22, 32, 35
 - SQL column 35
- Audit trail messages
 - overflow data 26
- authorization ID, SQL
 - as default schema name 51
 - determining 49
 - for TSF 105
 - in DDL statement security 77, 86
 - setting 89, 164

B

- base table, SQL 49, 106
- BINARY fields, Model 204
 - mapping 4, 34, 35
 - precision 39
- BLOB data type 56
 - see also data types, SQL

C

- CASCADE option 66
- CATALOG schema 15, 143
- CCACAT file
 - creation 17
 - file management 16 to 19
 - recovery 18
 - security 16, 18
- CCACATREPT subsystem 130
- CCACATREPT utility 8, 15, 129 to 143
- CCAIN file parameters 4

- CCATEMP file 22
- CCATSF subsystem 7, 98, 103
 - see also TSF
- CHARACTER data type 56, 117
 - see also data types, SQL
- character string type 185
- characters
 - non-printable 22
- CHECK clause, CREATE TABLE statement 53, 57
- CHECK column constraint 185
- clients, Model 204 SQL 9
- CLOB data type 56
 - see also data types, SQL
- CODED fields, Model 204 4
- Column Attributes panel, TSF 102, 113 to 118
- column attributes, SQL 35
- column definition
 - SQL catalog view of 147, 148
 - syntax and description 55 to 62
- Column List panel, TSF 102, 108 to 113
- column names, see names, SQL
- column position 78
- column updating privileges 83 to 86
- COLUMN_PRIVILEGES view, SQL catalog 143, 152
- columns
 - matching in NATURAL JOIN feature 176
- COLUMNS view, SQL catalog 143, 148
- Completion panel, TSF 102, 120, 125 to 127
- conditional join feature
 - using an ON clause 177
- Connect *
 - clients 9
 - processing components 5, 9
 - processing configuration 1
- consistency, data
 - see data consistency
- constraints
 - CHECK 53, 57, 185
 - column 56
 - FOREIGN KEY 52
 - nested table 173
 - NOT NULL, see NOT NULL clause
 - referential 100
 - see also REFERENCES clause

- table 52
- UNIQUE, see UNIQUE clause
- conversion, data
 - see data, Model 204 file
- CREATE SCHEMA statement
 - description 50
 - effect on SQL catalog 46
 - in TSF output 104
 - privileges for 86
 - schema context 75
 - syntax 50
- CREATE TABLE statement
 - description 52 to 54
 - effect on SQL catalog 46
 - privileges for 87
 - syntax 52
 - table types created 49
- CREATE VIEW statement
 - effect on SQL catalog 47
 - privileges for 87, 89 to 95
 - syntax and description 70
- CREATE.CCACAT procedure 17
- CROSS JOIN feature
 - no join condition specified 176
- CURRENT_DATE keyword
 - SELECT statement option 175
- CURRENT_TIME keyword
 - SELECT statement option 175
- CURRENT_TIMESTAMP keyword
 - SELECT statement option 175
- CVI utility
 - introduction to 13, 14

D

- data consistency
 - CCACAT file 16, 18
 - SQL catalog definitions 15, 162
 - TSF 100
- Data Definition Language, Model 204 SQL
 - see DDL, Model 204 SQL
- data length
 - for CHARACTER columns 117, 147, 148
- Data Manipulation Language, Model 204 SQL
 - see DML, Model 204 SQL
- data precision
 - see precision, data
- data types, SQL
 - default, TSF 116
 - mapping 25, 31 to 42, 116
 - precision limits 39
 - SQL catalog view of 148, 149
 - syntax of 55
- data, Model 204 file
 - conversion 35
 - dirty 37
 - features available to SQL 23
 - formats 31 to 42
 - precision, see precision
 - retrieval efficiency 32
- DDL, Model 204 SQL
 - extensions 48, 100
 - function of 3
 - manually created 14, 100
 - mixing with SQL DML 163
 - statement security 86
 - syntax 181
 - updates, large 19
 - utility-generated 14, 127
- DECIMAL data type 56, 117
 - see also data types, SQL
- DEFAULT clause, CREATE TABLE statement 57, 185
- DEFERRABLE fields, Model 204 4
- DEFINE commands, Model 204 4
- DELETE option
 - GRANT statement 82
 - REVOKE statement 83
- DELETE statement 165
- demonstration database, sample mapping of 189
- Dictionary, Model 204 13
- Dirty Data
 - migration issues 26
- dirty data
 - see data, Model 204 file
- DISTINCT option 75
- DML, Model 204 SQL
 - for DML statements 162
 - function of 3
 - mixing with SQL DDL 163
 - privileges, see privileges, SQL
 - using 161 to 174
- DOUBLE PRECISION data type 56, 117
 - see also data types, SQL
- DROP clause, ALTER TABLE statement 47, 78, 84
- DROP SCHEMA statement
 - effect on SQL catalog 47, 80
 - privileges for 87
 - syntax 79
- DROP TABLE statement
 - effect on SQL catalog 47
 - privileges for 87
 - syntax and description 80
- DROP VIEW statement
 - effect on SQL catalog 47, 80
 - privileges for 87
 - syntax 79

