Migrating DBMS Management from Microsoft Access Applications to Oracle 7.1 RDBMS Server

White Paper

Oracle Corporation, Inc.

December, 1995

Introduction

Accurate and timely data is essential for business operations in the 1990's. Faster computers along with easy-to-use database software has led to the creation of personal and departmental databases which manage important business data. Products such as Microsoft Access allow developers and advanced users to build complete business systems. Microsoft Access is based on file sharing technology. Because of this, Access lacks the speed, reliability and robustness provided by an independent RDBMS server.

Using an RDBMS server with Access in a client–server architecture uses the strengths of both technologies. Access provides excellent form and report systems, as well as a complete programming language (Access Basic). An independent relation database management system (RDBMS) provides reliable, robust, and secure high speed data management.

Oracle 7.1 RDBMS Server, from Oracle Corporation, is a modern, scaleable, high performance database server which can run on wide range of computers from PCs to mainframes. Oracle 7.1 operates in a networked, client–server environment and can support tens, hundreds or thousands of users depending on the server computer.

This paper, "Migrating DBMS Management from Microsoft Access Applications to Oracle 7.1 RDBMS Server", explains how to use Microsoft Access and Oracle 7.1 together in a client–server architecture. If you have an investment in Access applications, you can retain this investment while adding the advanced features of Oracle 7.1 RDBMS to you application architecture.

After reading this paper, you will know how to move a working Access application to an Access / Oracle architecture. In addition, you will know how to extend your application using Access or other client development tools such as Oracle Power Object or Oracle's Developer/2000 tool set.

The subjects covered in this paper also apply to Microsoft Visual Basic (VB) applications which use the Jet DBMS engine. Visual Basic applications can be moved to Oracle as easily as Access Applications.

Assumptions

This paper assumes that you are familiar with Microsoft Access; advanced topics assume that you are familiar with writing Access Basic programs. This paper also assumes that you know how to create triggers on Oracle 7.1.

If you have Access installed on your personal computer and you have access to Oracle 7.1 from your personal computer you can experiment with the concepts discussed in this paper. To write stored procedures and triggers on Oracle you can use Oracle's SQL*Plus utility.

Topics Covered

This paper contains three sections and several appendices.

"The Access / Oracle Client–Server Architecture" provides an overview of the major components involved in coupling Access to Oracle.

"Migrating Access Applications to Oracle RDBMS Server" reviews the steps required to move the data management portion of an Access application to Oracle 7.1 RDBMS.

"Extending your Application" explores ways in which the Access / Oracle application can be extended using other Oracle development tools.

The appendices include detailed information about the Access / Oracle architecture. The information in the appendices will be helpful if you are interested in the details of how Access operates with Oracle or if you want to tune your system for maximum performance.

The Access / Oracle Client–Server Architecture

Access Architecture

Microsoft Access is based on a file server DBMS technology named "Jet". As shown in Figure 1, Access' Forms, Reports and Basic code rely on Jet to manage data stored in a the native "mdb" file format.

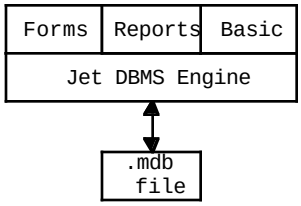

Figure 1 – Access Architecture.

In a single–user Access application, the mdb file, as well as Access itself, are located on the same machine. In a multi–user Access application, the mdb file is placed on a file server and shared. Each client runs a copy of Access and the Jet engine. In this configuration, Jet must move a large amount of data across the network, including whole tables, to complete its query processing.

Jet Recordsets

When Jet processes a query, it returns a Recordset (a cursor) for the resulting set of records. Jet supports two types of Recordsets, Snapshots and Dynasets.

A Snapshot is a 'picture' of data, as it existed at the time the query is run. When returning a snapshot, Jet runs the query to completion, extracts all of the resulting rows and columns into a virtual table and presents this virtual table to the user. The user of a Snapshot is able to perform a full range of operations on a snapshot, query the Snapshot, base Forms and Reports on the Snapshot, etc. Changes cannot be made to Snapshot data and changes made by others users after a Snapshot has been opened are not reflected in the Snapshot.

A Dynaset, on the other hand, is a 'live' view of the data. When returning a Dynaset, Jet extracts the "key values" from the data and stores these in memory. When a user requests rows of data from the Dynaset, Jet fetches the rows of interest by looking them up in the base tables via the internally stored key values. Once a Dynaset is opened, the set of key values cannot change. This means that while the data pointed to by the "key value" may change and will be reflected to the user, new rows added after the query is started will not be a part of the set of key values and will not be made visible to the user. Rows which are deleted after the keyset query is run are still part of the set of key values, however they are marked #DELETED# when presented to the user .

The Dynaset model is a powerful and flexible model which gives the user of PC based information the opportunity to browse large quantities of data and update the data at will. When used with local data, Dynasets are fast and effective. *However, the Dynaset model presents one of the key performance challenges when Access works with a RDBMS server such as Oracle.*

Jet Multi–User Updates

Jet handles updates by multiple users using two methods - optimistic and pessimistic locks[1].

Using pessimistic locking, Jet will place a hard lock on the data page[2] which contains the row being edited. Other users will not be able to start editing the locked row until the lock is abandoned or the changes are written to disk.

---

[1] When working with native data, Jet supports both optimistic and pessimistic locking. However, Jet support only optimistic locking with ODBC and Oracle.

Jet employs an optimistic locking scheme when working with Oracle 7.1 RDBMS. An optimistic locking scheme does not place hard lock on the source table(s). Instead, when a change is to be committed, Jet checks to make sure that the data has not been altered by another user before allowing the changes to be posted.

Jet Enforced Referential Integrity

Jet supports declarative referential integrity. This includes Primary Key / Foreign Key relationships with one–to–one and one–to–many cardinality with cascading updates and deletes.

Jet / ODBC / Oracle Architecture

Using Oracle 7.1 RDBMS Server with Access can increase the robustness and reliability of a multi–user system. Network traffic is reduced because only the query requests and the resulting data are sent over the network (instead of complete tables). Jet technology is focused on single–user performance with adequate multi–user capabilities; Oracle 7.1 RDBMS on the other hand is mature central server technology focused on multi–user performance, rollback and recovery, and centralized query processing.

Obtaining adequate performance from the combination of Access and Oracle requires an understanding of how Jet works with centralized servers.

Figure 2 shows that Access requires ODBC[1] to make its connection to Oracle 7.1 RDBMS.

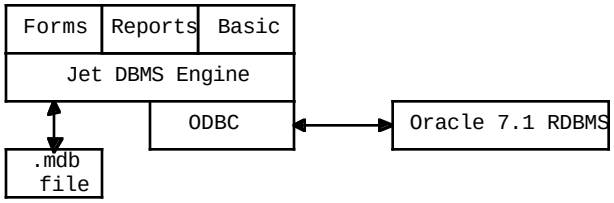

Figure 2 – Access / ODBC / Oracle Architecture.

ODBC is an API that allows client applications to connect to different RDBMS servers. Jet has been designed to make efficient use of ODBC while requiring a level 1 ODBC driver[2]. When Access uses ODBC to connect to remote RDBMS servers, Jet continues to function as the DBMS engine for Access. Access Forms, Reports and Basic code continue to work with Jet as if they were working with local or shared data in the mdb file format. Jet presents remote Oracle tables as attached tables[3]. These attached tables are created at design time and appear to be local tables.

Because of the way Jet works with remote tables, Jet requires a primary key on tables in Oracle in order to support Dynasets against those tables. If a remote table does not have a primary key, Jet will only open a non–updateable Snapshot on the table.

---

[2] Jet uses page locking with a page size of 2k bytes. If more than one record fits in a page, all records in the page are locked simultaneously.

[1] Open Database Connectivity (ODBC) is an industry standard API which allows client applications to connect to and use different DBMS systems. Access uses ODBC to connect to all DBMS Servers such as Oracle 7.1, SQLServer, DB2, etc.

[2] Please refer to Appendix 5 for details about ODBC, ODBC drivers, compliance levels, ODBC drivers for Oracle available from different vendors, etc.

[3] Jet also supports dynamic table connections. These are slow to establish and are not recommended for most applications. Please refer to the Appendices for details.

Migrating Access Applications to Oracle RDBMS Server

Overview

This section describes how to move the data management portion of a working Access application to Oracle. By following the steps in this section you can create an Access / Oracle client–server version of your application.

To move the data management portion of an Access application to Oracle you must first separate the data and declarative referential integrity portion of the application from the forms, reports and code in the application. After this is done, the database structure and data is moved to Oracle. Counter field equivalents are created in Oracle and the referential integrity support is added. Finally the remote tables must be attached to Access and the application can then be tested.

After you are comfortable with the migration process, please refer to the Appendices for more information including sample Oracle trigger code, information on security, and performance tuning suggestions.

Migration in Detail

Access applications can be contained in a single mdb file or multiple mdb files. Multiple mdb files are used to separate the data and application (form, report, code) portions of an application. Generally the data mdb file is shared on a central server and the application mdb file is located on each client machine.

When an application is in a single mdb file, you will need to take steps to separate the data from the rest of the application (forms, reports, code, etc.).

Remember to make backup copies of your Access application and data files before performing any of the following steps!

