

ODBC Data Driver Contents

Overview

The SQL Language

Installing ODBC

Connection and Statement Handles

Data Driver

Listing the Data Sources

ODBC error messages

Connecting to a data source

Selecting and reading data

Updating a record

Creating a new file

SQL Statements

SQL Elements

SQL Reserved Words

Overview

The Open Database Connectivity (ODBC) interface allows applications to access data from database management systems (DBMS), using Structured Query Language (SQL) expressions.

J Win9x/NT only supports 32-bit ODBC drivers.

The J/ODBC interface uses the *Data Driver* verbs, defined in script dd.ijs.

The processing required by the application is essentially independent of the DBMS. For example, if you have written programs to access dBase files using ODBC, then you can use similar programs to access Paradox, or indeed any other DBMS.

The application, and the driver programs that access the DBMS, are physically separate. To access a DBMS, you need only ensure a driver for that DBMS is available. There are ODBC drivers for virtually all commercial DBMS, and J itself is distributed with drivers for several popular systems, including Access, Btrieve, dBase, Excel, FoxPro, Oracle, Paradox and SQL Server.

Note that the DBMS need not itself support SQL, for example dBase does not. All that matters is that the ODBC DBMS *driver* is available.

An ODBC DBMS driver, together with information on where its datasets are stored, is typically referred to as a *data source*.

Between the application and the data there are 3 layers, though they appear to the application as a single unit:

J application		
Data Driver		
ODBC Manager		
Data Source	Data Source	Data Source
Data	Data	Data

The application sends requests to the Data Driver (using the verbs defined in script dd.ijs), which converts them into a standard format and sends them to the ODBC Manager. The ODBC Manager then sends them to the appropriate Data Source, and is also responsible for ensuring the required drivers are loaded. The Data Source drivers then handle the data access.

The SQL language

SQL, or Structured Query Language, is a widely accepted protocol used for data access. It is an ANSI standard with SQL-92 being the most recent specification. A summary of the language appears in Appendix B.

The language is defined with various levels, listed in the appendix as minimum, core and extended. All SQL servers support at least the minimum level, some may also support core or extended levels, or extensions of their own. ODBC DBMS drivers are distributed with Help files that list the functionality of the driver, plus other useful information - when you install the drivers you should also print out the Help files for reference.

Installing ODBC

ODBC must be installed on your system. An easy way to check for this is to look for the 32bit ODBC icon in Control Panel. ODBC is installed automatically with many Microsoft applications.

If you do not have ODBC installed, you may be able to obtain ODBC drivers from the Microsoft web site www.microsoft.com. Search for the ODBC FAQ (at the time of writing this is at www.microsoft.com/data/odbc/faq_odbc.htm).

To follow the examples in this chapter, you should install the dBase driver, even if you do not intend to use it later. When you install this driver, or any other driver, you will be prompted to enter the *data source name*. The data source indicates the DBMS driver and where the data files are stored. Note that it is *not* the name of a specific data file as may be imagined.

The examples here use the dBase driver, and files stored in J subdirectory `examples\data`, and the data source name used is *jdata*. To set this up, in Program Manager, select the ODBC icon in Control Panel. Use Add or Setup to create a panel as shown:

Note that you should at least have one data source name as shown above, but you can set up multiple data source names, using the same driver but typically with a different directory.

Although the data source name specifies a directory, this is in fact used by the dBase driver as the default directory. You can override this default either by specifying another directory when you connect to the dBase driver, or afterwards when you specify a file with its full pathname. However, note this capability is provided by extensions to ODBC in the dBase driver, which are not necessarily found in other ODBC drivers.

Connection and Statement Handles

You start accessing a database by first opening a connection to the data source. The result is a number, called the *connection handle*. Subsequent requests to the data source use this connection handle.

When you send a request to select some data, the result is also a number, called the *statement handle*. Again, subsequent requests that reference this selection use the statement handle. There may be more than one statement handle associated with a connection handle. You may also have a statement handle not associated with a connection handle, where the request made was not specific to a particular data file.

In both cases, you should close the handle to free up resources when you are finished with it. For example, to read records from a file, you typically:

- open a connection to the data source, returning a connection handle
- make a selection on a specific file using the connection handle, returning a statement handle
- read the records, using the statement handle
- close the statement handle
- close the connection handle

You must always close a connection handle explicitly. However, a statement handle may be closed if you have fetched all records selected - see the discussion of `defer` below. In any case, it is good practice to always close the statement handle explicitly, as there is no harm done if you try to close a handle that had already been closed.