E

- empty strings 37
- error messages, TSF 101
- exact numeric type 185
- EXISTS clause 70
- exponents, floating point 42
- extensions
 - Model 204 SQL DDL 48, 100

F

- field level security, Model 204 72
- fields, Model 204
 - attributes of 22, 32, 35
 - names of 111
 - SQL mapping of 42
- file groups, Model 204
 - in TSF 106
 - simulating 3, 23, 72
- FILEORG parameter 18, 24, 113
- files, Model 204
 - data indexes, see indexes
 - hash key, see hash key files
 - identifying, in TSF 106
 - INVISIBLE fields in 164
 - mixed record type 72, 73
 - names for 53 to 54
 - sorted, see sorted files
 - transaction backout 3, 21
 - TSF pending definition for 107
- FIPS 8
- FLOAT data type 56, 117
 - see also data types, SQL
- FLOAT fields, Model 204
 - mapping 4, 34, 35
 - precision 40
- floating point data 40, 42
- floating point numbers
 - SQL processing 40
- floating point representation 42
- FOREIGN KEY clause 63
- foreign keys
 - defining in TSF 108
 - description of 27 to 30, 107
 - mapping examples 169
 - rules for 65
- FRCVOPT parameter 18
- FRV fields
 - SQL SELECT DISTINCT processing 32

G

- Grant Authority panel, TSF 102
- GRANT statement
 - effect on SQL catalog 47
 - for SQL security 5, 81
 - generating with TSF 120
 - privileges for 88
 - syntax 81
- GROUP BY clause 70, 75, 166
- groups
 - file, see file groups
 - multiply occurring, see multiply occurring fields

H

- hash key files 24, 57, 113
- HASHKEY default 113
- HAVING clause 70, 75

I

- index field, multi-column unique 60, 122
- indexes, file data 25, 32
- INNER JOIN keywords
 - SQL features 176
- INSERT option
 - GRANT statement 82
 - REVOKE statement 83
- INSERT statement 165
- INTEGER data type 56, 117
 - see also data types, SQL
- INVISIBLE fields
 - in nested tables 63, 107, 108
 - SQL DML restrictions 164 to 168
 - TSF 113
 - usability 4, 24, 57
 - with multi-column unique 122
- IODEV threads, SQL
 - required for ONLINE job 4
- isolation level, SQL 162

K

- KEY fields, Model 204 4, 34
- KEY_COLUMN_USAGE view, SQL catalog 143, 150

L

- LEN field attribute, Model 204 4
- length, data

- see data length
- locking, record 162, 163
- LOGIN command 49
- login security, Model 204
 - protecting SQL files 16, 23, 49
 - user ID rules 105

M

- Main Menu panel, CCACATREPT 131
- Main Menu panel, TSF 102, 108
- Mapping
 - ODBC/SQL table to Model 204 data 25
- mapping SQL data
 - recommendations 40
- Migration issues
 - data extraction mismatches 26
- Model 204
 - field definitions with SQL SELECT DISTINCT 32
 - SQL data extraction mismatches 26
- Model 204 tables 25
- Model 204 TCP/IP facility
 - brief description 8
- Model 204 views 25
- MODIFY clause, ALTER TABLE statement 47, 79
- multi-column unique keys
 - generating with TSF 120, 120 to 123
 - INVISIBLE fields as 24
 - simulating 68
 - specifying 58 to 62
 - SQL catalog view of 150
- Multi-Column Unique panel, TSF 102, 120 to 123
- multiple record types, mapping 24
- multiply occurring fields
 - definition of 108
 - mapped to nested tables 49
 - mapping 22, 28 to 30, 65
 - mapping examples 168
 - retrieving 170
 - updating 39
- multiply occurring fields <singlepage>
- multiply occurring groups 170
 - see also multiply occurring fields

N

- names, SQL object
 - columns 58, 112, 123
 - multi-column unique keys 59
 - qualifying 75
 - schemas 51, 106
 - standard identifier rules 50