Preparing a Single mdb File Application for Exporting

Follow these steps to create two mdb files (one containing data, the other your application) from a single mdb file. These steps assume that you are starting with a file called app.mdb; at the end of these steps you will have app.mdb and data.mdb.

1] Make a backup of app.mdb.
2] Start Access and use the File…Compact Database menu to compress app.mdb.
3] In File Manager, copy app.mdb to data.mdb.
4] Open data.mdb in Access, delete all forms, reports, modules and macros.
5] In Access, open app.mdb and delete all the tables.
6] While still in app.mdb use File…Attach Table to make an attachment in app.mdb to each table in data.mdb.

At this point your application should run as it did before you split it into two mdb files[1]. You are now ready to export the data.mdb file to Oracle.

[1] If the application opens a table directly (not using a Dynaset or Snapshot) it will not work with attached tables. This restriction also applies after the data is moved to Oracle. If you encounter this situation, you may wish to leave some tables in the app.mdb file so that each client has an independent copy. This could be appropriate for tables with look–ups values such as a ‘State’ table. If you must move a table which is open directly to the data.mdb file, the application must be changed to use Dynasets or Snapshots.

Exporting Access Data to Oracle

Figure 3 illustrates an Access application before and after migration to Oracle. Both before and after migration, app.mdb contains the forms, reports, macros and Basic modules which make up the application.

Before migration, app.mdb contains an attached table from data.mdb. (We will use <tablename> to refer to the name of this table. Substitute the name of your actual tables, for example, “Customers”.)

After migration, app.mdb will have two attached tables for each original table and a name mapping query. The original <tablename> from data.mdb will be attached as L_<tablename> (e.g., L_Customers). The original table will be exported to Oracle and on Oracle will be called O_<TABLENAME>[1] (e.g. O_CUSTOMERS). An attachment will be created to O_<TABLENAME> inside Jet and will be named R_<tablename>.

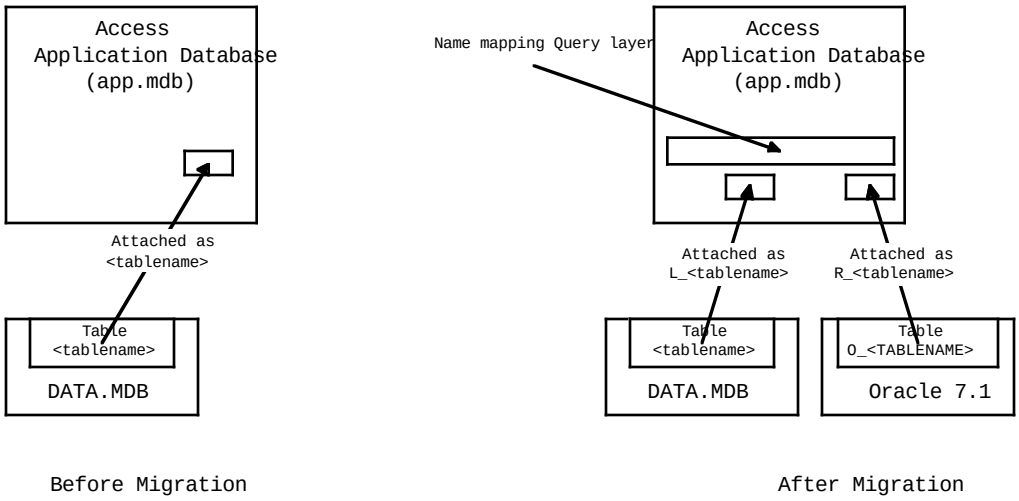

Figure 3 – Access Application before and after Migration to Oracle.

Because the forms, reports and modules in app.mdb are expecting a table called <tablename> a name mapping query called <tablename> is added. This query will take the place of the original table in the application. This query can refer to either L_<tablename> or R_<tablename>; you can switch between the local and remote table as you move your application to Oracle. The name mapping query layer also helps resolve reserved word conflicts between Access and Oracle. For example, a column named “Sequence” must be renamed (e.g., O_Sequence) as this is an Oracle reserved word. The name mapping query can re-map O_Sequence back to Sequence for use by the Access application.

You will also need to move any referential integrity definitions to Oracle and install triggers to support counter fields. In the next section, a detailed set of steps are described to lead you through this process.

[1] It is best to use all uppercase letters in Oracle.

Migration, Step by Step

The following sequence of steps will guide you as your migrate to Oracle. They assume that you are starting with separate application and data "mdb" files named app.mdb and data.mdb.

1] Make backup copies of app.mdb and data.mdb.
2] Insure that an ODBC connection exists to your Oracle database.
3] In app.mdb, rename each attached table to be prefixed with "L_" (e.g. rename <tablename> to L_<tablename>.)
4] Open data.mdb inside of Access and prepare each table for migration:
   a) Make a copy[1] of the table, naming the new table O_<TABLENAME>. That is, "O_" prepended to the table name, in all capital letters[2].
   b) Make sure that the column names follow the requirements for names in Oracle[3]. Change all column names to upper case.
   c) Select the File…Export Menu in Access.
   d) Follow the dialogs and these steps to export each table:
      i Select export to <SQL Database>.
      ii Select the table to export.
      iii Name the Oracle table (use O_<TABLENAME>).
      iv Select your ODBC DSN.
      v Supply your logon information.

      If you have checked for Oracle reserved words, there should be no errors to this point. If your tables are large, it could take many minutes to several hours to export each table. (Access provides a percentage complete indicator while exporting each table).

   e) Delete "O_<TABLENAME> from the data.mdb file.
6] Close the data.mdb file.
7] Use File…Compact with data.mdb to recover the space used for the table copies.
8] Open app.mdb file.
9] Attach each O_<TABLENAME> table from Oracle to app.mdb using the following steps:
   a) Select File…Attach Table menu.
   b) Select <SQL Database>.
   c) Select your ODBC DSN.
   d) Supply your Oracle Logon information.
   e) Select the table and press the Attach button.
10] Change the names of the attached tables from O_<TABLENAME> to R_<tablename>.
11] Create a mapping query for each attached Oracle table. The name of the query will be <tablename>, the original name of the table, as seen by your application. Make sure that the name of each column is mapped back to the original name found in the original tables[4].
12] Open the attached tables in datasheet view or open a form on the tables to make sure that the exporting and mapping steps have been successful. You will notice that you are not able to update the data in the tables. After you complete the migration steps and build primary keys, you will be able to modify your data.
13] For each column which was derived from an Access Counter field (counter fields are mapped to NUMBER(10,0)) perform the following steps[5]:

---

[1] If you do not want to have a local copy of the original data or if you do not have room for a copy, you can rename the tables.

[2] Some client software (e.g., Microsoft Query and others) only works if all table and column names in Oracle are in capital letters. It is suggested that you use only capital letters on Oracle.

[3] Refer to Appendix 1 - Oracle Reserved Words.

[4] See Appendix 2 for an example.

[5] Use Oracle's SQL*Plus utility for creating triggers, packages, packages, sequences, etc. on Oracle 7.1 RDBMS.

a) Create a Sequence for each counter. You are free to choose any starting number and increment number, however you must insure that you do not overlap numbers that have been exported from Access. You might want to start the sequence at the next major increment. For example, if 258 is the largest value in a counter field, you might want to start at 1000; this will make it clear which records were inserted after the move to Oracle.
b) Create trigger code for the counter field which uses the sequence[1].
c) Make the 'counter field' column the Primary key or at least create a unique index on the column; you must insure that this index is the one selected by Access as the 'key value' index, so you may want to prepend "aaaa" to the name of the index.

15] Create primary key and foreign key definitions in Oracle to match the Access structure. Oracle supports declarative Cascade Delete relationships, but not Cascade Updates. To support Cascade Updates, you will need to write trigger code in Oracle[2].
16] Map default value definitions to Oracle[3].
17] Map row and table validation statements to Oracle CHECK statements[4].
18] If you have any tables in Oracle which you need to update from Access and which do not, at this time, have a primary key, you must define one. As in step 13 above, you must make the primary key index the first index in alphabetic order for the table.
19] Finally, use the Access Attachment Manager to refresh the attached table connections. This insures that Jet caches the latest information about primary keys and other table parameters. After you complete this step you will be able to update your tables.
20] Open the tables in app.mdb in a datasheet or form to insure that the migration was successful.

---

[1] Refer to Appendix 2 for a code example.

[2] Refer to Appendix 2 for a code example.

[3] Oracle does not support the Access function set. {*** Geeta - do we want to include chapter on the mappings, tell them how to get the packaged support on Oracle or what?}

[4] Oracle does not support the Access function set. {*** Geeta - do we want to include chapter on the mappings, tell them how to get the packaged support on Oracle or what?}

Extending your Application

After you move the data management portion of your Access application to Oracle you can rely on Oracle 7.1 RDBMS to protect your data, maintain all referential integrity and business rules that you have encoded in PL/SQL.

With this foundation, you can extend your application with Access or a wide range of other tools. Oracle offers several high productivity tools such as Oracle Power Objects, the Developer/2000 toolset, and Oracle Objects for OLE. Oracle Objects for OLE (OO4O) is a high performance connectivity solution for Visual Basic, Delphi and other client tools which can control OLE Automation Servers.

