Microsoft® Jet Database Engine 2.0

A User's Overview

By Paul Litwin, Editor
Smart Access

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, this paper should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This document is for informational purposes only. MICROSOFT AND PINNACLE PUBLISHING MAKE NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

Microsoft Word was used in the production of this paper.

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399
800-426-9400

Pinnacle Publishing, Inc.
18000 72nd Avenue South, Suite 217
Kent, WA 98032
800-788-1900, 206-251-1900, fax 206-251-5057

9403 Part # 098-54938

Table of Contents

Introduction

This white paper provides an introduction to the Microsoft Jet Database Engine—the advanced relational database engine built into Microsoft Access® and Visual Basic®. The intended audience for this paper is the Microsoft Access user who wishes to understand how data is managed within the product. After reading this paper, you should have a good idea of what Jet is, how it works, and how best to take advantage of its features.

What Is A Database Engine?

Database management systems (DBMS) are programs that are used to manage data. DBMS programs consist of many components, not the least of which are the user interface and the database engine (see Figure 1). The *user interface* handles all of the interactions between the user and the DBMS program, including the viewing and editing of data through forms, reports, and so forth. The *database engine*, on the other hand, is the part of the DBMS program that is the link between the DBMS program and the data.

Some DBMS programs have simple database engines that are nothing more than routines for reading and writing records of data. Because of this, these DBMS programs lack support for security, referential integrity, transaction processing, remote data and other advanced features that users in today's corporate environments require.

Advanced database engines provide most or all of these critical features, as well as providing a single entry point for all users of a database. This ensures that all users—even if the user is another program—are using the data stored in the database in a manner consistent with the intentions of the database owner.

The database engine built into Microsoft Access—the *Microsoft Jet Database Engine*—is an advanced database engine that provides all of these critical features and more.

Figure 1. *A typical DBMS program.*

Microsoft Jet Database Engine

Jet is the name of Microsoft's relational database engine that runs on the Microsoft Windows™ family of operating systems. Jet handles all of the database processing for Microsoft Access and Visual Basic. The Jet engine also can provide data to Open DataBase Connectivity (ODBC™) client applications using the Microsoft Access 2.0 ODBC driver.

The Microsoft Jet Database Engine is not a separately packaged product. Currently, it is only sold as part of Microsoft Access and Visual Basic. Jet has no user interface of its own. Instead users and application developers utilize Jet via an application program, most typically Microsoft Access. Figure 2 shows how the Jet engine fits into Microsoft Access.

Microsoft Jet is a modern relational database engine. It is a sophisticated, multiuser database engine and integrated query processor with built-in integrity, security and remote data access.

The Microsoft Jet engine is composed of a series of Dynamic Link Libraries (DLLs). These DLLs, which can be thought of as distinct but interconnected programs, all work together to comprise the Jet database engine. They are:

Figure 2. *Microsoft Access and Jet.*

- The Jet DLL
- The Data Access Objects DLL
- The External ISAM DLLs

In addition, Jet uses the ODBC Driver Manager DLL in order to provide access to ODBC data sources.

Figure 3 shows how these components interrelate.

Jet DLL

The Jet DLL (MSAJT200.DLL) is the main Jet engine program that evaluates and executes requests for data. If the request is for native Jet data—data stored in the Microsoft Access Database (.MDB) format—then the Jet DLL also handles the reading and writing of the data. If, on the other hand, the request involves non-native data, then the Jet DLL makes calls to either the ODBC Driver Manager DLL or one of the external ISAM DLLs.

Data Access Objects DLL

The Data Access Objects DLL (DAO2016.DLL) is the Jet component that provides a programmer interface to the Jet engine. Data Access Objects (DAO) is an object-based data access language that can be used by Access Basic and Visual Basic application developers to talk to Jet. DAO is a rich, high-level set of objects that insulates developers from the physical details of reading and writing records.

External ISAM DLLs

Jet provides access to several external Indexed Sequential Access Method (ISAM) format files using a series of installable DLL files. The Jet engine supports three external ISAM formats: Xbase (XBS200.DLL), Paradox (PDX200.DLL) and Btrieve (BTRV200.DLL). These DLLs handle the reading and writing of data stored in dBASE®, FoxPro®, Paradox®, and Btrieve® file formats.

Figure 3. *The Microsoft Jet Database Engine architecture.*

The Microsoft Access Database File Format

Jet stores data in the Microsoft Access Database (.MDB) file format, a type of Indexed Sequential Access Method (ISAM) format file. A single .MDB database file can be used to store data for hundreds or even thousands of related tables (there is no arbitrary limit to the number of tables in a database). But Jet stores more than just tables in its database files. It also stores indexes (up to 32 per table), relationships between tables, validation rules, query definitions, and security permissions in the .MDB file. In addition, when using Microsoft Access, .MDB files also store all of the form, report, macro and Access Basic modules used by Microsoft Access.