Data Driver

ODBC access is provided by the Data Driver verbs, that are defined in script dd.ijs. First load this file:

```
load 'dd'
```

The Data Driver verbs may be summarized as follows:

Here, *ch* refers to a connection handle, and *sh* a statement handle. Note that SQL commands are not case-sensitive.

ddcnm r=. ddcnm sh

Column names of selected data

ddcnt r=. ddcnt '' (available in J Win95 only)

Rowcount of last ddsq1 command

ddcol r=. 'tdata' ddcoll ch

Column names and attributes in a database

ddcom r=. ddcom ch

Commit a transaction (after a ddtrn)

ddcon ch=. ddcon 'dsn=jdata'

Connect to ODBC data source name. The result is a connection handle. The argument can set several parameters, separated by semicolons. Some are supported by all databases, and others have a meaning only for specific databases. Parameters recognized by most database systems are:

dsn	ODBC data source name
dlg	dlg=1 prompts for a connection string with a dialog box with entries for User Id and Password (not supported by the distributed dBase driver)
uid	user name
pwd	user password
modifysql	set to 1 (the default) to use ODBC SQL grammar. Set to 0 to use native database grammar.
rereadafterupdate	set to 1 to force a re-read of a record after an update. This is useful for retrieving auto-updated values such as timestamps.
rereadafterinsert	set to 1 to force a re-read of a record after an insert

For example:

```
ch=. ddcon'dsn=mydata;uid=george;pwd=sesame'
```

dddis r=. dddis ch

Closes connection handle (disconnects from the data source)

ddend r=. ddend sh

Closes statement handle

dderr r=. dderr ''

Return error message on last command. An error message is given when a data driver verb returns `_1`.

ddfet r=. ddfet sh,n

Fetch next records from selected data. Note that after you have read a record, the next fetch will not read it again. If you need to read it again, you must select it again. For example:

```
r=. ddfet sh            fetch next record (same as ddfet sh,1)
r=. ddfet sh,5          fetch next 5 records
r=. ddfet sh,_1        fetch all remaining records.
```

If you fetch all remaining records using `ddfet sh,_1`, or if your fetch returns fewer records than you requested (i.e. the fetch reads past the end of the file), then `ddfet` closes the statement handle. Otherwise, the statement handle remains open, and you should explicitly close it if you have finished reading the file.

ddfch r=. ddfch sh,n (available in J Win95 only)

As `ddfet`, but returns data in columns

ddrbk r=. ddrbk ch

Discards (rollbacks) a transaction (after a `ddtrn`)

ddsel sh=. 'select * from tdata' ddsel ch

Select data from a database, returning a statement handle

ddsql r=. 'create table mydata' ddsq1 ch

Execute an SQL statement

ddsrc r=. ddsrc ''

In J Win95:

Data source names available from the ODBC manager.
These names can be used as the `dsn=` argument to `ddcon`.

In J Win31:

a statement handle that can be used with `ddfet` to return data source names.

ddtbl `sh=. ddtbl ch`

Returns a statement handle for tables in the data source. Some ODBC drivers, including the distributed dBase driver, do not support this service and the result will be empty.

ddtrn `r=. ddtrn ch`

Begin a transaction on a connection. Subsequent actions are not committed to the database until a `ddcom` is done. Actions since the `ddtrn` can be discarded by doing a `ddrbk` (rollback).

Listing the Data Sources

The list of data sources that are specified in the ODBC Control Panel dialog box can be retrieved under program control. The verb `ddsrc` requests this information. In J Win95, the result is the list of data sources; in J Win31, the result is a statement handle that may be used with `ddfet` to return the list of data sources.

```
ddsrc ''
+-----+-----+
|MS Access 7.0 Database|Microsoft Access Driver (*.mdb) |
+-----+-----+
|Excel Files           |Microsoft Excel Driver (*.xls)  |
+-----+-----+
|FoxPro Files          |Microsoft FoxPro Driver (*.dbf) |
+-----+-----+
|Text Files            |Microsoft Text Driver (*.txt; *.csv)|
+-----+-----+
|dBASE Files           |Microsoft dBase Driver (*.dbf)  |
+-----+-----+
|jdata                 |Microsoft dBase Driver (*.dbf)  |
+-----+-----+
```

The result will depend on the drivers you have set up. Note that you should see an entry for *jdata*, which is the data source name you assigned to the `examples\data` subdirectory.