In addition, if your application grows, you can move your Oracle server to larger computers without changing your application.

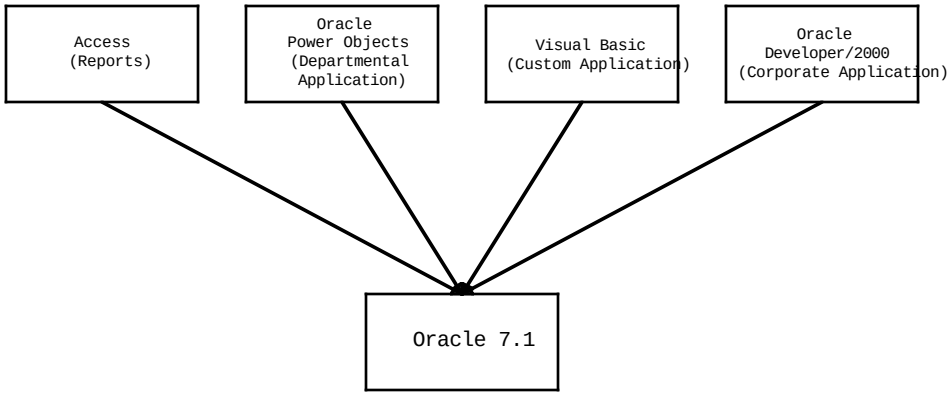

Figure 4 - Extending the Application with a Mix of Client Tools.

Appendices

Appendix 1 - Oracle Reserved Words

The following list of Oracle reserved words are from the Oracle RDBMS Server Manual and the PL/SQL Users Manual. In forming Oracle names the valid character set includes the upper and lower case letters (A..Z, a..z) and $, _, # and numerals (0..9) after the first character. Unlike Access, spaces are not allowed.

It is suggested that all Oracle identifiers be exclusively upper case. While Oracle supports lower case identifiers, some client tools required that identifiers be surrounded by quotes if an identifier contains a lower case letter.

ABORT
ACCEPT
ACCESS
ADD
ALL
ALTER
AND
ANY
ARRAY
ARRAYLEN
AS
ASC
ASSERT
ASSIGN
AT
AUDIT
AUTHORIZATION
AVG
BASE_TABLE
BEGIN
BETWEEN
BINARY_INTEGER
BODY
BOOLEAN
BY
CASE
CHAR
CHAR_BASE
CHECK
CLOSE
CLUSTER
CLUSTERS
COLAUTH
COLUMN
COLUMNS
COMMENT
COMMIT
COMPRESS
CONNECT
CONSTANT
COUNT
CRASH
CREATE
CURRENT
CURRVAL
CURSOR
DATA_BASE
DATABASE
DATE
DBA
DEBUGOFF
DEBUGON
DECIMAL
DECLARE
DEFAULT
DEFINITION
DELAY
DELETE
DELTA
DESC
DIGITS
DISPOSE
DISTINCT
DO
DROP
DUAL
ELSE
ELSEIF
END
ENTRY
EXCEPTION
EXCEPTION_INIT
EXCLUSIVE
EXISTS
EXIT
FALSE
FETCH
FILE
FLOAT
FOR
FORM
FROM
FUNCTION
GENERIC
GOTO
GRANT
GROUP
HAVING
IDENTIFIED
IF
IMMEDIATE
IN
INCREMENT
INDEX
INDEXES
INDICATOR
INITIAL
INSERT
INTEGER
INTERSECT
INTO
IS
LEVEL
LIKE
LIMITED
LOCK
LONG
LOOP
MAX
MAXEXTENTS
MIN
MINUS
MLSLABEL
MOD
MODE
MODIFY
NATURAL
NEW
NEXTVAL
NOAUDIT
NOCOMPRESS
NOT
NOWAIT
NULL
NUMBER
NUMBER_BASE
OF
OFFLINE
ON
ONLINE
OPEN
OPTION
OR
ORDER
OTHERS
OUT
PACKAGE
PARTITION
PCTFREE
POSITIVE
PRAGMA
PRIOR
PRIVATE
PRIVILEGES
PROCEDURE
PUBLIC
RAISE
RANGE
RAW
REAL
RECORD
RELEASE
REM
RENAME
RESOURCE
RETURN
REVERSE
REVOKE
ROLLBACK
ROW
ROWID
ROWLABEL
ROWNUM
ROWS
ROWTYPE
RUN
SAVEPOINT
SCHEMA
SELECT
SEPARATE
SESSION
SET
SHARE
SIZE
SMALLINT
SPACE
SQL
SQLCODE
SQLERRM
START
STATEMENT
STDDEV
SUBTYPE
SUCCESSFUL
SUM
SYNONYM
SYSDATE
TABAUTH
TABLE
TABLES
TASK
TERMINATE
THEN
TO
TRIGGER
TRUE
TYPE
UID
UNION
UNIQUE
UPDATE
USE
USER
VALIDATE
VALUES
VARCHAR
VARCHAR2
VARIANCE
VIEW
VIEWS
WHEN
WHENEVER
WHERE
WHILE
WITH
WORK
XOR

Appendix 2 - Code and Query Samples

The following sections contains samples which support counter field emulation, cascade update referential integrity, and name mapping queries.

Counter Datatype Emulation

Access supports a counter datatype. A counter provides a monotonically increasing sequence of Long Integers for a column in a native Jet DBMS file. Oracle supports sequences. Sequences generate a set of numbers which can be used in columns as unique identifiers. An important difference between Access counters and Oracle sequences is that trigger code is required in Oracle to place a sequence number in a column when a new record is inserted into a table.

When Jet has an attachment to an Oracle table and an Oracle trigger changes or initializes the key values at the time of an insert (*not updates*), Jet will perform a sequence of queries to retrieve the new key value so that the inserted row can become a member of the Dynaset. If Jet has trouble re-selecting the inserted row the rows appears as #DELETED to the user.

The example below shows how to emulate a counter datatype in Oracle. The Oracle table is defined as:

```
CREATE TABLE        OTBLCOUNTERTEST(
        PK          NUMBER          (10,0),
        NAME        VARCHAR2        (50),
        CONSTRAINT  PK_OTBLCOUNTERTEST PRIMARY KEY (PK))
```

An Oracle sequence is defined as:

```
CREATE SEQUENCE   TEST    INCREMENT BY 1    START WITH 1000
```

The trigger code is:

```
Create Trigger TRG_CNT_OTBLCOUNTERTEST
Before INSERT OR UPDATE on OTBLCOUNTERTEST
FOR EACH ROW
DECLARE
  iCounter SCOTT.OTBLCOUNTERTEST.PRIMARYKEY%TYPE;
   cannot_change_counter EXCEPTION;

BEGIN
  IF INSERTING THEN
    SELECT TEST.NEXTVAL into iCounter FROM dual;
    :new.PRIMARYKEY := iCounter;
  END IF; -- End of Inserting Code

  IF UPDATING THEN
    -- Do not allow the PK to be changed.

    IF NOT(:new.PRIMARYKEY = :old.PRIMARYKEY) THEN
      RAISE cannot_change_counter;
    END IF;

  END IF; -- End of Updating Code

EXCEPTION
   WHEN cannot_change_counter THEN
     raise_application_error(-20000,'Cannot Change Counter Value');
END;
```

This trigger emulates the counter datatype by trapping both INSERT and UPDATE operations on a table. On any insert the trigger will get the next value in the sequence “TEST” for the PRIMARYKEY column. On updates, the trigger checks to see if the user is trying to update the counter; if so, an exception is raised and the error is passed back to Access.

It is not recommended to silently protect the counter on update. For example, with the following code:

```
IF UPDATING THEN
  -- Do not allow the PK to be changed.

  IF NOT(:new.PRIMARYKEY = :old.PRIMARYKEY) THEN
    :new.PRIMARYKEY := :old.PRIMARYKEY);
  END IF;
```

```
END IF; -- End of Updating Code
```

Jet becomes confused in its management of the Dynaset and produces strange results to the user.

As a possible enhancement to strict counter field emulation, one could use the following code in the trigger to allow Access to pass a value for the counter on a row insert:

```
IF INSERTING THEN
  IF (:new.PRIMARYKEY IS NULL) THEN
    SELECT test.NEXTVAL into iCounter FROM dual;
    :new.PRIMARYKEY := iCounter;
  END IF:
END IF; -- End of Inserting Code
```

This code will generate a new counter value only if the passed value is NULL.

Name Mapping Query

It is easy to build a name mapping query in Access. Use either the QBE or SQL window to define the query. In this example, the original Access table is called SeqDateTable and is exported to Oracle as O_SEQDATETABLE. After the export the table is attached to Jet as R_SeqDateTable.