Datatype Support

Jet supports a rich assortment of simple and complex datatypes. The supported datatypes are listed in the following table:

Text	Up to 255 bytes	Variable length text
Memo	Up to 1.2 GB (max database size)	Large variable length text
Numeric-Byte	1 byte	0 through 255
Numeric-Integer	2 bytes	-32,768 through 32,767
Numeric-Long Integer	4 bytes	-2,147,483,648 through 2,147,483,647
Numeric-Single	4 bytes	-3.4×10^{38} through 3.4×10^{38}
Numeric-Double	8 bytes	-1.8×10^{308} through -1.8×10^{308}
Currency	8 bytes	Numbers with up to 15 digits to the left of decimal point and 4 to the right
Counter	4 bytes	Automatically incremented long integer
Yes/No	1 bit	Boolean data
Date/Time	8 bytes	Dates and time values
OLE	Up to 1.2 GB (max database size)	OLE objects, graphics, and other complex data
Binary	Up to 1.2 GB (max	Variable length binary data

database size)

Multiuser Data Access

Jet controls access to shared data in a multiuser environment using *locks*. Jet employs a *page locking* model. Thus, when Jet needs to lock an individual record for a table, it locks a 2K (2048 bytes) page containing that record. This also means that Jet reads and writes data a page at a time, rather than a record at time. For many situations, this type of locking is more efficient than the locking of individual records because it requires less overhead. It also means, however, that Jet will possibly lock more than one record at a given point in time. For example, in Figure 4, if a user locks order# 105, orders 101, 102, and 112 will also be locked.

Figure 4. *Jet locks 2K pages of records.*

Two Locking Schemes

Jet supports two multiuser locking schemes: optimistic and pessimistic locking. By default, Jet employs *optimistic locking*, which means that it doesn't request a lock on a page of records until the user saves changes to a record. The advantage of optimistic locking is that when using this scheme, Jet locks records for a very short period of time minimizing potential locking collisions. The disadvantage is that more than one user can be editing the same record at the same time and the application must then deal with the situation when several users attempt to save different versions of a record.

Under *pessimistic locking*, when a user requests a lock, Jet immediately locks the page containing that record. In an application, this typically occurs when a user begins editing a record using a data entry form. Until the user releases the lock, by saving or abandoning changes to the locked record, the record—and all records on the same page—will be unavailable for editing. While locked records can't be edited by other users, they can still be viewed by those users on the network. Regardless of which locking scheme is employed, Jet maintains page locks automatically without user (or developer) intervention.

Jet Transactions

A *transaction* is a set of operations that must be considered as a single unit. For example, when transferring funds from one account to another in a financial application, a well-behaved application should ensure that *both* operations (the credit to the receiving account and the debit to the source account) succeed or that neither one does. The Jet engine supports *transaction processing*.

Jet's Data Access Objects language supports transaction processing by means of three workspace methods (see the section on Data Access Objects later in this paper for more details on methods):

- BeginTrans—designates the start of a new transaction
- CommitTrans—designates the end of a transaction and also serves to commit (or save) the transaction
- Rollback—abandons any changes made since the beginning of the transaction and ends the transaction

Jet supports transactions only against foreign data sources (ISAM and ODBC) that support transactions themselves. For example, there is no transaction capability in Paradox, so transactions are not supported on Paradox data. SQL Server, on the other hand, does support transactions. Jet will support one level of transactions against an ODBC data source.

Since the operations in a transaction are saved up in memory until the entire transaction is committed, application developers can benefit from their use even when a transaction would not otherwise be necessary. If, for example, an application routine needed to loop through 5,000 records and update each in turn, it would normally require 5,000 disk writes. If this loop was wrapped in a transaction, however, the 5,000 single-record disk writes would be replaced with one 5,000-record disk write. This would result in a considerable savings in execution time.

Data Integrity

Data integrity is an important component of relational databases, a component that desktop database management systems often lack. Microsoft Jet supports data integrity in several ways:

- Built-in support for primary keys, foreign keys, and referential integrity
- Support for database-specific rules
- A robust and programmable security model

Jet supports these key data integrity features at the *engine level*. This means that the application developer is free from having to create integrity rules using procedural code. It also means that everyone follows the same set of rules since the rules are consistently enforced by the database engine, not a specific user, form, or application. And because Jet databases can be completely secured, database administrators can easily prevent users without proper security clearance from "turning off" these data integrity features.

Primary Keys and Referential Integrity

The relational database model, as defined by Dr. E.F. Codd in 1969, requires support for two general integrity rules: entity integrity and referential integrity. Jet supports both rules at the engine level.

In order for a database to support *entity integrity*, it must first support the concept of primary keys. A *primary key* is a column (or multiple columns) that serves as the unique identifier for a row. The entity integrity rule requires that primary key columns never be null. Jet supports both primary keys and the entity integrity rule.

In order for a database to support *referential integrity*, it must also support the concept of foreign keys. A *foreign key* is a column in one table used to reference a primary key in another table. The referential integrity rule requires that all foreign keys' values reference valid primary key values. This means, for instance, that a user can't enter an order for a nonexistent customer. Jet supports the enforcement of referential integrity.

Jet 2.0 (as part of Microsoft Access 2.0) adds support for the optional *cascading* of updates and deletes to foreign keys. For example, if you've defined referential integrity with cascading updates between the Vendors and the Orders tables, and change the Vendor # of a vendor record from102 to 105, Jet will automatically change all related records referencing that Vendor # in the Orders table (see Figure 5). Support for this feature is rarely seen in desktop databases.