- tables 53, 105
- NATURAL JOIN feature
 - common columns 176
- Nested tables 26
- nested tables
 - constraints for 173
 - creating 62
 - description 27 to 30, 49
 - for multiply occurring fields 4, 24
 - porting applications with 174
 - SQL DML against 168 to 174
 - TSF 106, 108
- NESTED USING clause
 - description 48
 - in syntax 62, 63
 - in TSF DDL 100, 106
- nesting keys, see foreign keys
- NOT NULL clause
 - data checking 38
 - for nested tables 63, 108, 174
 - in multiple record type files 25
 - in TSF 115
 - INVISIBLE field restrictions 168
 - mapping 56
 - modifying 79
 - reported in SQL catalog view 148
 - syntax rules 56
 - with UNIQUE 58
- nulls, SQL
 - in TSF 115
 - missing fields as 38
 - retrieval of 38
 - see also NOT NULL
- NUMERIC data type 56, 117
 - see also data types, SQL
- NUMERIC RANGE fields, Model 204 4, 34

O

- OCCURS field attribute, Model 204 4
 - see also multiply occurring fields
- ODBC Driver, Connect * 9
- ODBC/SQL compliant applications
 - updating tables 25
- ON DELETE clause 66
- ON UPDATE clause 66
- Open Database Connectivity (ODBC) Interface 9
- ORDER BY clause
 - for SQL catalog queries 145
- ORDERED attributes
 - SQL WHERE processing 32
- ORDERED field
 - in Model 204 table 25

- ORDERED fields, Model 204
 - for primary keys 63, 108
 - for UNIQUE columns 117, 122
 - mapping 4, 22, 34
- OUTER JOIN features
 - preserving unmatched rows 177

P

- PAD character, Model 204 23
- parent table, SQL 27, 106 to 108
- pending definition, TSF 107
- PF keys
 - CCACATREPT 132
 - TSF 103
- precision, data 39, 117, 147, 148
- PRIMARY KEY
 - application tables defined with 26
 - definition in SQL catalog 25
- PRIMARY KEY clause
 - in TSF 113
 - syntax rules 53, 56, 63
 - with SYSTEM 48
- PRIMARY KEY table columns 25
- primary keys
 - description of 27 to 30
 - in TSF 107, 108, 113
 - mapping examples 169
 - system-generated 68, 82, 170
 - see also PRIMARY KEY
- PRIVDEF parameter 18
- privileges, SQL
 - changing 81
 - column updating 83
 - for DDL statements 86
 - granting, see GRANT
 - reporting 15, 133
 - SQL catalog view of 151, 152
- PUBLIC option
 - GRANT statement 82, 133
 - REVOKE statement 83

Q

- query expression
 - CREATE VIEW statement 70
- query specifications
 - for views 70
 - see also SELECT statement
- querying, SQL catalog
 - see SQL catalog

R

- REAL data type 56, 117
 - see also data types, SQL
- record locking 162, 163
- recovery, CCACAT file 18
- REFERENCES clause
 - description 48, 65
 - in syntax 30, 53, 56, 63
 - in TSF 100
 - privileges for 82, 186
 - with referential triggered action 66
- referential constraints 65
 - see also REFERENCES clause
- referential triggered action 66
- reserved words, Model 204 SQL 187
- REVOKE statement
 - effect on SQL catalog 47
 - effect on UPDATE privileges 85
 - for SQL security 5
 - privileges for 88
 - syntax 82

S

- scale, column 117, 147, 148
- SCFE 7
- SCHEMAS view, SQL catalog 143, 145
- schemas, SQL
 - default context for 75, 76, 164
 - dropping, see DROP SCHEMA
 - names for 106
 - SQL catalog view of 145
- SCHEMATA view, SQL catalog 143, 145
- security, Model 204
 - field level 71
 - file 23
 - login, see login security, Model 204
 - subsystem 4
- security, SQL
 - from GRANT and REVOKE 5, 13, 23
 - includes authorization ID 81, 105
 - see also GRANT statement
- SELECT DISTINCT statement
 - Model 204 field definitions affected 32
- SELECT LIST statement
 - assigning alias names 174
- SELECT option
 - GRANT statement 82, 89
 - REVOKE statement 83
- SELECT statement
 - against views 71, 74
 - defining views 74