The result has 2 columns: the data source name, and a description of the driver.

ODBC error messages

The data driver verbs that open and close connections, and send SQL statements, all return a number. Typically, if the number is positive, it is a handle. If the number is 0, it means the operation completed successfully (but the function returns no handle). If the number is `_1`, it means there was some error. You can get more information about the error using `dderr`. For example, try closing a non-existent statement handle:

```
ddend 42  
_1
```

```
dderr''  
ISI04 Bad statement handle
```

Connecting to a data source

The `ddcon` command connects to a data source returning a connection handle, using the form:

```
[ch=. ddcon 'dsn=jdata'  
1
```

We have now connected to a data source, but not yet to a data file. The file we will use is *tdata*, and the next statement uses `ddcol` to retrieve the column names and attributes for this file:

```
$cols=. 'tdata' ddcol ch  
7 13
```

```
3 4 7 {"1 cols  
+-----+-----+-----+  
|Column|Type|Type_Name|  
+-----+-----+-----+  
|NAME  |1   |CHAR      |  
+-----+-----+-----+  
|SEX   |1   |CHAR      |  
+-----+-----+-----+  
|DEPT  |1   |CHAR      |  
+-----+-----+-----+  
...
```

Selecting and reading data

To read data from a file, you first select the data you want to read using the `ddsel` verb, which returns a statement handle. You then use `ddfet` or `ddfch` to fetch the records.

The left argument of `ddsel` is a SQL selection expression. Here are typical examples:

Select all records (* means all columns):

```
sh=. 'select * from tdata' ddsel ch
```

Fetch the first 3 records:

```
ddfet sh,3
+-----+-----+-----+-----+-----+
|MACDONALD B   |F|D101|1.95906e7|1.97805e7|32591|
+-----+-----+-----+-----+-----+
|GENEREAUX S   |F|D103|1.94503e7|1.96602e7|95415|
+-----+-----+-----+-----+-----+
|KOEBEL R      |M|D101|1.93711e7|1.98009e7|63374|
+-----+-----+-----+-----+-----+
```

Fetch the next record:

```
ddfet sh
+-----+-----+-----+-----+-----+
|KELLER J      |F|D101|1.95105e7|1.97404e7|48898|
+-----+-----+-----+-----+-----+
```

Close the statement handle:

```
ddend sh
```

You should always close the statement handle when you no longer need it. However to avoid repetition, the remaining examples do not show this.

Select males with salary exceeding 40000:

```
sel=.'select * from tdata where sex='M' and salary >= 40000'
sh=. sel ddsel ch
ddfet sh,4
+-----+-----+-----+-----+-----+
|KOEBEL R      |M|D101|1.93711e7|1.98009e7|63374 |
+-----+-----+-----+-----+-----+
|NEWTON R      |M|D108|1.95601e7|1.97902e7|73368 |
+-----+-----+-----+-----+-----+
|DINGEE S      |M|D103|1.9641e7 |1.98309e7|46877 |
+-----+-----+-----+-----+-----+
|ROGERSON G    |M|D101|1.95712e7|1.98302e7|108777|
+-----+-----+-----+-----+-----+
```

Select only the name, department and salary fields, where date of birth is before 1950:

```
sel=. 'select name,dept,salary from tdata where dob<19500000'
```

Fetch the first such record:

```
ddfet (sel ddsel 1),1
+-----+-----+-----+
|GENEREAUX S   |D103|95415|
+-----+-----+-----+
```

Use ddfch to return data in columns:

```
[a=. ddfch 1005,_1
+-----+-----+-----+-----+-----+
|MACDONALD B   |F|D101|1.95906e7|1.97805e7|32591|
|GORDON E     |F|D103|1.95202e7|1.97908e7|29960|
|BAUERLEIN J  |F|D103|1.96204e7|1.98409e7|33668|
|CHESHER D    |F|D103| 1.9561e7|1.98408e7|35184|
+-----+-----+-----+-----+-----+
```

```
(;:'name sex dept dob doh salary')=. a
salary
32591
29960
33668
35184
```

Updating a record

To update a record, you process an SQL update statement. Here we update the salary field for ABBOTT K.