Figure 5. *Changes to Vendor# in the Vendors table are automatically cascaded to the Orders table.*

Table Validation Rules

In addition to the two general integrity rules, Jet supports the definition of table and column level validation rules. In relational database parlance, this is often referred to as *business rules* support. Once defined, validation rules will be enforced for any application that uses the database. This means that the database administrator can embed business rules *in the database itself* rather than in each and every application that uses the database. For databases shared by many applications, the savings in time spent programming and re-programming the same business rules can be considerable.

For each Jet table, the database developer may define a rule for each column in the table *and* an additional global rule that applies to the whole table. Column-level rules can only reference that column, but table-level rules can refer to multiple columns in the table. A rule for a table for tracking the orders for a mail order business, for instance, might require that all order dates be no greater than the current date. This type of rule would be associated with the OrderDate column. On the other hand, a rule that required the

shipping date to always be on or after the order date would be defined as a table-level rule.

The Jet Security Model

The Jet engine supports an advanced *workgroup-based security model* for controlling access to database objects. There are two basic components to the Jet security model:

- *User and Group accounts*
  User and Group accounts and their passwords are stored in a special database called the SystemDB database. This database (by default, SYSTEM.MDA) is typically centrally located on a network file server. A given SystemDB defines a workgroup made up of users and groups.
- *Object permissions*
  The access permissions for database objects (such as tables and queries) are stored in each database.

This sophisticated workgroup-based system has several advantages over simpler database-centric security systems. By separating account information from permission information, system administrators need to manage only one set of accounts for all databases on the network. In addition, by virtue of its support for two levels of accounts (user and group accounts), Jet makes it much easier to manage large numbers of users with similar permission levels.

With the introduction of Jet 2.0, security is now fully programmable (see “Data Access Objects” later in this paper for more details).

The Jet Query Engine

The Jet Query Engine, a component of the Jet Engine, is a complex query processor that is responsible for the interpretation and execution of queries. Jet processes queries in four steps:

- Definition
- Compilation
- Optimization
- Execution

Query Definition

Jet users can create queries using several different mechanisms. Within Microsoft Access, users can create queries using the Query By Example (QBE) interface. Users can also create queries using the Microsoft Access SQL Window. SQL Select statements can also be used by the Microsoft Access and Visual Basic developer to define the control source for certain types of controls such as list boxes and combo boxes. Finally, Access Basic developers can use the Data Access Objects (DAO) language to create a query definition (also known as a *QueryDef*) that uses a standard SQL Select statement.

Whatever the method chosen to create the query definition, Jet receives the query in the form of a SQL Select statement. The Select statement is passed to the Jet Query Optimizer, which then compiles and optimizes the query.

Figure 6. *The Jet Query Engine takes a query definition and compiles, optimizes, and executes it to produce a result set.*

Query Compilation

Before Jet can optimize the query, it must parse the SQL Select statement that defines the query and must bind the names referenced in the query to columns in the underlying tables. At this point, the syntax of the SQL statement is checked and an error is returned if any problems exist. The Jet Query Engine compiles the SQL string into an internal query object definition format, replacing common parts of the query string with tokens. The internal query object format can be likened to an inverted tree: the query's result set sits at the top of the tree (the tree's root) and the base tables are at the bottom (the leaves).

Query Optimization

Figure 7. *The Jet Query Compiler parses the SQL and compiles it into a query tree.*

The Jet Query Optimizer is one of the most complex components of the Jet Query Engine. The optimizer is responsible for taking the compiled query tree and choosing the optimum query execution strategy.

The Jet Query Engine uses a cost-based algorithm, costing and comparing each potential execution strategy and choosing the one that is the fastest.

Figure 8. *The Jet Query Optimizer determines the cost of various base table access plans and join strategies and calculates the best execution plan.*

There are two major operations associated with the execution of a query, each which must be assigned a cost: the reading of base tables and joins.

For each table in a query, the Jet Query Optimizer must choose from one of three different base table access plans: index range, Rushmore™ restriction or table scan. The decision as to which access plan to

use for a table depends on the size of the table, the presence of any criteria and how restrictive they are, and the presence of indexes.

For queries involving more than one table, the Jet Query Optimizer must also consider the cost of joins, choosing from five different join strategies: nested iteration, index, lookup, merge or index-merge.

Rushmore query optimizer

Jet 2.0 includes support for the Rushmore query optimizer. In Jet 1.x, only one index could be used to solve a query. Using Rushmore techniques borrowed from Microsoft FoxPro, Jet 2.0 can now use more than one index to restrict records. Rushmore-based query optimization is used on queries involving restrictions on multiple indexed columns of the following types:

- *Index Intersection*—the two indexes are intersected. Used on restrictions of the form
  " Where Company = 'Ford' and CarType = 'Sedan' "
- *Index Union* —the two indexes are unioned. Used on restrictions of the form
  " Where CarType = 'Wagon' or Year = '1994' "
- *Index Counts*—queries that return record counts only. Used for queries of the form
  " Select Count(*) from Autos Where Company = 'Dodge' and CarType='Truck'; "

Many queries can be executed much faster using the Rushmore query optimizer.

Query Execution

Once the optimum query execution plan has been determined by the optimizer, the Query Engine runs through the final query tree and executes each step to return the result set.