When the following query is saved as SeqDateTable, it will take the place of the original table and complete the mapping to Oracle. The query maps the column names PRIMARYKEY, O_SEQUENCE and FIRSTDATE to PrimaryKey, Sequence and FirstDate for use by Access.

```
SELECT      NameMapper.PRIMARYKEY AS PrimaryKey,
            NameMapper.O_SEQUENCE AS Sequence,
            NameMapper.FIRSTDATE AS FirstDate
            FROM R_SEQDATETABLE;
```

Default Values

Oracle supports declarative default values. However, when moving an application from Access to Oracle, you may encounter situations where you need an insert trigger to support defaults. A reasonable design decision is to move all default processing to triggers to centralize the code and reduce maintenance complexity. The following code sample demonstrates supporting default values in a trigger:

```
CREATE OR REPLACE TRIGGER BIU_M2
 BEFORE INSERT OR UPDATE
 ON M2
 FOR EACH ROW

  BEGIN
    IF INSERTING THEN
    /* Manage Default Values if a new value is NULL */
      IF :new.Address IS NULL THEN
        :new.Address := 'Default';
      END IF;
    END IF; -- End of Inserting Code
  END; -- Trigger BI_M2
```

Column and Table Validation

Oracle supports CHECK statements which can be used to enforce Table and Column constraints. However, when moving an application from Access to Oracle, you may encounter situations where you need an insert trigger to support validation. The following code sample demonstrates supporting default values in a trigger. Notice that <Access Validation Code > indicates where you can insert the validation code from an Access Application[1].

```
CREATE OR REPLACE TRIGGER BIU_M2
 BEFORE INSERT OR UPDATE
 ON M2
 FOR EACH ROW
```

[1] Access and Oracle support different functions. {*** Geeta - do we mention your function package?}

```
BEGIN
    -- Validation Code
    IF NOT ( <Access Validation Code > ) THEN
       raise_application_error (-20000, '<Access Error Message>');
    END IF;
END; -- Trigger BI_M2
```

Cascade Update Trigger Code

Oracle does not provide direct support for Cascade Update referential integrity constraints. Cascade Update support means that when a Primary Key is changed, that change will be made to all associated Foreign Keys in linked tables. It is not surprising that Oracle does not support this capability, as it not a common design feature in applications. Primary Keys are supposed to be stable, usually for the life of an application.

The following code example is based on two tables:

```
create table M1 (
                    f1 number,
                    f2 number,
                    f3 number )

create table M2 (f1 number,
                    f2 number,
                    f3 number )

alter table M1 add primary key (f1)

alter table M2 add primary key (f1)
```

This definition will support one–to–many cardinality. To add support for one–to–one cardinality add the following:

```
alter table M1 add constraint uq_M1_001 unique (f2, f3)

alter table M2 add constraint uq_M2_001 unique (f2, f3)
```

The following code implements update cascade code for the two tables, M1 and M2. Not that this example uses two columns in the Primary / Foreign Key relationships. This is more complex than most relationships and is used to fully illustrate the proper code.

Please note that declarative and procedural support for RI cannot coexist between two tables. To support Cascade Update between two tables, all declarative Primary / Foreign Key relationships and RI between the tables must be removed and supported instead with procedural code.

```
CREATE OR REPLACE PACKAGE P_M1 AS
   fire_trigger boolean := TRUE;
END P_M1;

CREATE OR REPLACE PACKAGE P_M2 AS
   fire_trigger boolean := TRUE;
END P_M2;

CREATE OR REPLACE PACKAGE UQ_M1_M2 AS

PROCEDURE cascade_update (
       o_F2    IN  number,
       o_F3    IN  number,
       n_F2    IN  number,
       n_F3    IN  number,
       bResult OUT boolean );

      PROCEDURE cascade_delete (
       F2      IN  number,
       F3      IN  number,
       bResult OUT boolean );

     FUNCTION pk_exists (
       F2      IN  number,
       F3      IN  number) RETURN boolean;

     FUNCTION fk_exists (
       F2      IN  number,
       F3      IN  number) RETURN boolean;

END  UQ_M1_M2;
```

```
CREATE OR REPLACE PACKAGE BODY UQ_M1_M2 AS

 /* Procedure cascade_update is called when field(s) */
 /*   F2 or                                          */
 /*   F3                                             */
 /*   are changed in table M1.                       */
 /*    The changes are cascaded in table M2          */

 PROCEDURE cascade_update (
         o_F2    IN  number,
         o_F3    IN  number,
         n_F2    IN  number,
         n_F3    IN  number,
         bResult OUT boolean ) IS

 CURSOR d_cur (n1 number, n2 number) IS
   SELECT * FROM m2
    WHERE f2 = n1 AND f3 = n2
    FOR UPDATE of f2, f3;

 BEGIN

    FOR d_cur_rec IN d_cur ( o_F2, o_F3 )
    LOOP
      UPDATE M2 SET f2 = n_F2, f3 = n_F3
        WHERE CURRENT OF d_cur;
    END LOOP; -- Detail Record Loop

    bResult := true;

 END cascade_update;

 /* Procedure cascade_delete is called when a record */
 /*   in M1 is being deleted and associated          */
 /*   child records in M2 must also be deleted.      */

 PROCEDURE cascade_delete (
        F2      IN  number,
        F3      IN  number,
        bResult OUT boolean ) IS

 CURSOR d_cur (n1 number, n2 number) IS
   SELECT * FROM m2
    WHERE f2 = n1 AND f3 = n2
    FOR UPDATE;

 BEGIN

    FOR d_cur_rec IN d_cur ( F2, F3 )
    LOOP
      DELETE FROM M2
        WHERE CURRENT OF d_cur;
    END LOOP; -- Detail Record Loop

    bResult := true;

 END cascade_delete;

 /* Procedure pk_exists is called to determine is a given
    primary key exists in table M1                       */

 FUNCTION pk_exists (
      F2      IN  number,
      F3      IN  number) RETURN boolean IS

 l_F2        number;
 l_F3        number;
 bResult     boolean;

 CURSOR p_cur (n1 number, n2 number) IS
   SELECT F2, F3 FROM m1
    WHERE f2 = n1 AND f3 = n2;

 BEGIN
   OPEN p_cur( F2, F3 );
   FETCH p_cur INTO l_F2, l_F3;
    IF p_cur%NOTFOUND THEN
      bResult := false;
   ELSE
      bResult := true;
```

```
    END IF;

    CLOSE p_cur;

    RETURN( bResult );

  END pk_exists;

  /* Procedure pk_exists is called to determine is a given
     primary key exists in table M1                        */

  FUNCTION fk_exists (
       F2      IN  number,
       F3      IN  number) RETURN boolean IS

  l_F2         number;
  l_F3         number;
  bResult      boolean;

  CURSOR d_cur (n1 number, n2 number) IS
    SELECT F2, F3 FROM m2
     WHERE f2 = n1 AND f3 = n2;

  BEGIN
    OPEN d_cur( F2, F3 );
    FETCH d_cur INTO l_F2, l_F3;
     IF d_cur%NOTFOUND THEN
       bResult := false;
    ELSE
       bResult := true;
    END IF;

    CLOSE d_cur;

    RETURN( bResult );

  END fk_exists;

END UQ_M1_M2;


CREATE OR REPLACE TRIGGER AUD_M1
AFTER UPDATE OR DELETE
ON M1
FOR EACH ROW

  DECLARE
  bResult_OK      BOOLEAN;
  bCascadeDeletes BOOLEAN  := TRUE;

  BEGIN

     IF UPDATING THEN
       IF (:old.F2 <> :new.F2) OR (:old.F3 <> :new.F3) THEN
          P_M2.fire_trigger := FALSE;
          UQ_M1_M2.cascade_update( :old.F2, :old.F3, :new.F2, :new.F3,
                                   bResult_OK );
          P_M2.fire_trigger := TRUE;
       END IF;
     END IF; -- End of Updating Code

     IF DELETING THEN
        IF bCascadeDeletes THEN
          UQ_M1_M2.cascade_delete( :old.F2, :old.F3, bResult_OK );
       ELSE
          IF UQ_M1_M2.fk_exists( :old.F2, :old.F3 ) THEN
             raise_application_error( -20000, 'Rows exist in child table');
          END IF;
       END IF;
     END IF; -- End of Deleting Code

  END; -- Trigger AUD_M1


CREATE OR REPLACE TRIGGER AIU_M2
AFTER INSERT OR UPDATE
ON M2
FOR EACH ROW

  DECLARE
  bResult_OK    BOOLEAN;
```

```
BEGIN

  IF INSERTING THEN
    IF NOT( UQ_M1_M2.pk_exists( :new.F2, :new.F3 ) ) THEN
       raise_application_error (-20000, 'No corresponding row in parent
              table');
    END IF;
  END IF; -- End of Inserting Code

  IF ( UPDATING AND P_M2.fire_trigger ) THEN
    IF NOT( UQ_M1_M2.pk_exists( :new.F2, :new.F3 ) ) THEN
       raise_application_error (-20000, 'No corresponding row in parent
              table');
    END IF;
  END IF; -- End of Updating Code

END; -- Trigger AUD_M2
```

Appendix 3 - Jet Architecture Details

The Jet DBMS Engine is Microsoft's client and file server RDBMS technology. It is not sold separately, but is included in all Access versions, Visual Basic version 4.0 and Visual C++ V 4.0[1]. Jet is implemented in a set of DLLs. Figure 5 shows how Jet integrates with Access (and Microsoft Visual Basic).