First read the record to see the current value (the salary is 50817):

```
sel=. 'select * from tdata where name=''ABBOTT K'''  
ddfet sel ddsel ch  
+-----+-----+-----+-----+-----+  
|ABBOTT K      |M|D103|1.9631e7|1.98309e7|50817|  
+-----+-----+-----+-----+-----+
```

Next we process an update statement:

```
us=. 'update tdata set salary=45000 where name=''ABBOTT K'''  
  
us ddsq1 ch
```

Finally, we read the record again, to ensure the update was successful:

```
sel=. 'select * from tdata where name=''ABBOTT K'''  
ddfet sel ddsel ch  
+-----+-----+-----+-----+-----+  
|ABBOTT K      |M|D101|1.9631e7|1.98309e7|45000|  
+-----+-----+-----+-----+-----+
```

Creating a new file

To create a new file, use the SQL *create* command, with a list of the column names and attributes in the new data set. Here we create file *test*, with columns for name and salary.

First, drop test in case it already exists (the `_1` result means the table was not found):

```
'drop table test' ddsql ch
_1
```

Now create the new file:

```
'create table test (name char(12),sal numeric)' ddsql ch
0
```

Add a record:

```
t=. 'insert into test (name,sal) values (''Neumann,E'',40000)''
t ddsql ch
0
```

Add another record:

```
t=. 'insert into test (name,sal) values (''James, P'',42000)''
t ddsql ch
0
```

Now read the file:

```
ddfet _1,~ 'select * from test' ddsel ch
+-----+-----+
|Neumann, E |40000|
+-----+-----+
|James, P   |42000|
+-----+-----+
```

SQL Statements

The following SQL statements define the base ODBC SQL grammar.

Statement	Min	Core	Ext
alter-table-statement ::= ALTER TABLE base-table-name { ADD column-identifier data-type ADD (column-identifier data-type [, column-identifier data-type]...) }		X	
create-index-statement ::= CREATE [UNIQUE] INDEX index-name ON base-table-name (column-identifier [ASC DESC] [, column-identifier [ASC DESC]]...)		X	
create-table-statement ::= CREATE TABLE base-table-name-1 (column-element [, column-element] ...) column-element ::= column-definition table-constraint-definition column-definition ::= column-identifier data-type DEFAULT default-value [column-constraint-definition [,column-constraint- definition]...] column-constraint-definition ::= NOT NULL UNIQUE PRIMARY KEY) REFERENCES base-table-name-2 referenced-columns CHECK (search-condition) default-value ::= literal NULL USER table-constraint-definition ::= UNIQUE (column-identifier [, column-identifier] ...) PRIMARY KEY (column-identifier [, column-identifier] ...) CHECK (search-condition) FOREIGN KEY referencing-columns REFERENCES base-table-name-2 referenced-columns	X		
create-view-statement ::= CREATE VIEW viewed-table-name [(column-identifier [, column-identifier]...)] AS query- specification		X	
delete-statement-positioned ::= DELETE FROM table-name WHERE CURRENT OF cursor- name		X X	(v2) X
delete-statement-searched ::= DELETE FROM table-name [WHERE search-condition]	X		
drop-index-statement ::= DROP INDEX index-name		X	

drop-table-statement ::= DROP TABLE base-table-name [CASCADE RESTRICT]	X		
drop-view-statement ::= DROP VIEW viewed-table-name [CASCADE RESTRICT]		X	
grant-statement ::= GRANT { ALL grant-privilege [, grant-privilege]... } ON table-name TO { PUBLIC user-name [, user-name]... } grant-privilege ::= DELETE INSERT SELECT UPDATE [(column-identifier [, column-identifier]...)] REFERENCES [(column-identifier [, column-identifier]...)]		X	
insert-statement ::= INSERT INTO table-name [(column-identifier [, column- identifier]...)] VALUES (insert-value[,insert-value]...)	X		
insert-statement ::= INSERT INTO table-name [(column-identifier [, column-identifier]...)] { query-specification VALUES (insert-value [, insert-value]...)}		X	
ODBC-procedure-extension ::= ODBC-std-esc-initiator [?=]call procedure ODBC-std-esc- terminator ODBC-ext-esc-initiator [?=]call procedure ODBC-ext-esc- terminator			X
revoke-statement ::= REVOKE { ALL revoke-privilege [, revoke-privilege]... } ON table-name FROM { PUBLIC user-name [, user-name]... } [CASCADE RESTRICT] revoke-privilege ::= DELETE INSERT SELECT UPDATE REFERENCES		X	
select-statement ::= SELECT [ALL DISTINCT] select-list FROM table-reference-list [WHERE search-condition] [order-by-clause]	X		