- INVISIBLE field restrictions 164
- SELECT statement options
 - CURRENT_DATE 175
 - CURRENT_TIME 175
 - CURRENT_TIMESTAMP 175
 - ON clause for conditional join 177
 - SQLERROR 176
 - SQLSTATE 176
 - SQLVERSION 175
 - USER 175
- serializability, transaction 162
- SERIALX parameter 163
- SET SCHEMA statement
 - description 76
 - effect on SQL catalog 47
 - in CCACATREPT DDL 136
 - privileges for 88
 - with SQL DML 164
- SET USER statement 47, 76, 89, 164
- SMALLINT data type 56
 - see also data types, SQL
- sorted files 24, 57, 113
- SORTKEY default 113
- SQL catalog
 - consistency, see data consistency
 - description 16
 - populating, see DDLUTIL
 - PRIMARY KEY definition 25
 - querying 13, 15, 143 to 158
 - record types 46
 - reporting, see CCACATREPT
 - views of 143
- SQL Client Front End, see SCFE
- SQL communications interface 7
- SQL data
 - recommended mappings 40
- SQL data extraction
 - dirty data 26
 - Model 204 messages 26
 - Model 204 mismatches 26
- SQL Engine 7
- SQL join features
 - INNER JOIN 176
 - NATURAL JOIN 176
 - ON clause 177
 - OUTER JOIN 177
 - USING clause 177
- SQL pattern search
 - guidelines 23
- SQL processing
 - floating point numbers 40
- SQL security, see security, SQL
- SQL SELECT DISTINCT statements
 - FRV fields 32

- SQL Server Front End, see SSFE
- SQL Server, Model 204
 - clients 1
 - components 7
- SQL WHERE processing
 - Model 204 ORDERED fields 32
- SQLERROR keyword
 - SELECT statement option 176
- SQLFILE parameter 53 to 54
- SQLSTATE keyword
 - SELECT statement option 176
- SQLVERSION keyword
 - SELECT statement option 175
- SSFE 7
- standards, SQL 8
- STRING fields, Model 204
 - hexadecimal zeros in 22
 - mapping 4, 34, 35
- subsystem security 4
- syntax
 - Model 204 SQL DDL statements 181
- SYSNAME clause
 - description 48
 - for columns 58
 - for multi-column key 59
 - for tables 53, 105
 - in TSF 100, 112, 123
 - rules for 64
- SYSTEM clause
 - description 48
 - in TSF 100, 113
 - rules for 64
- system-generated keys
 - defining 67
 - in TSF 100, 107, 112, 113
 - INSERTs into tables having 69
 - see also SYSTEM clause

T

- Table Specification facility, see TSF
- TABLE_COLUMNS view, SQL catalog 143, 147
- TABLE_CONSTRAINTS view, SQL catalog 143, 150
- TABLE_PRIVILEGES view, SQL catalog 143, 151
- Tables
 - nested 26
 - updating 25
- TABLES view, SQL catalog 143, 146
- tables, SQL
 - dropping, see DROP TABLE
 - names for 53 to 54, 105
 - nested, see nested tables

- SQL catalog view of 146, 149
- types of 106
- TCP-IP software, IBM 8
- Third-party application
 - tables 25
- transaction backout files 3, 21
- TSF
 - data length 117
 - description 7, 97 to 100
 - panels 127
 - parent tables 107
 - PF keys 103
 - processing 102
 - schema names 106
 - table names 105
 - table types 106
 - see also CCATSF subsystem

U

- UNION operator 70, 74
- UNIQUE clause
 - for multi-column keys, see multi-column unique keys
 - in syntax 53, 56
 - in TSF 117
 - modifying 79
 - with NOT NULL 58
- UNIQUE field
 - in Model 204 table 25
- UNIQUE fields, Model 204
 - for primary keys 22, 63, 108
 - for UNIQUE columns 39, 117, 122
- UNIQUE key
 - in ODBC/SQL table or view 25
- updatability, view 95
- UPDATE AT END field attribute 63
- UPDATE clause
 - GRANT statement 82, 83
 - REVOKE statement 83, 85
- UPDATE statement 165
- Updating tables
 - in ODBC/SQL compliant applications 25
- user ID, Model 204 49, 86, 105
- USER keyword 49
 - SELECT statement option 175
- USING clause feature
 - matching column names 177

V

- Views 25
 - defined with a PRIMARY KEY 26

- Model 204 25
- VIEWS view, SQL catalog 143, 149
- views, SQL
 - defining 69
 - description of 49, 106
 - dropping, see DROP VIEW
 - for mapping multiple record types 25
 - privileges for 89
 - querying 74

W

- WHERE clause 166
- wildcard
 - asterisk (*) 174
- WITH CHECK OPTION clause, CREATE VIEW
 - statement 53, 57, 70, 149
- WITH GRANT OPTION, GRANT statement
 - effect on UPDATE privileges 85
 - SQL catalog view of 152