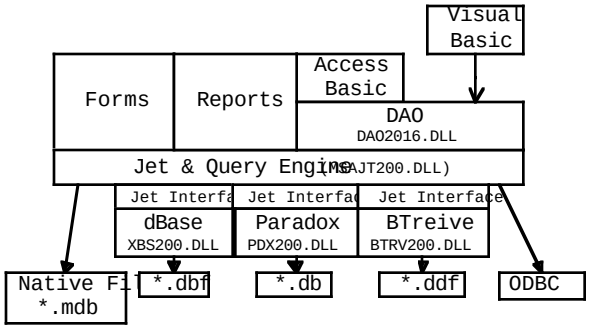

Figure 5 - Access / Jet Architecture Block Diagram[2].

The Data Access Objects DLL (DA02016.DLL) provides a hierarchy of classes (DAOs) to Access Basic and Visual Basic. DAOs define and expose Databases, Workspaces, Query Definitions, Parameters, Recordsets, Tables, Fields, Indexes, Relationships, Users and Groups from Jet.

Links to external data sources are managed by Jet. Links to dBase (XBS200.DLL), Paradox (PDX200.DLL) and Btrieve (BTRV200.DLL) are made through an internal Jet ISAM driver interface. These DLLs are included with the Access product. Jet also supports a link to ODBC through which a wide range of DBMS servers can be reached[3].

Jet Datatypes

DataType	Description	Min	Max
Text	Stores variable length text	1	255
Memo	Large variable length text	1	1.2 Gigabytes
Number: Byte	1 Byte Storage	0	255
Number: Integer	2 Bytes Storage	-32,768	32767
Number: Long Integer	4 Bytes Storage	-2,147,483,648	2,147,483,647
Number: Single	4 Bytes Storage	-3.4×10^{38}	3.4×10^{38}
Number: Double	8 Bytes Storage	-1.8×10^{308}	1.8×10^{308}
Currency	8 Bytes Storage - Monitory Values	-999,999,999,999,999.9999	999,999,999,999,999.9999

[1] Starting with C++ version 4.0, Microsoft is including the Jet Engine with its C and C++ compiler. The interface to the Jet DLL is proprietary. However, access to the DAOs is supported through a set of new MFC classes in MFC version 4.0.

[2] This diagram shows Jet as shipped with Access 2.0 and Visual Basic 3.0. Access 7.0 and Visual Basic 4.0 support Jet version 2.5 (a 16 bit version) and Jet version 3.0 (a 32 bit version).

[3] There are ODBC drivers for dBase and Paradox. These are different from the Jet ISAM driver for dBase and Paradox. One could connect Access to dBase via ODBC (Jet D ODBC D dBase) but this would not be efficient.

Counter[4]	4 Bytes - AutoIncrement Field	0	2,147,483,647
Yes/No	1 Bit Storage - Boolean value		
Date/Time	8 Bytes Storage		
OLE	OLE, graphics other complex data	1	1.2 Gigabytes
Binary	Binary Data	1	1.2 Gigabytes

Table 1

[4] A counter type is automatically incremented for each row insertion by the Jet engine. There is no way to start or stop the assignment of values and their is no way to recover used (but missing) values.

Jet Query Processor

Jet's query processor does not support a full implementation of SQL. It does, however, do a very good job of optimizing queries, especially when the query references both local and remote tables. Jet can connect to a wide range of data sources and process queries against all of them. Transaction support is limited to native file format database table. Jet relies on the transaction support of any RDBMS attached via ODBC.

Jet Transactions

Jet supports an explicit transaction model. Transactions are not started until a BeginTrans statement is executed. Transactions are committed with CommitTrans and aborted with Rollback. In addition to using transactions to group related units of work, developers use transactions to improve performance. If a program makes a lot of references to a table, grouping the work in a single transaction will force Jet to perform the operation in memory and then commit all work to disk when the transaction is committed. Be aware that this type of transaction use may not map directly to Oracle.

Appendix 4 - Oracle 7.1 RDBMS Architecture Details

Oracle 7.1 is a powerful, flexible, and scaleable RDBMS server which runs on a range of computers systems from a personal computer to the largest mainframes. Oracle 7.1 has been designed to run effectively in a client / server environment and supports hundreds to thousands of users.

The Oracle 7.1 architecture supports advanced server features such as record locking with versioning (not page locking as provided by Access, SQLServer and Sybase), advanced query optimization, the PL/SQL programming language, data replication, distributed database management and other important features.

The architectural features discussed here are only a few of the features found in Oracle 7.1 and are focused on the elements that pertain to working with Microsoft Access. A complete description of the Oracle 7.1 architecture can be found in the following Oracle7 Server Manuals:

PL/SQL User's Guide and Reference
Oracle7 Server Administrator's Guide
Oracle7 Server Application Developer's Guide
Oracle7 Server Concepts Manual
Oracle7 Server Messages and Codes Manual
Oracle7 Server SQL Language Reference Manual
Oracle7 Server Utilities User's Guide
Oracle7 Server Distributed Systems: Replicated Data
Oracle7 Server Manager User's Guide
Oracle7 Server Messages and Codes Manual
Oracle7 Parallel Server Administrator's Guide
Oracle7 Server Utilities User's Guide
Trusted Oracle7 Administrator's Guide

These documents are available in electronic form on a CD from Oracle Corporation.

Computer and Operating Systems Supported

At the time of the writing of this paper, versions of Oracle are available for Windows 3.1, Windows 95 and Windows NT as well as operating systems on larger computers. Please contact Oracle for the latest list of versions and operating system support.

Oracle Datatypes

Oracle tables supports the following datatypes:

DataType	Description	Min	Max
CHAR(size)	Fixed length character string <size> long	1	255
DATE	Date and Time	Jan 1, 4712 BC	Dec 31, 4712 AD
LONG	Variable length character string	1	2 Gigabytes
LONG RAW	Binary data	1	2 Gigabytes
NUMBER(p,s)	Number having precision p and scale s[1]		
	p	1	38
	s	-84	127
RAW(size)	Binary data	1	2000
ROWID	Values from the pseudo-column ROWID		
VARCHAR2(size)	Variable length character data	1	2000
VARCHAR(size)	Same as VARCHAR2		
FLOAT	Special Datatype provided for ANSI and IBM Compatibility. FLOAT provides 38 digits of storage		

Table 2

Triggers and Stored Procedures

Oracle allows code to be written and stored in the DBMS along with data. A trigger is code which is executed in response to an event in the database. Trigger code can be associated with an Update, Insert or

[1] Precision, p, refers to the number of digits stored. Scale, s, is the scaling factor or 10^s multiplier. For example, a NUMBER(3,-1) refers to numbers of the form “XX.X”.

Delete event for each row or for a table as a whole. In addition, a trigger can be set to run just before the event or just after (e.g., a trigger can be set to run after any row is updated).

A stored procedure is a general routine (either function or subroutine) which is stored in pre-compiled form on the server.

A trigger may call stored procedures, but triggers are only activated by specific database activity (such as the insertion of a row in a table).

When using Access with Oracle, triggers and stored procedures play a role in mapping the functionality of Access to Oracle, such as in the support for the Access datatype 'counter' on Oracle.

PL/SQL Programming Language

The PL/SQL Programming Language is an ALGOL-based language (like Pascal). PL/SQL is a modern, full featured programming language with exception handling. PL/SQL is used to write stored programs and triggers within an Oracle 7.1 RDBMS server. It is also used as the programming language in many of Oracle's client-side tools (e.g., Forms 4.5 from the Developer/2000 collection).

Sequences

A Sequence is a unique number generator which is implemented in shared memory on a server. It is designed to provide a set of unique values for PL/SQL programs for use as Primary Keys. Sequences are designed for high performance applications which might otherwise 'single-thread' on table based unique number generator.

Sequences, along with supporting code in a trigger are used to emulate the Counter field type in Access.

Transactions

Unlike Access, Oracle supports an implicit transaction model. Each SQL statement is part of a logical transaction. A logical transaction begins with the first SQL statement and ends with a Commit or Rollback statement. Immediately after either of these statements, a new transaction takes effect with the next SQL statement.

As mentioned in the Jet Architecture Appendix, Access developers use transactions to improve the performance of Jet. Grouping database statements in a transaction forces Jet to attempt to complete all database work in memory; Jet will defer writing to disk until the transaction is committed. When this use of transactions is mapped to Oracle via ODBC, Jet will send only the outer most pair of Begin / Commit transaction requests. This will cause Oracle to keep an open transaction during the entire processing period. You must decide if this is the desired outcome when moving from Access to Oracle.

Other Oracle Features

A Database administrator has great flexibility when configuring Oracle. Data can be written on multiple disks for increased performance, rollback and recovery options can be tuned, are computer resources can be allocated to optimize the configuration for each server. Oracle also supports distributed processing, so data can be distributed across multiple machines. Oracle also offers a Trusted Oracle Server version for applications that require a higher level of user and use authentication.

Appendix 5 - ODBC Architecture Details

Open Database Connectivity (ODBC) is an industry standard programming interface (API) designed to allow a client application to connect to and use different data sources. Just as the Windows Printer Interface masks the differences between printers and allows a Windows application to write to a single printing API, ODBC masks differences between databases. Applications can rely on a single interface.

Unlike printers however, databases can offer a very large range of capabilities. Because of this, there are separate levels of compliance with the ODBC standard[1]:

> ODBC defines three levels of API compliance: Core, Level 1 and Level 2.
>
> ODBC also defines levels of SQL support; these are Minimum, Core, Extended.