The Dynaset Model

Figure 9. *The final step in the processing of a query is the running of the best execution plan to produce the result set.*

In most commercial databases, from Paradox on the PC desktop to DB2® on IBM® mainframes, the results of queries are stored in temporary result sets, often called views or answer tables. Users can browse through the results of their queries using these views/answer tables, and, in most cases, can save the results as permanent tables. Any updates to answer table result sets, if allowed, are never reflected back in the original tables that were used to generate the query. (In some products, users can update the data in views if the queries that produced the views were based on a single table.) In contrast, Jet queries return *dynasets* —dynamic sets of data that are hot-linked back to the query's source tables. Updates to data in Jet dynasets are *automatically* propagated back to the original tables.

The idea behind dynasets—or the updatable view—is hardly new, but the fact that a commercial database product supports the idea is indeed revolutionary. Dr. E.F. Codd, the creator of the relational database model, wrote about the concept of closure—that operations on tables should return tables which can then become the input to other relational operations—over 20 years ago. Until the introduction of Microsoft Access and the Jet engine in 1992, however, no commercial database product so fully supported this important relational feature.

For the Jet user, the dynaset is significant because the user or developer can treat the results of queries as if they were tables themselves. This means that application developers using Microsoft Access, for example, can use dynasets as the source for data entry forms and other queries. By taking advantage of the updatability of dynasets, the application developer is saved from having to write hundreds of lines of application code that would have been necessary to implement the equivalent functionality using code

alone.

Dynasets versus Snapshots

Jet normally creates dynasets, but there are times when an application developer may prefer a non-updatable query result. For these occasions, Jet also supports an alternate recordset called a *snapshot*. Unlike dynasets, snapshots are static sets of records. Snapshots are never updatable.

In most situations, it is more efficient for the application developer to use dynasets because of the different way the two types of queries are treated by Jet. When Jet runs a dynaset-based query, it creates a set of unique key values sometimes called a keyset in memory that points back to the rows in the underlying tables. This *keyset-driven* cursor model is very efficient, since Jet only needs to read these key values and store them in memory (overflowing to disk if necessary). The values of the other columns in the dynaset aren't read until needed (such as when a user scrolls the datasheet to that screen of dynaset rows), minimizing the time needed to execute the query.

Figure 10. *The snapshot on the left is read-only, whereas the dynaset on the right can be edited, and changes will be propagated back to the original tables.*

For snapshot-based queries, Jet must run the query entirely to completion and extract all the query's columns into the snapshot. When the query contains many columns, it is likely that Jet won't be able to fit the entire snapshot into memory, requiring it to overflow the result set to disk, substantially slowing the query. On the other hand, since Jet reads only the key values of dynasets into memory, the same dynaset-based query might fit entirely in memory, resulting in a significant performance boost.

Result Set Population and Data Fetching

When Jet executes a query involving ODBC data, it must first populate the result set using a *population query*. As with regular Jet queries, the Jet engine treats queries returning updatable dynasets differently than those returning static snapshot result sets.

For a snapshot, Jet requests all the selected columns of the rows meeting the query's criteria from the server. In contrast, a dynaset is populated by a query that selects only the unique key column(s) of each qualifying row. In both cases, these result sets are stored in memory (overflowing to disk if necessary), allowing the user to scroll through the records arbitrarily.

When rows of data are needed (for example, when a user scrolls to a new screen of records), a snapshot has the data available locally. A dynaset, on the other hand, has only the key values and must use an additional query to ask the server for the data corresponding to the requested key values.

Performance Considerations

Jet's sophisticated query optimizer means that the end user and developer needn't be concerned about the order of columns and tables in the query definition. The Jet Query Optimizer decides upon the most efficient query strategy and reorders the query's tables and columns to best optimize the query. The user and application developer, however, can help the optimizer by following these simple guidelines:

- Create indexes on *all* columns used in query joins, restrictions and sorts.
- Use primary keys instead of unique indexes whenever possible. Primary key indexes disallow Nulls, giving the Jet Query Optimizer additional join choices.
- Use unique indexes instead of non-unique indexes whenever possible. Jet can then better optimize queries because statistics on unique indexes are more accurate.
- Include as few columns as needed in the result set. The fewer columns returned, the faster the query, especially if the user or developer can completely eliminate columns from a table that is only necessary for the restriction of records.
- Refrain from using complex expressions, such as those involving the IIf()

function, in queries. If using nested queries (queries based on the results of other queries), try to move any expressions up to the highest (last) query.

- Use Count(*) instead of Count([ColumnName]). Jet has built-in optimizations that make Count(*) much faster than column-based counts.
- Use the Between operator in restriction clauses rather than open-ended “>”, “>=”, “<”, and “<=” restrictions. This returns fewer rows. For example, use “Age Between 35 and 50” rather than “Age >= 35”.
- Normalize tables, decomposing large non-normalized tables into smaller normalized ones. Since this reduces the size of tables (and therefore the number of pages required to hold tables), it causes join strategies that involve table scans to execute more quickly.”

Avoiding “Query Too Complex” Errors

When compiling queries, the Jet Query Engine must fit the entire compiled query into a single 64K segment due to the 16-bit architecture of Jet. Because of this segment size limitation, it is possible for Jet to be unable to compile certain very large queries. The most common manifestation of this limit is the “Query Too Complex” error message seen by Microsoft Access users (the message was “Out of Memory” in Microsoft Access 1.x). When this error occurs, it may be possible to create an equivalent query that uses less memory, by following these guidelines:

- Reduce the number of output columns in the query result set. Don't include columns that aren't absolutely needed.
- Refrain from using complex expressions, such as those involving the IIf() function, in queries. If using nested queries, try to move any expressions up to the highest (last) query.
- Replace complex expressions with user-defined functions that perform the equivalent functionality.
- Use shorter table and field names.