select-statement ::= SELECT [ALL DISTINCT] select-list FROM table-reference-list [WHERE search-condition] [GROUP BY column-name [, column-name]...] [HAVING search-condition] [UNION select-statement]... [order-by-clause]		X	
select-for-update-statement ::= SELECT [ALL DISTINCT] select-list FROM table-reference-list [WHERE search-condition] FOR UPDATE OF [column-name [, column-name]...]		X (v1)	X (v2)
statement ::= create-table-statement delete-statement-searched drop-table-statement insert-statement select-statement update-statement-searched	X		
statement ::= alter-table-statement create-index-statement create-table-statement create-view-statement delete-statement-positioned delete-statement-searched drop-index-statement drop-table-statement drop-view-statement grant-statement insert-statement revoke-statement select-statement select-for-update-statement update-statement-positioned update-statement-searched		X	

statement ::= alter-table-statement create-index-statement create-table-statement create-view-statement delete-statement-positioned (continued) delete-statement-searched drop-index-statement drop-table-statement drop-view-statement grant-statement insert-statement ODBC-procedure-extension revoke-statement select-statement select-for-update-statement statement-list update-statement-positioned update-statement-searched			X
statement-list ::= statement statement;statement-list			X
update-statement-positioned ::= UPDATE table-name SET column-identifier = {expression NULL} [, column-identifier = {expression NULL}]... WHERE CURRENT OF cursor-name		X (v1)	X (v2)
update-statement-searched UPDATE table-name SET column-identifier = {expression NULL } [, column-identifier = {expression NULL}]... [WHERE search-condition]	X		

Elements Used in SQL Statements

The following elements are used in the SQL statements listed previously .

Element	Min	Core	Ext
approximate-numeric-literal ::= mantissaEexponent mantissa ::= exact-numeric-literal exponent ::= [+ -] unsigned-integer		X	
approximate-numeric-type ::= FLOAT DOUBLE PRECISION REAL		X	
argument-list ::= expression expression, argument-list	X		
base-table-identifier ::= user-defined-name	X		
base-table-name ::= base-table-identifier	X		
base-table-name ::= [user-name.]base-table-identifier		X	
between-predicate ::= expression [NOT] BETWEEN expression AND expression		X	
binary-literal ::= {implementation defined}			X
binary-type ::= BINARY (length) VARBINARY (length) LONG VARBINARY			X
character ::= {any character in the implementor's character set}	X		
character-string-literal ::= '{character}...'	X		
character-string-type ::= CHARACTER(length) CHAR(length)	X		
character-string-type ::= CHARACTER(length) CHAR(length) CHARACTER VARYING(length) VARCHAR(length)		X	
character-string-type ::= CHARACTER(length) CHAR(length) CHARACTER VARYING(length) VARCHAR(length) LONG VARCHAR			X
column-identifier ::= user-defined-name	X		
column-name ::= [table-name.]column-identifier	X		
column-name ::= [{table-name correlation-name}.]column-identifier		X	
comparison-operator ::= < > <= >= = <>	X		
comparison-predicate ::= expression comparison-operator expression	X		
comparison-predicate ::= expression comparison-operator {expression (sub-query)}		X	

correlation-name ::= user-defined-name		X	
cursor-name ::= user-defined-name		X	
data-type ::= character-string-type	X		
data-type ::= character-string-type exact-numeric-type approximate-numeric-type		X	
data-type ::= character-string-type exact-numeric-type approximate-numeric-type binary-type date-type time-type timestamp-type			X
date-literal ::= 'date-value'			X
date-separator ::= -			X
date-type ::= DATE			X
date-value ::= years-value date-separator months-value date-separator days-value			X
days-value ::= digit digit	X		
digit ::= 0 1 2 3 4 5 6 7 8 9	X		
dynamic-parameter ::= ?	X		
exact-numeric-literal ::= [+ -] { unsigned-integer [.unsigned-integer] unsigned-integer. .unsigned-integer }		X	
exact-numeric-type ::= DECIMAL(precision,scale) NUMERIC(precision,scale) SMALLINT INTEGER			X
exact-numeric-type ::= DECIMAL(precision,scale) NUMERIC(precision,scale) BIT SMALLINT INTEGER BIGINT		X	
exists-predicate ::= EXISTS (sub-query)		X	