ODBC's strength lies in its ability to allow a single client application to connect to different data sources without recompilation. The user of an application simply changes the data source driver and the application will access data from a different data source. For example, Access uses ODBC to connect to Oracle, Sybase, rDB and other DBMS servers.

ODBC does have limitations. Because ODBC defines different levels of support, the developer of a client application must select the level of support from ODBC. The more support that is required, the smaller the set of database drivers that can satisfy the requirements.

Because ODBC defines its own SQL grammar and then translates this grammar to the host dialect of SQL, it is possible that an advanced RDBMS server will offer features in SQL that do not correspond to the ODBC SQL grammar.

ODBC also defines a set of data types and performs data type mapping between the ODBC set and the RDBMS server's native data types. This data type mapping can take additional processing time.

Finally, ODBC supports a limited transaction model.

ODBC Layers

ODBC is a layered architecture with two levels - the ODBC driver manager and the individual ODBC datasource drivers. Figure 6 shows ODBC's layered architecture.

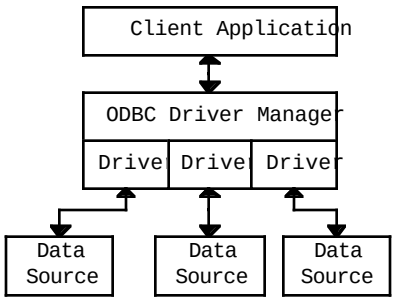

Figure 6 - ODBC Architecture

A client application calls on the ODBC Driver Manager for services. Some operations (such as loading a data source driver) are handled by this layer. Other operations are passed to the ODBC Driver for processing.

An ODBC driver can take on one of three forms: single tier, two tier and three tier. A single tier driver contains all of the logic needed to access a datasource (such as a native file format). Examples of single tier

[1] Refer to "Microsoft ODBC 2.0 Programmer's Reference and SDK Guide", published by Microsoft Press for complete information on the ODBC standard.

drivers include dBase and Paradox ODBC drivers. Single tier drivers execute completely on the client computer.

A two tier driver can perform some operations on the PC, but generally passes requests for processing to a DBMS server. Oracle's Oracle 7.1 ODBC driver is a two tier driver.

A three tier driver connects to a gateway where an additional driver at that location makes the connection to the ultimate DBMS. An example of a three tier driver is a connection via a gateway to DB2 on an IBM MVS system.

ODBC drivers are offered by Microsoft, Oracle, Visigenic, Intersolve and others.

ODBC Transactions

The ODBC definition supports two types of transaction models, "auto–commit" and "manual–commit". With the default, auto–commit, each SQL statement is a committed transaction. In manual–commit mode, when the application submits an SQL statement and a transaction is not active, then one is started. The transaction remains open until an explicit commit or rollback is requested. The auto–commit mode matches Access and SQLServer, the manual–commit mode matches Oracle's transaction model.

In addition, the Jet / ODBC combination supports only a single level of transactions.

Jet ODBC Driver Requirements

Jet is designed to work with ODBC drivers which are Level 1 API compliant. According to the "Jet Database Engine 2.0 - ODBC Connectivity White Paper" by Neil Black and Stephen Hecht, published by Microsoft, Jet uses the following APIs from ODBC:

SQLAllocConnect	SQLFetch	SQLParamData
SQLAllocEnv	SQLFreeConnect	SQLPrepare
SQLAllocStmt	SQLFreeEnv	SQLPutData
SQLCancel	SQLFreeStmt	SQLRowCount
SQLColumns	SQLGetData	SQLSetConnectOption
SQLDescribeCol	SQLGetFunctions	SQLSetParam
SQLDisconnect	SQLGetInfo	SQLSetStmtOption
SQLDriverConnect	SQLGetStmtOption	SQLSpecialColumns
SQLError	SQLGetTypeInfo	SQLStatistics
SQLExecDirect	SQLMoreResults (optional)	SQLTables
SQLExecute	SQLNumResultCols	SQLTransact

Oracle's Oracle 7.1 ODBC Driver

Oracle's Oracle 7.1 ODBC driver is a two-tier, Level 1 API, minimum SQL level compliant driver[1]. Figure 7 show the ODBC architecture with Oracle's Oracle 7.1 ODBC driver. The Oracle 7.1 driver consists of the Oracle ODBC driver and Oracle OCI[2]; Oracle OCI relies on SQL*Net[3] to make a connection to an instance of Oracle 7.1 RDBMS.

[1] For detailed information, refer to drvora71.hlp located in your Windows / System subdirectory.

[2] Oracle's Call Level Interface. This is the lowest level API client side interface available for connecting to an Oracle Server.

[3] SQL*Net is a network transport product available from Oracle Corporation. SQL*Net relies on network software such as TCP/IP.

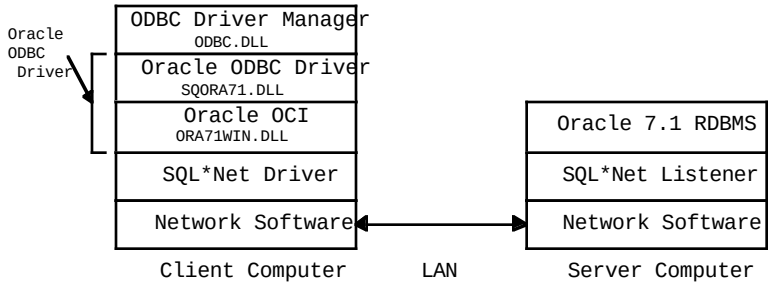

Figure 7 - Oracle ODBC Driver Architecture

Table 3[1] shows the mapping in the Oracle ODBC driver between ODBC datatype and Oracle datatypes.

Oracle Datatype	ODBC Datatype
CHAR	SQL_CHAR
DATE[2]	SQL_TIMESTAMP
FLOAT	SQL_FLOAT
LONG	SQL_LONGVARCHAR
LONG RAW	SQL_LONGVARBINAR Y
NUMBER(p,s)[3]	SQL_NUMERIC(p,s)
RAW	SQL_VARBINARY
VARCHAR	SQL_VARCHAR
VARCHAR2	SQL_VARCHAR

Table 3 - Oracle to ODBC Datatype Mapping

[1] From the Oracle 7.1 ODBC driver help file.

[2] Oracle's DATE column type store time to seconds resolution. ODBC's SQL_TIMESTAMP is defined to store fractional seconds to a resolution of a nanosecond (1 billionth of a second).

[3] Oracle's NUMBER supports 17p738 and -847s7127. SQL_NUMERIC support is defined as 17p715 and 07s7p.

Appendix 6 - Jet / ODBC / Oracle Architecture Details

There are two fundamentally different ways to make a connection between Access and Oracle. The first is to connect Jet to Oracle via ODBC. This relies on the power and flexibility of the Jet engine and leverages the capabilities of Access' Form and Reports.

The second method is to use the Basic programming language inside Access as an independent programming environment. One can use Access Basic to call Windows APIs and DLLs such as the ODBC API or Oracle's Oracle Objects for OLE.

The first method, the Jet / ODBC connection will be explored in detail in this paper while the second, using Access Basic to connect to Oracle will discussed only briefly in the next section.

Independent Access Basic Language Connection

Access includes Access Basic, a rich implementation of the Basic programming language. When used as an independent programming language and development environment, one scarifies the rich Forms and Reports capabilities. If one is willing to make this sacrifice there a wide range of connection options:

1] Oracle Objects for OLE (OO4O), is an OLE 2.0 in-process server which provides a connection to Oracle. OO4O provides many of the same capabilities as Jet, including Dynasets[1].
2] ODBC API. The entry points to the ODBC API can be declared within Access Basic and called directly.
3] OCI API. The entry points to the OCI API can be declared within Access Basic.
4] Oracle Mobile Agents can be used to establish radio links to a central database server.

While these alternate techniques can provide some important capabilities and performance gains, the loss of Access' Forms and Reports can make building an application with Access much more difficult. If one needs to use these alternate methods of connecting to Oracle, it is very likely that another client development tool will be easier to use and more productive.

Jet ODBC Connection

By using the Jet / ODBC connection, Access users and developers benefit from the full power of Access. Jet maps external tables through the internals of Jet. Jet presents these external tables as if they were internal tables after performing data type and other mappings. Jet requires ODBC drivers to be Level 1 API compliant.

Jet connects to ODBC tables either through attachments or connections. When Jet connects to a table via ODBC it requires a lot of information from the server. (See "Attached Tables" below for a detailed description of the attachment process). This process can require many seconds per table. Access allows a user to make a static table "attachment" or a dynamic "connection". In almost all cases, it is best to use a static "Attached Table" connection to Oracle tables. Attached tables are visible in the user interface, have their connection information cached inside of Access (which allows for rapid opening of the link) and are fully represented by Jet's DAO as if they were local tables. A dynamic connection requires significant overhead at run time each time a connection is opened. In addition the remote table is only available via the DAOs.

Access can update data in an Oracle table only when that table has a unique index. Without a unique index, Access cannot insert, update or delete data in the table - even with Access bulk queries (passthrough queries will work with non-indexed tables as they are passed directly to Oracle). If an Oracle table does not have a unique index, it can only be opened as a snapshot.