Foreign Data

Traditionally, desktop database programs require that users with existing data import that data into the database program's native file format. Sometimes this process requires the conversion of data to some common file format before the import can be performed. In any case, this requires that the data be moved to the new format without any provision for coexistence with existing systems. While Jet (via Microsoft Access) supports this more traditional import process, it also supports the direct access of existing data stored in common file formats. This represents a tremendous advantage to the user or application developer who wishes to keep that data in its original file format, possibly sharing the data with existing legacy systems.

Installable ISAM Support

Jet supports the following external Index Sequential Access Method (ISAM) file formats:

- Microsoft FoxPro, versions 2.0, 2.5 and 2.6
- dBASE III PLUS and dBASE IV
- Paradox, versions 3.*x* and 4.*x*
- Btrieve, versions 5.1*x* and 6.0

Jet supports two methods of access to external ISAM files: *attached tables* and *direct table opening*. Once attached to a Jet database, the end user and application developer can (with a few exceptions) use external ISAM data as if it were native Jet data. Jet treats non-native ISAM data very similarly to how it treats native Jet data. Queries on non-native ISAM tables are compiled and optimized just as if the tables were internally stored Jet tables.

There are, however, some differences when using external ISAM tables. Index statistics, for example, are less likely to be accurate for external ISAM tables, because Jet has less control over the maintenance of these statistics. Also, some ISAM databases don't support Jet-style unique and primary key indexes. In addition, not all external ISAM indexes can be maintained by Jet, such as indexes with expressions that Jet doesn’t support. The net result is that the Jet Query Optimizer is usually less efficient when optimizing non-native ISAM tables.

Microsoft Access users can attach to external ISAM data using the Microsoft Access user interface. Microsoft Access and Visual Basic developers can also establish attachments using Jet's Data Access Objects (see “Data Access Objects”). For example, to attach to a Paradox 4.0 table named ORDERS.DB using DAO, the application developer might use the following commands:

```
Dim db as Database
Dim tdfOrders as TableDef

'Set a database object to point to the currently opened database
Set db = DBEngine.Workspaces(0).Databases(0)

'Create a TableDef object
Set tdfOrders = db.CreateTableDef("Attached Paradox Table")

'Set the connection information
tdfOrders.Connect = "Paradox 4.X;DATABASE=\\PDOX\\DATA\ORDERS"
tdfOrders.SourceTableName = "Orders"

'Append the TableDef to create the link
db.TableDefs.Append tdfOrders
```

There are times when the Microsoft Access or Visual Basic developer may wish to directly open an external ISAM table using the OpenDatabase command. For example, to open the same Paradox table

directly, the application developer could use the following commands:

```
Dim db as Database
Dim rst as RecordSet

'Open the external Paradox table directly
Set db = DBEngine.Workspaces(0).OpenDatabase("\\PDOX\\DATA\ORDERS", False, False,
  "Paradox 4.x")

'Open the recordset on the orders table
Set rst = db.OpenRecordset("Orders")
```

Xbase Data

Jet includes support for FoxPro 2.0, 2.5 and 2.6, dBASE III PLUS, and dBASE IV .DBF files. For FoxPro datafiles, Jet supports the use of .IDX and .CDX indexes. For dBASE datafiles, Jet supports the use of .NDX and .MDX indexes. Jet maintains information on which index files to utilize in a special information file with the extension .INF.

Jet maps Xbase datatypes to Jet datatypes as shown in the following table:

Character	Text
Numeric, Float	Number—Double
Logical	Yes/No
Date	Date/Time
Memo	Memo
General (FoxPro)	OLE Object

Options can be set to control the display of deleted records and the case-sensitivity for searches. These settings are stored in the [dBASE ISAM] section of MSACC20.INI.

Paradox Data

Jet supports Paradox 3.x and 4.x datafiles. Unless the Paradox tables have a primary key defined, the tables will be read-only.

Jet maps Paradox datatypes to Jet datatypes as shown in the following table:

Alphanumeric	Text
Number	Number—Double
Short Number	Number—Integer
Currency	Number—Double
Date	Date/Time
Memo	Memo
OLE	OLE Object (read-only)
Graphic	Not supported
Binary	Not supported
Formatted Memo	Not supported

Various options that affect the handling of Paradox data can be adjusted using options found in the [Paradox ISAM] section of the MSACC20.INI file.

Btrieve Data

Jet supports Btrieve version 5.1x and 6.0 datafiles. In order to use Btrieve datafiles, FILE.DDF and FIELD.DDF data definition files corresponding to the datafile must be present. In addition, the user must have a licensed copy of the Novell Btrieve for Windows DDL file (WBTRCALL.DLL). This DLL file is not supplied with Jet or Microsoft Access and must be obtained from Novell.