expression ::= term expression {+ -} term term ::= factor term {*/ } factor factor ::= [+ -]primary primary ::= column-name dynamic-parameter literal (continued) (expression) primary ::= column-name dynamic-parameter literal set-function-reference USER (expression) primary ::= column-name dynamic-parameter literal ODBC-scalar-function-extension set-function-reference USER (expression)	X		
hours-value ::= digit digit			X
index-identifier ::= user-defined-name		X	
index-name ::= [index-qualifier.]index-identifier		X	
Index-qualifier ::= user-defined-name		X	
in-predicate ::= expression [NOT] IN {(value {, value}...) (sub-query)} value ::= literal USER dynamic-parameter		X	

SQL Reserved Keywords

The following words are reserved for use in ODBC function calls. These words do not constrain the minimum SQL grammar; however, to ensure compatibility with drivers that support the core SQL grammar, applications should avoid using any of these keywords.

ABSOLUTE	DECLARE	INTEGER	REVOKE
ADA	DEFERRABLE	INTERSECT	RIGHT
ADD	DEFERRED	INTERVAL	ROLLBACK
ALL	DELETE	INTO	ROWS
ALLOCATE	DESC	IS	SCHEMA
ALTER	DESCRIBE	ISOLATION	SCROLL
AND	DESCRIPTOR	JOIN	SECOND
ANY	DIAGNOSTICS	KEY	SECTION
ARE	DICTIONARY	LANGUAGE	SELECT
AS	DISCONNECT	LAST	SEQUENCE
ASC	DISPLACEMENT	LEFT	SET
ASSERTION	DISTINCT	LEVEL	SIZE
AT	DOMAIN	LIKE	SMALLINT
AUTHORIZATION	DOUBLE	LOCAL	SOME
AVG	DROP	LOWER	SQL
BEGIN	ELSE	MATCH	SQLCA
BETWEEN	END	MAX	SQLCODE
BIT	END-EXEC	MIN	SQLERROR
BIT_LENGTH	ESCAPE	MINUTE	SQLSTATE
BY	EXCEPT	MODULE	SQLWARNING
CASCADE	EXCEPTION	MONTH	SUBSTRING
CASCADED	EXEC	MUMPS	SUM
CASE	EXECUTE	NAMES	SYSTEM
CAST	EXISTS	NATIONAL	TABLE
CATALOG	EXTERNAL	NCHAR	TEMPORARY
CHAR	EXTRACT	NEXT	THEN
CHAR_LENGTH	FALSE	NONE	TIME
CHARACTER	FETCH	NOT	TIMESTAMP
CHARACTER_LENGTH	FIRST	NULL	TIMEZONE_HOUR
CHECK	FLOAT	NULLIF	TIMEZONE_MINUTE
CLOSE	FOR	NUMERIC	TO
COALESCE	FOREIGN	OCTET_LENGTH	TRANSACTION
COBOL	FORTRAN	OF	TRANSLATE
COLLATE	FOUND	OFF	TRANSLATION
COLLATION	FROM	ON	TRUE
COLUMN	FULL	ONLY	UNION
COMMIT	GET	OPEN	UNIQUE
CONNECT	GLOBAL	OPTION	UNKNOWN
CONNECTION	GO	OR	UPDATE
CONSTRAINT	GOTO	ORDER	UPPER
CONSTRAINTS	GRANT	OUTER	USAGE
CONTINUE	GROUP	OUTPUT	USER
CONVERT	HAVING	OVERLAPS	USING
CORRESPONDING	HOUR	PARTIAL	VALUE
COUNT	IDENTITY	PASCAL	VALUES
CREATE	IGNORE	PLI	VARCHAR
CURRENT	IMMEDIATE	POSITION	VARYING
CURRENT_DATE	IN	PRECISION	VIEW

CURRENT_TIME	INCLUDE	PREPARE	WHEN
CURRENT_TIMESTAM	INDEX	PRESERVE	WHENEVER
CURSOR	INDICATOR	PRIMARY	WHERE
DATE	INITIALLY	PRIOR	WITH
DAY	INNER	PRIVILEGES	WORK
DEALLOCATE	INPUT	PROCEDURE	YEAR
DEC	INSENSITIVE	PUBLIC	
DECIMAL	INSERT	RESTRICT	