---

[1] Oracle Objects for OLE has implemented Dynasets differently than Jet. Refer to the documentation included in OO4O for details.

When Jet establishes a link to an Oracle table it will inquire about indexes in the following order: Clustered, Hashed, and then other indexes. (In each of these groups, Jet will look at the indexes alphabetically). Jet will use the first index it finds as the keyvalue index. To insure that a particular index is used (and its columns are the keyset values in the Dynaset query), use a name prefixed with “aaaa” or something similar to force the index to be the first found in alphabetical order.

Attached Tables (draft)

A table attachment is a static caching of information inside of Access about a remote table connection. An attachment can be made through the Access user interface (File...Attach Table) or through code. The following subroutine will create an attachment to an external table:

```
Sub Create_Table_Attachment ()
   Dim wk As WorkSpace
   Dim db As Database
   Dim td As TableDef
      '
   '   Setup a  workspace and get the current database
   '
    Set wk = dbengine.workspaces(0)
    Set db = wk.databases(0)
   '
    '   Create an attached table named "Remote Oracle Table"
   '
    Set td = db.CreateTableDef("Remote Oracle Table")
   '
    '   Connect the attached table to an Oracle Database through
    '   the ODBC connection named "Scott". This is connected to an
    '   Oracle database which has the password "Tiger".
   '
    '   Point the connection to the Oracle table named OTBLISMALLNAMES
    '   owned by the user "Scott".
   '
   td.Connect = "ODBC;DSN=scott;DBQ=2:;PWD=Tiger;"
   td.sourceTableName = "SCOTT.OTBLISMALLNAMES"
   '
    '   Append the table attachement definition to Jet,
    '   making this a permanent connection.
   '
    db.tabledefs.Append td
   '
    '   Close and release the resources
   '
   Set td = Nothing
   db.Close
   Set db = Nothing
   wk.Close
   Set wk = Nothing
End Sub
```

When Jet attaches a table, it retrieves and caches the following information:

1] ODBC Driver behavior, including transaction capabilities, cursor commit behavior, and cursor rollback behavior.
2] A means to determine if a row has changed. This is needed for the optimistic locking scheme. If a timestamp column is available on the server table, Jet will use this one column. If a timestamp column is not available, Jet will use *all* of the columns in the row to determine if changes have been made during an edit cycle.
3] Jet retrieves information on all columns in the table.
4] Jet retrieves statistics on the table which it uses in its cost based query optimizer.
5] Jet inquires about the availability of functions on the server[1] (String, Number, Timedate, system, and conversion functions). Jet also inquires about the level of SQL supported (outer joins, expressions in order by clauses) and the server concatenation behavior with NULLs.

[1] For an exact list of the capabilities of the Oracle ODBC driver, refer to the driver *.hlp file.

If significant[1] changes are made to the table within the server, a table attachment must be refreshed. This can be done with that Attachment Manager (File...Add Ins...Attachment Manager) or via the RefreshLink method of a TableDef object.

Connected Tables

A dynamic connection to a table can be made at run time through Access Basic code. The following subroutine attaches a table from Oracle and then disconnect the table:

```
Sub Connect_Table ()
   Dim wk As WorkSpace
   Dim db As Database
   Dim rs As Recordset
   '
   '   Setup a  workspace and get the current database
   '
    Set wk = dbengine.workspaces(0)
    Set db = wk.databases(0)
   '
   '   Open the database connection to Oracle
   '
    Set db = wk.OpenDatabase("", 0, 0, "ODBC;DSN=Scott;DBQ=2:;UID=Scott;PWD=Tiger")
   '
   '   Open the recordset as a Dynaset
   '
    Set rs = db.OpenRecordset("OTBLSMALLNAMES", DB_OPEN_DYNASET)
   '
   '   Close and release the resources
   '
   rs.Close
   Set rs = Nothing
       db.Close
   Set db = Nothing
       wk.Close
   Set wk = Nothing

End Sub
```

When this connection is made, Access does essentially the same work that it does when creating a table attachment. Dynamic connections should only be used when a static table attachment is not wanted. For example, if one did not want to show a remote table attachment through the Access user interface, one would use a dynamic connection.

Opening an Oracle Table

Once Jet knows how to link to an Oracle table, the table can be opened as either a Snapshot or a Dynaset.

Dynasets

As described earlier, Microsoft Access uses Dynasets to manage editable data. In order to open a Dynaset on an Oracle table that table must have a unique index.

Figure 8 shows Access with an open Dynaset on an Oracle table. A keyset and a data query are both needed to implement a Dynaset. The keyset query is run once to obtained a set of keys for the data query[2].

In the example in Figure 8, Access has retrieved a keyset before “905,New,Person” was inserted into the server. Therefore keyvalue 905 is not available to Access. Also notice that Access has retrieved two full rows of data, Bob Carlin and Ellen Jones. Also notice that Ellen Jones was subsequently changed to Ellen Merrill on the server. Access will not see this change until a refresh is requested or the user attempts to edit the Ellen Jones record.

---

[1] Significant changes include: changes to the structure of the table, the name of the table, a significant change in the number of rows stored in the table, the addition or deletion of indexes, etc.

[2] Jet does not user Oracle ROWIDs.

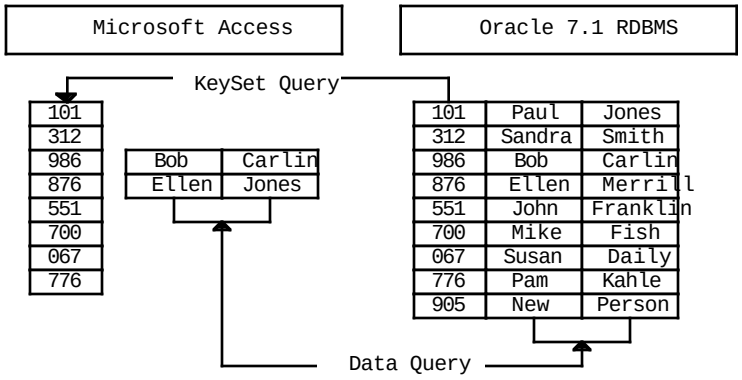

Figure 8

When a user edits data in Access, Jet performs the following steps:

1] Jet re-reads the values in the row to be edited and will refresh the information for the user if it has changed since the previous fetch. (e.g., Ellen Jones would be updated to Ellen Merrill)
2] The user is free to edit the data.
3] To post the changed information, Jet again reads the row from the server to determine if someone else has changed the row while the edit was taking place. If the row has not been changed during the edit cycle, the posting is allowed, otherwise the user is notified that the data has changed.

Dynasets in Practice

The following describes Access' exact observed behavior when opening a Dynaset with Oracle.

The Oracle is defined as:

```
CREATE TABLE        OTBLISMALLNAMES (
        PK          NUMBER          (10,0),
        FIRSTNAME   VARCHAR2        (50),
        LASTNAME    VARCHAR2        (50),
        CONSTRAINT  PK_OTBLISMALLNAMES PRIMARY KEY (PK))
```

It was attached to Access with a resulting table defined as:

PK	Number	Double
FIRSTNAME	Text	50
LASTNAME	Text	50

The table OTBLISMALLNAMES contains 1000 rows.

When Jet opened a Dynaset on the table, the following sequence of events occurred:

1] KeySet query is opened and the first 102 key values are fetched.
2] A new query is opened to fetch 19 rows of complete information (this is to allow Access to quickly fill the visible portion of the datasheet view. After this fetch, this query is closed.
3] KeySet query retrieves keys for rows 103 to 202.
4] The Data query is set up to retrieve 10 rows at a time. (The query is prepared once and executed each time rows of data are needed for display).
5] Access fetches complete rows with the Data query for rows 20 to 99 in batches of 10 rows each.
6] Keyset query retrieves keys for rows 203 to 302.
7] A new query is used to retrieve full data for rows 100 and 101 (a 'row fixup query').
8] Keyset query retrieves key values for rows 303 to 1000 and Access can now report the total number of rows in the table in the datasheet view.

Once the table has been opened, Jet supports a user scrolling through the data. When the user moves to a new location Jet will retrieve the full row values for the target row and 50 rows on either side of this row (100 row cache). If the user scrolls back to a portion of the table which has already been visited, the rows must be retrieved again as they are not kept in a local cache in Access.

From this observed behavior, it becomes clear that while scrolling is a powerful feature for the end user, it has a high cost in terms of network traffic and load on the server.

Snapshots

A snapshot is a query which represents data in the server at a particular time. Data cannot be changed with a snapshot query. Snapshots are simpler than Dynaset queries and for smaller tables, they are faster because a single query is used to fetch the information.

However, for large tables where only a part of the data is needed, it may be faster to open a Dynaset, even if the data is not going to be edited, as only the keyset query must complete before the user is given access to the results of the query.

Snapshots in Practice

The following describes Access' exact observed behavior when opening a Dynaset with Oracle.

The Oracle is defined as:

```
CREATE TABLE                OTBLSMALLNAMES (
        PK                  NUMBER          (10,0),
        FIRSTNAME           VARCHAR2        (50),
        LASTNAME            VARCHAR2        (50))
```

It was attached to Access with a resulting table defined as:

PK	Number	Double
FIRSTNAME	Text	50
LASTNAME	Text	50

The table OTBLSMALLNAMES contains 1000 rows.