Jet maps Btrieve datatypes to Jet datatypes as shown in the following table:

String, lstring, zstring	Text
Integer (1-, 2- or 4-bytes), Autoinc (2- or 4-bytes)	Number—Byte, Integer or Long Integer
Float or bfloat (4-byte)	Number—Single
Float or bfloat (8-byte)	Number—Double
Money	Currency
Logical, bit	Yes/No
Date, time	Date/Time
Note	Memo
Lvar	Memo

Various settings can be adjusted in the [Btrieve]section of the WIN.INI file (for Btrieve 5.1x files) or the NOVDB.INI file (for Btrieve 6.0 files).

ODBC Data

Jet supports access to Open DataBase Connectivity (ODBC) data sources. The ODBC connectivity standard is typically used to connect to server-based back ends. Microsoft Access and Visual Basic ship with ODBC drivers for the following client-server databases: Microsoft SQL Server, Sybase® SQL Server, and Oracle® Server (available in the ODBC Driver Fulfillment Pack, offered at nominal cost via a coupon in Microsoft Access 2.0).

Armed with additional drivers from Microsoft, server vendors, and third-party connectivity vendors, the application developer has virtually unlimited connectivity options. ODBC drivers are available for ORACLE, DB2®, Rdb® and many other server databases on a variety of platforms. The ODBC standard can also be used to connect to non-server databases and spreadsheets.

Jet supports three methods of access to ODBC data sources: attached tables, direct opening of the tables, and SQL Pass-through queries.

The use of attached tables is preferable and they will perform better than the direct opening or pass-through methods. Unlike non-native ISAM tables, the Jet Query Optimizer attempts to offload as much query processing as possible to the server when executing queries against ODBC tables (see the earlier discussion in "Query Optimization").

Jet maps ODBC datatypes to Jet datatypes as shown in the following table (lPrecision is the ODBC precision of a column; wScale is the ODBC scale of a column):

SQL_BIT	Yes/No
SQL_TINYINT SQL_SMALLINT	Number—Size: Integer
SQL_INTEGER	Number—Size: Long Integer
SQL_REAL	Number—Size: Single
SQL_FLOAT SQL_DOUBLE	Number—Size: Double
SQL_TIMESTAMP SQL_DATE	DateTime
SQL_TIME	Text
SQL_CHAR SQL_VARCHAR	if lPrecision <= 255, then Text (Field Size = lPrecision) if lPrecision > 255, then Memo
SQL_BINARY	if lPrecision <= 255, then Binary (Field Size = lPrecision)

SQL_VARBINARY	if lPrecision > 255, then OLE Object
SQL_LONGVARBINARY	OLE Object
SQL_LONGVARCHAR	Memo

SQL_DECIMAL
SQL_NUMERIC
	if wScale = 0, then
	if lPrecision <= 4, then Number—Size: Integer
	if lPrecision <= 9, then Number—Size: Long Integer
	if lPrecision <= 15, then Number—Size: Double
	if wScale > 0, then
	if lPrecision <= 15, then Number—Size: Double
	Special cases for SQL Server/Sybase:
	if lPrecision = 19 and wScale = 4, then Currency
	if lPrecision = 10 and wScale = 4, then Currency

Any other field types mapped to Text (Field Size = 255).

An analogous set of mappings occur when going the other way (converting Jet datatypes to ODBC datatypes).

Connection Management

Jet shares ODBC connections as much as possible. This is important since many servers put limits on the number of concurrent connections that can be active at a given time. Two connections can be shared if they have the same data source name (DSN) and DATABASE parameters in their connect strings. If neither connect string has a DATABASE parameter, then only the DSN must match. To avoid constantly disconnecting and reconnecting, Jet maintains server connections even when they aren't explicitly in use. This is transparent to the user.

Pass Through Queries

Jet 2.0 introduces the concept of the SQL Pass-through (SPT) query. (Microsoft Access 1.x supported pass-through queries through the use of a separate DLL.) This type of query is an arbitrary SQL string and an ODBC connect string that Jet passes directly to the server without interpretation. Pass-through queries are useful for:

- Selecting data from tables using server-specific syntax.
- Changing data in server tables.
- Calling stored procedures.
- Performing administrative tasks, such as adding and deleting user accounts.

Any result sets returned by SPT queries are stored as snapshots.

Performance Considerations

Snapshots and dynasets differ significantly in several performance characteristics. Dynasets are usually more efficient, especially for large result sets. On the other hand, snapshots may be faster when dealing with relatively small result sets that don't contain memo or OLE object fields. Snapshots, however, cannot be used where data needs to be updated.

Most of the guidelines discussed earlier in the Jet Query Engine “Performance Considerations” section are also applicable to ODBC queries.

Application developers and users must be aware of Jet-supported functionality that is not supported on the server. When this non-server functionality is requested (for example, Jet-specific extensions to SQL, queries involving Access Basic operators or functions or user defined functions, or Microsoft Access reports involving multiple levels of grouping), Jet must execute the query locally.

Data Access Objects

Users and application developers can access Jet engine data by using the standard user interface tools provided by application programs. For example, Microsoft Access users and developers can use forms, datasheets and queries to manipulate data. Similarly, the Visual Basic programmer can gain access to Jet data by using the Data Control in Visual Basic. These tools make it easier for the user to manipulate data, but they also limit the application developer's options. Often, the application developer must go beyond these interface tools and manipulate the data more directly. Data Access Objects (DAO) provide this more direct interface to the Jet database engine.

DAO provides a consistent, object-based and programming language-independent language for the definition and manipulation of Jet engine data, including table and query definitions, recordsets, and security. Although currently implemented in Access Basic and Visual Basic, DAO will be available to developers who use Visual C++™ and Visual Basic for Applications some time in the future.

DAO uses a sophisticated containership model that establishes a hierarchy for each class of Jet object. The object hierarchy available to Microsoft Access 2.0 developers is depicted in Figure 11.

Figure 11. *The Jet Data Access Object containership hierarchy available to Microsoft Access developers.*

Properties, Methods, and Collections

Objects in the DAO hierarchy have properties, methods and collections. *Properties* are attributes of objects that can be retrieved and (sometimes) set. For example, RecordSet objects have a property called Updatable. Depending upon the type of recordset (table, dynaset or snapshot), the Updatable property will be true or false. Properties can be thought of as the adjectives that describe objects.

Methods are actions that can be applied to objects. For example, the User object has a CreateUser method used to create a new user account. Methods can be thought of as the verbs that act upon objects.

Some objects have *collections* of other objects. For example, a TableDef object has two collections: fields and indexes that correspond to the fields and indexes for a table. Collections can be thought of as the possessions of objects.

Data Definition Language

DAO includes a set of properties and methods for creating objects. In database parlance, this component of a language is often called the *data definition language* or DDL. The data definition language elements of DAO can be used to programmatically create tables, queries, and relationships “on the fly.”

For example, to define a new table with DAO, the application developer would create a TableDef object and each of the fields in the table and then append the fields to the TableDef object. Next, the developer would append the table's primary key and any other indexes to the TableDef object. Finally, the developer would append the TableDef object (which now contains all of the fields and indexes) to the Database object to create the new table. Using Microsoft Access, the application code to create a table named Orders, containing four fields OrderId, OrderDate, CustomerId, and SalesId, with OrderId as the primary key, would look like this:

```
Dim db as Database
Dim tdfOrders as TableDef
Dim fldOrderField as Field
Dim idxOrderIndex as Index

'Set a database object to point to the currently opened database
Set db = DBEngine.Workspaces(0).Databases(0)

'Create a TableDef object
Set tdfOrders = db.CreateTableDef("Orders")

'Create the four fields
tdfOrders.Fields.Append tdfOrders.CreateField("OrderId", DB_LONG)
tdfOrders.Fields.Append tdfOrders.CreateField("OrderDate", DB_DATE)
tdfOrders.Fields.Append tdfOrders.CreateField("CustomerId", DB_INTEGER)
tdfOrders.Fields.Append tdfOrders.CreateField("SalesId", DB_INTEGER)

'Create the primary key for the table
Set idxOrderIndex = tdfOrders.CreateIndex("PrimaryIndex")
idxOrderIndex.Primary = True
idxOrderIndex.Required = True
idxOrderIndex.Unique = True

idxOrderIndex.Fields.Append idxOrderIndex.CreateField("OrderId")
tdfOrders.Indexes.Append idxOrderIndex

'Final step is to append the TableDef to the database
db.TableDefs.Append tdfOrders
```

Data Manipulation Language

DAO also contains properties and methods for manipulating data. This component of a language is often called the *data manipulation language* or DML. The data manipulation language elements can be used to add new records to a table, edit and delete records, and query the database using queries and SQL statements.

For example, the following application code could be used to define and execute a query that selected all Order records for a customer with a CustomerId of 10 using Microsoft Access:

```
Dim db as Database
Dim qdfOrderCust10 as QueryDef
Dim rstOrderCust10 as RecordSet
Dim strSQL as String

'Set a database object to point to the currently opened database
Set db = DBEngine.Workspaces(0).Databases(0)

'Create a querydef object called OrderCust10
Set qdfOrderCust10 = db.CreateQueryDef("OrderCust10")

'Create the querydef's SQL string
strSQL = "SELECT Orders.* FROM Orders WHERE CustomerID = 10;"
qdfOrderCust10.SQL = strSQL

'Run the query by opening a record set on the querydef
Set rstOrderCust10 = qdfOrderCust10.OpenRecordSet()
```

Programmatic Security

Data Access Objects can be used to manipulate Jet security. Using DAO, the database administrator or application developer can do any of the following:

- Create new User accounts.
- Create new Group accounts and add users to them.
- Modify and delete existing User and Group accounts.
- Add and subtract permissions on both engine objects (such as tables, queries and relationships) and application objects (such as forms, reports, macros and Access Basic modules).
- For a given user or group, check the permissions on an object.
- Change a user password.
- Change ownership for an object.

Of course, the above activities require the currently logged-in user to have a certain security clearance level. For example, to be able to change the ownership of an object, the user must be either the current owner or have administrator-level security status.

Summary

The Microsoft Jet Database Engine is a sophisticated multiuser desktop relational database engine supporting engine-enforced referential integrity, business rules, workgroup-based security and remote data access to ISAM and server-based data. The Jet engine supports two-way updatable views (dynasets) and includes a sophisticated cost-based query optimizer. It also supports an object-based and programming language-independent data access language.

Jet is a modern, next-generation database engine. Today, Jet supports many advanced features that are seldom seen in desktop database engines, but it is also architected with the future in mind, making it *the* database engine for the next decade and beyond. Possible enhancements to look for in future versions include trigger and OLE 2.0 Automation Server support.