When Jet opened a snapshot on the table, the following sequence of events occurred:

1] A query is opened and the first 100 complete rows are retrieved.
2] Access waits for a time internal and then retrieves the next 100 records. This continues until all rows in the query are retrieved
3] The query is closed.

Unlike a keyset query, Access will cache all the rows returned from a snapshot. Once information has been retrieved with a snapshot, scrolling and accessing different rows is performed locally and is fast.

Mappings

This section explores all of the significant mappings that occur between Oracle, ODBC and Jet.

Data Types

Access, ODBC and Oracle each have their own data types. Jet performs default mapping between its data types and ODBC defined types. The Oracle ODBC driver maps the Oracle datatypes to ODBC datatypes.

Jet / ODBC Datatype Mapping

The following table[1] shows the data type mapping negotiation Jet performs with ODBC. For each Jet datatype there is a prioritized list of preferred datatypes. Since a given ODBC driver may not support all of the possible ODBC datatypes, Jet must determine this mapping for each driver dynamically.

For example, the Access datatype Yes/No will map to SQL_BIT, if that is supported; otherwise to SQL_SMALLINT, if that is supported, etc. There will always be a datatype which can accept the Access datatype.

Access Datatype	**ODBC Datatype**					
Yes/No	SQL_BIT	(else)	SQL_SMALLINT	(else)	SQL_INTEGER	(else) SQL_VARCHAR(5)
Number: Byte or Integer	SQL_SMALLINT	(else)	SQL_INTEGER	(else)	SQL_VARCHAR(10)	
Number: Long Integer	SQL_INTEGER	(else)	SQL_VARCHAR(20)			
Currency	SQL_DECIMAL(19,4)[2]	(else)	SQL_FLOAT	(else)	SQL_VARCHAR(30)	
Number: Single	SQL_REAL	(else)	SQL_FLOAT	(else)	SQL_VARCHAR(30)	
Number: Double	SQL_FLOAT	(else)	SQL_VARCHAR(40)			
DateTime	SQL_TIMESTAMP	(else)	SQL_VARCHAR(40)			
Text(size)	SQL_VARCHAR(min(size, ServerMax))					
Binary(size)	SQL_VARBINARY(min(size, ServerMax)					
Memo	SQL_LONGVARCHAR(ServerMax)			(else)	SQLVARCHAR(2000)	

Table 4 - Jet / ODBC Datatype Mapping Negotiation

[1] This table is from "Jet Database Engine ODBC Connectivity White Paper" by Neil Blank and Stephen Hecht of Microsoft.

[2] This is only supported for SQLServer or Sybase.

Jet / ODBC / Oracle Datatype Mapping

The following table shows the results of Jet's negotiation with ODBC for datatype mappings as it carries out with the Oracle ODBC driver. The datatypes on the left are the datatypes (and ranges of the types) and the Jet / Access datatypes that they get mapped to.

Oracle Datatype		Access Datatype	
number	(1-4,0)	Number:	Integer
number	(5-9,0)	Number:	Long Integer
number	(10-15,0)	Number:	Double
number	(16-38,0)	Text	(255)
number	(1-15,n)	Number:	Double
number	(16-38,n)	Text	(255)
float		Number:	Double
date		Date/Time	
char	(n)	Text	(n)
raw	(n)	Binary	(n)
long		Memo	
longraw		OLE Object	
rowid		Text	(18)
varchar	(n)	Text	(n)

Table 5 - Oracle / Access Datatype Mapping

The following table illustrates what happens when a table, which is originally defined in Access is exported to Oracle and then attached as a remote table inside of Access. Notice that the data types do not always end up in the attached table as they originally started in the internal Access table. This is due to both datatype mismatching between Access, ODBC and Oracle as well as the design decision to make the Jet datatypes mappings conservative (very unlikely to loose data or precision).

Access Datatype		Export D	Oracle Datatype		Attachment D	Access Datatype	
Yes/No			number	(5,0)		Number:	Long Integer
Number:	Byte		number	(5,0)		Number:	Long Integer
Number:	Integer		number	(5,0)		Number:	Long Integer
Number:	Long Integer		number	(10,0)		Number:	Double
Number:	Single		float			Number:	Double
Number:	Double		float			Number:	Double
Currency			float			Number:	Double
Date/Time			date			Date/Time	
Counter			number	(10,0)		Number:	Double
Text[1](n)			varchar	(n)		Text	(n)
Memo			long			Memo	
OLE Object			long raw			OLE Object	

Table 6

If the datatype definitions on the server and the mappings of the datatypes back through Access are important to an application, the only way to affect the mappings is to tune the datatypes on the Server; use Table 5 as your guide.

Transactions

Access uses the default Autocommit mode of ODBC. Because Jet and ODBC only support a single level of transactions, if you group transaction in Access Basic, only the outer BeginTrans / CommitTrans pair is sent to Oracle.

[1] Zero-length text values fetched from Oracle are treated as if a NULL value had been fetched.

Appendix 7 - Access / Oracle Tuning and Customization

This appendix contains suggestions for tuning and customizing the way Oracle and Access work together.

ODBC Driver Selection

There are a number of ODBC drivers available for Oracle 7.1. In addition to the driver supplied by Oracle, drivers are also available from Microsoft, Visigenic, Intersolve and others.

The performance of the drivers can vary. If you are building a large scale application, it will be important to profile the different ODBC drivers with your application. The best way to determine the performance is with ODBC or OCI 'spy' programs. Mercury Software sells ODBC Inspector and SQLInspector for Oracle (an OCI spy program). These programs will show you the calls that are being made by Jet to the ODBC API and will show you the calls the Oracle ODBC driver is making against OCI.

Server based Parameter Table

When Access first makes a connection to an Oracle database it checks to see if a special parameter table is present. The table has the following definition:

```
create table MSysConf (
                CONFIG      NUMBER,
                 nValue     INTEGER
                )
```

The following Config and nValue customize the way Access works with Oracle:

Config	nValue	Meaning
101	0	Do not allow the user to store the USERID and PASSWORD in attachments. This is important to set for secure installations.
101	1	Allow the user to store the USERID and PASSWORD in table attachments (default)
102	D	Access delays D seconds between each background chunk fetch when managing Dynasets
103	N	Access fetches N rows on each background chunk fetch when populating a Dynaset.

It is suggested that you create an MSysConf table in each Oracle database, even if you plan on using the defaults. It will be easier to change the values in the table than to remember how to create and name the table at a later time.

Jet's Query Processor (prelim)

Jet's Query Processor includes a cost based optimizer which can make decisions on joining heterogeneous data via an ODBC connection.

Application startup performance (prelim)

Access can be tuned to speed up the process of establishing an ODBC connection at application startup time. These are made by making entries or changes in the MSACC20.INI file, found in the Windows subdirectory.

When Access opens a connection to an ODBC database, it goes through several steps to determine the level of functionality provided by the particular ODBC driver. Make the following entries under the [ODBC] section of MSACC20.INI:

If you are relying on Oracle 7.1 to provide full security, one can bypass Access' attempt to logon to Oracle using it's own user / group / password information with the following entry:

TryJetAuth = 0

(This will save between 1 and 2 seconds when making the first connection to Oracle)

The Oracle 7.1 ODBC driver does not support an asynchronous connection. Make the following entry to prevent Access from making an inquiry of the driver about its asynchronous capabilities each time Jet uses ODBC.

DisableAsync=1

Run Time performance

The most important issue related to run time performance in a client / server configuration is reducing network traffic.

Form Loading Time

Performance is both perceived and actual. Eliminate any requests for data from Oracle while a form is loading. This can be done by including a button which will retrieve information. Another possibility is to cache the last information a form displayed locally.

Qualified and Restricted Queries

You can reduce network traffic by requesting only the columns you need from a table. You should also use the most restrictive qualifications possible to reduce the size of the query result set.

Snapshots vs. Dynasets

If possible, use Forward Only Snapshots to work with Oracle data, especially when the result set is small. For larger result sets, and for queries which must be updateable, use a Dynaset. Even if you are not going to update data, a Dynaset will be faster than a Snapshot if the result set is large. This is because only the Key values are retrieved for the Dynaset, not the full set of complete rows.

Drop Down Lists

Access tries to minimize the amount of network traffic when it needs to populate a DropDown List box. When a snapshot is used to populate a drop down list, Access uses the same batch fetching of records that it uses to populate a grid or a form. Access will fetch an initial ‘chunk’ of data (100 rows) and then periodically retrieve sets of 100 rows from the server.

This works smoothly unless the user makes an entry which does not match a row already fetched. In this case Access will begin fetching records from the server until a match is found or until all records are retrieved. If the returned set is large this can take a long time and it will freeze the User Interface.

Unfortunately, Access does not share queries for dropdowns. You cannot define a snapshot query and reference it from multiple listboxes expecting to use information which is retrieve once. Access will instead, treat each ‘activation’ of a query as independent.

If a drop down list is short (< 100 records), it is probably sufficient to have Access perform its normal operations. If the list is long, you may want to build synchronized shadow table in Access. That is, store the table information locally in Access and periodically synchronize the local table with information from Oracle.