Glossary

Attached table—A remote table that can be accessed as if it were stored locally.

Cascade—A referential integrity option that specifies that changes or deletions to the primary key in one table will be propagated to any other tables that reference that primary key value.

Collection—A group of objects.

Cursor—A method of accessing rows of data. Jet supports dynaset and snapshot cursors.

Data Access Objects (DAO)—The programming language-independent object-based data access language used to manipulate Jet engine data.

Database engine—A program (or part of a program) that serves as the link between a DBMS or application and the data. The part of a DBMS program that reads and writes data records.

Database Management System (DBMS)—A program that is used to manage data.

Database statistics—Information stored in a database that can be used to optimize queries. In Jet, database statistics come from the indexes.

Dynamic Link Library (DLL)—A program that can be dynamically called from another program.

Dynaset—A dynamic (two-way) updatable query result set. A type of cursor.

Entity integrity—A relational database rule that requires primary key values to never be null.

Foreign key—A reference to the primary key in another table.

Indexed Sequential Access Method (ISAM)—A physical method for storing data in a database. Also used to refer to non-server databases.

Join—A connection between two tables in a relational database.

Keyset—A set of unique (usually primary key) values that can be used to reference a specific row in a table.

Lock—A way to prevent two users from modifying the same piece of data at the same time.

Method—An action that can be applied to an object. For example, to move to the first row of a recordset, you apply the MoveFirst method to the RecordSet object.

Microsoft Access—A relational desktop database management system from Microsoft. Runs under Windows and uses the Jet engine.

Null—The state of lacking a value. A field in a database has “null” value before it is assigned a value.

Object Linking and Embedding (OLE)—A standard method to link and embed objects created by one program to another program. Also, a type of field in Jet used to store complex objects created by other programs.

Objects—A package of things created and manipulated by programs. In Jet, Tables, Users and Query Definitions are all examples of objects.

OLE automation—A standard set of properties and methods that can be used to programmatically manipulate objects created by other programs. Part of the OLE 2.0 standard.

Open DataBase Connectivity (ODBC)—A standard way to connect to and read and write records from other databases, usually server databases.

Optimistic locking—A way to share data in a multiuser environment that only locks record pages when the record is saved. An optimistic lock is set for a short period of time.

Page lock—A lock applied to a page-size group of records. Jet uses 2K (2,048 byte) size pages to lock

records.

Pessimistic locking—A way to share data in a multiuser environment that locks record pages whenever a user starts editing a record. A pessimistic lock is set for a longer period of time than an optimistic lock.

Primary key—A column (or multiple columns) in a table that ensures that a record can be uniquely identified.

Property—An attribute of an object that can be retrieved and (sometimes) set. For example, the index property of a table can be set to the name of a one or more columns in the table.

Query By Example (QBE—A standard visual method of defining queries.

Query compilation—The process of parsing a query definition and compressing it into a format that can be used by the query optimizer.

Query definition—A way of representing a query. With Jet, the query definition can be represented by QBE or SQL.

Query execution—The process of running a query to produce a result set using the execution plan determined by the Jet Query Optimizer.

Query optimization—The act of calculating the most efficient execution plan for a query definition.

Query tree—A way to internally represent a compiled query definition that can be likened to an inverted tree.

Referential integrity—A relational database rule that requires that all foreign key values reference valid primary key values.

Relational database—A type of database consisting of tables and (more or less) following a set of rules established by E.F. Codd (the creator of the relational database model).

Snapshot—A static temporary result set of records produced by a query. Not updatable. Similar to a Paradox "answer table." A type of cursor.

Structured Query Language (SQL)—A standard method of querying data stored in relational databases.

Transaction—A sequence of actions that must occur as a single unit.

Transaction processing—A mode of database processing that supports the creation, and saving (commit) or undoing (rollback) of transactions.

Trigger—A complex procedure that can be linked to a change of data so that it always occurs when a certain type of data modification is made.

Validation rule—An expression that can be linked to a change of data so that it is always evaluated when a certain type of data modification is made. A simple form of a trigger.

Visual Basic—Microsoft's visual programming dialect of BASIC. It includes support for the Jet Engine.

Visual Basic for Applications—A new application programming language from Microsoft that uses the Visual Basic language in other Microsoft Office applications such as Microsoft Excel.

Visual C++ —Microsoft's visual programming dialect of the C and C++ programming languages.

Suggested Readings

"Jet Database Engine ODBC Connectivity White Paper," Neil Black & Stephen Hecht, Microsoft (Part No. 098-53349). *[Can be downloaded from the MSACCESS forum on CompuServe®.]*

"Get Better Performance with Your Single-table Queries," Steve Brandli, *Smart Access™*, July 1993, Pinnacle Publishing. *[Can be purchased by calling Pinnacle Publishing at 800-788-1900 or 206-251-1900.]*

"Multi-table Query Optimization," Steve Brandli, *Smart Access*, October 1993, Pinnacle Publishing. *[Can be purchased by calling Pinnacle Publishing at 800-788-1900 or 206-251-1900.]*

Microsoft Access Advanced Topics Manual, Microsoft. Chapters 3, 4, 7, and 8. *[Can be purchased as part of the Microsoft Access Developer's Toolkit.]*